CAST: Critique-Aware Supervision for Training Reliable Long-Horizon Tool-Calling Agents

Amir Saeidi[†*] **Zehua Zhang**[†*] **Rishitosh Singh**[†] **Naman Ahuja**[†]
Vivek Gupta[†] **Ali Payani**[‡] **Gaowen Liu**[‡] **Jayanth Srinivasa**[‡] **Chitta Baral**[†]

[†]Arizona State University [‡]Cisco Research

Abstract

Large language model (LLM) agents are increasingly deployed in long-horizon, interactive, and stateful environments. In these settings, a single wrong action, such as refunding the wrong purchase, can cause irreversible task failure and must be intercepted before execution. Such failures may not appear in every single run, but can emerge across repeated trials, making reliability across steps and trials critical. However, ensuring agentic reliability is challenging: even frontier LLMs struggle to explain why an action may be wrong, especially in long, intertwined trajectories governed by domain-specific policies. Much recent work relies on prompt-based critique agents, while optimization-based methods lack a systematic way to produce rich verification rationales for training. We address this gap with CAST, a critique-aware training framework that converts sparse task outcomes into action-level supervision for critique learning and policy optimization. CAST analyzes agent trajectories to synthesize structured rationales explaining action validity under partial observability. The resulting critique model is used to construct critique-aware training data for optimizing the policy model. Fine-tuning Qwen3-family models on dynamic tool-calling benchmarks, CAST improves reliability across domains, outperforming GPT-OSS-120B by over 10% pass^4 on Retail tasks and yielding an additional 9% improvement on Telehealth in an out-of-domain setting. These results demonstrate that critique-aware training improves the robustness of LLM agents in realistic dynamic environments.

1 Introduction

Large language models have demonstrated strong capabilities in reasoning (Guo et al., 2025), question answering (Yue, 2025), and instruction following (Zhou et al., 2023). More recently, the agentic paradigm has extended these capabilities to interactive environments, where models use tools, track evolving states, and make decisions over long horizons (Qin et al., 2024). Despite this progress, reliability remains a central challenge for language agents (Yao et al., 2024). In dynamic environments, agents may solve a task in one trajectory but fail in an identical or highly similar scenario due to small changes in observations, tool outputs, or previous actions (Barres et al., 2025). This inconsistency becomes especially problematic in realistic deployments, where each action can affect the subsequent interaction and repeated attempts are often unavailable. The challenge is amplified in longer conversations, where small mistakes can accumulate and cause task failure (Laban et al., 2025).

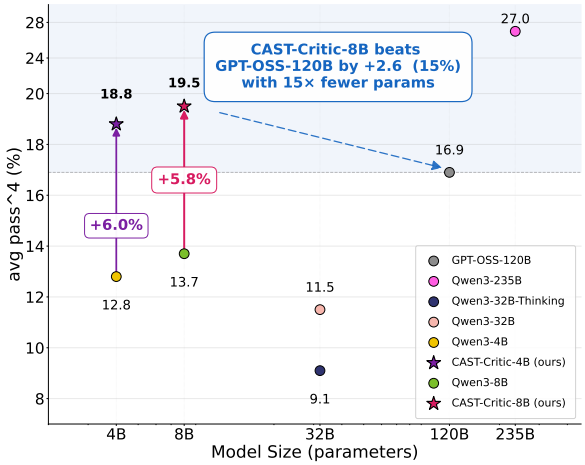

Figure 1: CAST help improve reliability of LLM agents in dynamic environments across trials, which allows smaller models to outperform substantially larger models on reliability-oriented metrics.

To address this inconsistency, recent works (Mishra et al., 2025; Tan et al., 2025; Qian et al., 2025) have explored both inference-time agentic mechanisms and learning-based optimization methods. Complex agentic systems can improve decision making through planning, reflection, or self-correction, but they often intro-

[*]Equal contribution. Correspondence: **ssaeidi1@asu.edu**

Proceedings of the 2026 Conference on Empirical Methods in Natural Language Processing, pages 42006–42034
October 24-29, 2026 ©2026 Association for Computational Linguistics

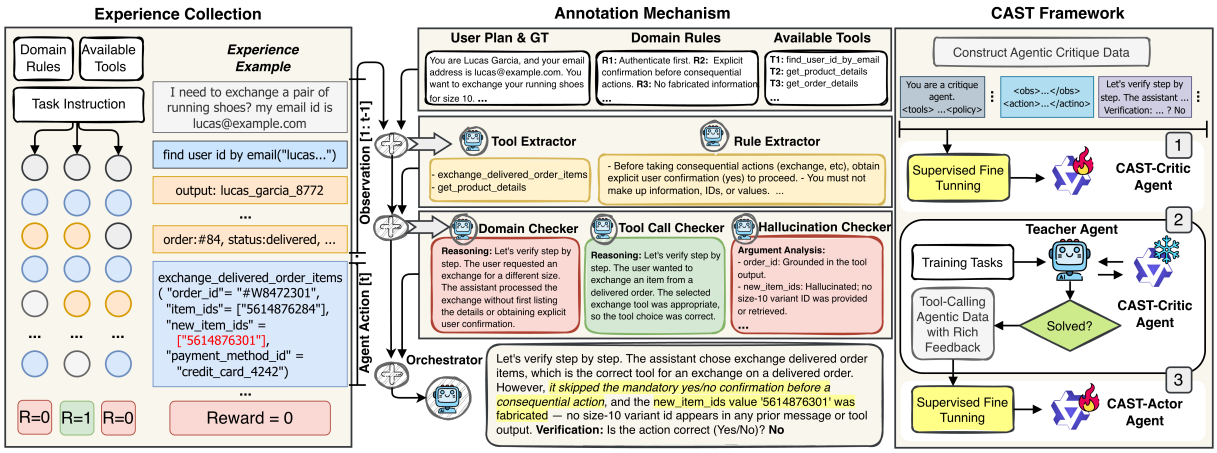

Figure 2: CAST overview pipeline. **Stage 1** collects trajectories by repeatedly running a teacher policy on the same training tasks. **Stage 2** applies an agentic verification framework to generate rationales that explain whether each action is valid or problematic. **Stage 3** uses these annotated trajectories to train a critique agent, then redeploys the critique agent to collect verification-enriched experiences for training a critique-aware tool-calling agent.

duce substantial inference latency, token overhead, and limited scalability (Renze and Guven, 2024). Although reinforcement learning methods such as GRPO (Shao et al., 2024) reduce inference-time overhead compared with these frameworks and can improve overall task success, single shot performance gains do not necessarily constitute reliability across multiple trials. Natural-language actor-critic methods provide a promising direction by using textual critiques as intermediate feedback, but they typically optimize the actor and critic jointly without sufficient supervision for the critic to produce reliable verification rationales (Jiang et al., 2026; Hong et al., 2025). This limitation is particularly important because even frontier models can struggle to judge whether an actor's action is wrong in long-horizon interactions.

To overcome these limitations, we propose CAST, a critique-aware training framework for improving long-horizon tool-calling reliability. CAST utilizes a multi-agent system to convert sparse trajectory-level outcomes into structured action-level verification supervision. Trained on it, a critique model (CAST-Critic) learns to judge whether each proposed action is valid under the information available at that step. CAST-Critic is then reused to annotate new interactions with fine-grained verification feedback. The resulting critique-enriched successful trajectories are then used to optimize the policy agent (CAST-Policy) that can benefit from explicit critique during inference or operate independently as a fine-tuned policy agent.

We evaluate CAST across four dynamic tool-calling domains, including retail, airline (Yao et al., 2024), telecom, and telehealth (He et al., 2025). Using retail as the in-domain environment, fine-tuned Qwen3 (Yang et al., 2025) models improve over base instruct models by 22.6% on pass^1 and 12.94% on pass^4. Across out-of-domain settings, CAST achieves average gains of 3.7% on pass^1 and 5.1% on pass^4. In addition, CAST achieves performance comparable to resource-intensive agentic frameworks while substantially reducing token usage and inference latency. These results show that structured critique supervision can improve both the reliability and efficiency of LLM agents in dynamic long-horizon environments. Our main contributions can be summarized as follows:

1. We propose CAST, a scalable critique-aware training framework that converts trajectory-level outcomes into structured action-level verification signals for training tool-calling agents.

2. We introduce an agentic data-generation pipeline for producing high-quality step-level critique rationales, which are used to train a critique model and construct critique-enriched policy optimization data.

3. We conduct extensive evaluations across two models and four domains, showing that CAST improves agent reliability while reducing the cost of inference compared with more resource-intensive agentic baselines.

2 Related Work

Tool-Augmented LLM Agents. Tool-augmented LLMs extend language models into interactive agents that plan and act over multi-step environments (Yao et al., 2022; Schick et al., 2023). While single-turn tool use is increasingly reliable, performance degrades sharply in long-horizon interactions, where hallucinated arguments, domain-rule violations, and incorrect tool calls compound across turns into unrecoverable failures (Lu et al., 2025; Yu et al., 2026). Recent reliability-oriented benchmarks such as τ-Bench (Yao et al., 2024) expose this gap directly by measuring whether agents succeed *consistently* across repeated runs of the same task, which is non-trivial to certain tasks that requires reliability. Closing this gap requires mechanisms that verify intermediate actions *before* errors propagate, rather than scoring only the final trajectory outcome.

Language-Based Feedback and Self-Correction. While task rewards are often sparse and binary, recent work supervises LLM behavior with natural-language feedback (Lin et al., 2024). Frameworks such as Reflexion and Self-Refine let models iteratively revise their outputs from textual critiques generated after each attempt (Shinn et al., 2023; Madaan et al., 2023). Two limitations make these methods difficult to deploy in long-horizon tool use. Critiques are produced post-hoc, Reflexion requires a terminal reward and Self-Refine requires a finished output, and neither formalizes the information available to the agent at the moment an action is taken, so when transplanted to multi-turn tool use the critic implicitly conditions on later observations or ground-truth user intent that were not part of the LLM's context (Li et al., 2025). Recent methods also improve reliability through failure recovery or modular optimization. PALADIN uses failure-conditioned retrieval, while EvoTool evolves tool-use policies from trajectory-level feedback (Vuddanti et al., 2025; Yang et al., 2026). These methods still provide coarser supervision than direct action-level verification. This motivates a critic that judges each action using only the partial history available at that step.

Actor-Critic Paradigms with Textual Supervision. Recent work has attempted to formalize language-based feedback within actor-critic architectures. Natural Language Actor-Critic (Hong et al., 2025) trains a generative critic to produce textual feedback rather than value estimates, and the Asymmetric Actor-Critic framework of Jiang et al. (2026) pairs a fixed proprietary actor with a lightweight open-source critic for multi-turn reliability. More broadly, generative verifiers cast reward modeling as next-token prediction so each judgment contains a Chain-of-Thought rationale (Zhang et al., 2025), and process-supervised RL uses an LLM judge for turn-level credit assignment in long-horizon tool-integrated reasoning (Tan et al., 2025). These approaches either jointly optimize actor and critic under sparse rewards or rely on heavyweight RL pipelines, leaving the critic without supervision for why an intermediate action is unreliable.

3 Method

In long-horizon tasks, an agent may follow a largely valid trajectory, yet a single erroneous action can jeopardize task completion by triggering cascading and sometimes irreversible consequences. For such tasks, it is important to verify individual actions so that incorrect actions can be prevented early or their downstream effects can be remediated before they lead to failure. However, verification is non-trivial: the critique model must judge a proposed action using only the information available at the current step, without access to future observations. We thus propose CAST, a critique-aware training framework that learns action-level verification signals and uses them to improve tool-calling policies (Figure 2). We formalize the problem setting and then describe the framework in detail as follows.

3.1 Problem Definition

We consider a dynamic tool-calling environment in which an agent interacts with a user over multiple turns. At step t, the agent receives the current observation o_t, maintains an interaction history $h_t = \big((o_1, a_1), \ldots, (o_{t-1}, a_{t-1})\big)$., and selects an action

$$a_t \sim \pi_\phi(\cdot \mid h_t, o_t, \mathcal{K}),$$

where $\mathcal{K}$ denotes the fixed tool set and domain constraints provided in the system prompt. The action a_t may either respond to the user or invoke a tool. Executing a_t produces the next observation o_{t+1} and updates the latent environment state.

A trajectory is defined as

$$\tau = \{(o_t, a_t)\}_{t=1}^{T},$$

with a trajectory-level success label $Y(\tau) \in \{0, 1\}$. Standard policy optimization uses this outcome to learn a policy that maximizes

$$\max_{\phi} \mathbb{E}_{\tau \sim \pi_\phi}[Y(\tau)].$$

Because $Y(\tau)$ alone does not reveal which action caused a failure, we additionally learn an action-level signal that judges each a_t before errors propagate.

We define an action verifier g as a function that maps the current interaction context and candidate action to a structured verification output

$$g(h_t, o_t, a_t, \mathcal{K}) \rightarrow (e_t, y_t),$$

where $y_t \in \{0, 1\}$ indicates whether action a_t is valid under the information available at step t, and e_t is a structured rationale justifying the verification decision. If the verifier accepts the candidate action, the action is executed and the environment advances; otherwise, the verifier's decision and rationale are returned to the assistant for revision. Verifier feedback and rejected candidates are not included in the original agent-environment trajectory, but instead form an auxiliary critique trace used before execution. Verification is conditioned only on the current interaction context and does not use future observations $o_{t+1}, \ldots, o_T$.

CAST learns a parameterized critique model (CAST-Critic)

$$C_\theta(h_t, o_t, a_t, \mathcal{K}) = (\hat{e}_t, \hat{y}_t),$$

and uses its outputs to construct critique-aware training data for optimizing the policy π_ϕ.

3.2 Overview of CAST

CAST consists of three learning stages. First, we collect trajectories from an initial policy π_ϕ over a task set $\mathcal{D} = \{x_i\}_{i=1}^M$. For each task x_i, we sample N executions $\mathcal{T}_i = \{\tau_i^{(j)}\}_{j=1}^N$ and construct the experience buffer $\mathcal{B} = \{\tau_i^{(j)} : i \in [M], j \in [N]\}$. Because the environment is dynamic, repeated executions of the same task produce diverse action sequences and outcomes. The buffer $\mathcal{B}$ contains both successful and failed trajectories.

Second, CAST constructs an annotated critique dataset. For each action a_t in each trajectory $\tau \in \mathcal{B}$, an agentic verification procedure V using a teacher model generates

$$V(h_t, o_t, a_t, \mathcal{K}, \Omega) \rightarrow (e_t, y_t).$$

where Ω denotes privileged information (if available) observable only at annotation time, such as ground truth action trajectories. The student critique model C_θ and the deployed policy never observe Ω and condition only on the step-local context $c_t = (h_t, o_t, a_t, \mathcal{K})$. This asymmetry allows V to assign reliable labels that are often undecidable from c_t alone to better facilitate critique learning. The resulting dataset is

$$\mathcal{A}_{\text{critique}} = \{(h_t, o_t, a_t, \mathcal{K}, e_t, y_t)\}.$$

Third, we train CAST-Critic and use it to collect critique-enriched trajectories. Successful critique-enriched trajectories are then selected to optimize the policy model π_ϕ.

3.3 Agentic Verification

The verification procedure assigns a structured rationale e_t and binary label y_t to each action. Given context $c_t = (h_t, o_t, a_t, \mathcal{K})$, the verifier determines whether a_t is supported by the information available at step t.

We decompose action failures into three categories: hallucination $f_{\text{hall}}(c_t)$, domain violation $f_{\text{domain}}(c_t)$, and wrong tool usage $f_{\text{tool}}(c_t)$. Hallucination captures unsupported assumptions, fabricated information, or arguments inconsistent with the observed state. Domain violation captures actions that conflict with system rules, task requirements, or domain-specific constraints. Wrong tool usage captures cases where the agent selects a tool that is incorrect or inconsistent with the current context. The final verification label is computed as

$$y_t = \mathbf{1}[\neg f_{\text{hall}}(c_t) \wedge \neg f_{\text{domain}}(c_t) \wedge \neg f_{\text{tool}}(c_t)].$$

The rationale e_t explains the verification decision by grounding it in the detected failure type or confirming that no failure is found.

CAST implements V with multiple specialized LLM agents. A rule extractor agent retrieves relevant domain constraints and a tool extractor agent identifies candidate tools for the current step. Additional agents separately evaluate hallucination, domain violations, and tool usage. An orchestrator agent aggregates the outputs of agents and produces the final pair (e_t, y_t).

3.4 Critique Model Learning

The critique model, CAST-Critic, is trained to generate both the verification rationale e_t and the verification label y_t from the action context

$$C_\theta(h_t, o_t, a_t, \mathcal{K}) = (\hat{e}_t, \hat{y}_t).$$

Model	τ-Retail (in-domain)			τ-Airline (out-of-domain)			τ-Telecom (out-of-domain)			τ-Telehealth (out-of-domain)		
	Pass^1	Pass^3	Pass^4	Pass^1	Pass^3	Pass^4	Pass^1	Pass^3	Pass^4	Pass^1	Pass^3	Pass^4
Q3-32B-ReAct	33.0%	11.7%	8.2%	29.6%	17.2%	15.2%	36.1%	20.8%	16.7%	28.0%	9.0%	6.0%
Q3-32B-FC	35.0%	15.0%	13.1%	17.6%	12.0%	10.8%	32.0%	21.0%	18.0%	42.0%	37.0%	35.0%
Q3-32B-Thinking	34.1%	11.8%	8.2%	43.0%	20.5%	16.0%	36.7%	15.0%	12.2%	29.0%	10.5%	7.0%
Base-4B	7.2%	6.3%	6.1%	**32.5%**	**25.5%**	**24.0%**	25.0%	12.5%	11.1%	22.5%	12.5%	10.0%
RFT-4B	27.6%	14.3%	12.2%	13.5%	8.5%	8.0%	**31.2%**	**19.4%**	16.6%	30.0%	22.5%	20.0%
Policy-4B (**ours**)	26.1%	**17.4%**	**16.5%**	19.0%	12.5%	12.0%	26.4%	16.7%	**16.7%**	**33.8%**	**30.0%**	**30.0%**
Policy-4B+Critic-4B (**ours**)	**28.3%**	12.2%	9.6%	27.5%	13.0%	10.0%	29.2%	16.7%	**16.7%**	28.8%	21.3%	20.0%
Base-8B	14.1%	6.3%	5.2%	13.5%	8.0%	8.0%	31.9%	9.7%	5.6%	32.5%	21.3%	20.0%
RFT-8B	27.6%	13.9%	12.2%	14.0%	7.5%	6.0%	29.2%	12.5%	11.1%	**37.5%**	**31.3%**	**30.0%**
Policy-8B (**ours**)	**30.0%**	**16.7%**	**14.8%**	9.5%	6.5%	6.0%	30.6%	12.5%	11.1%	**37.5%**	30.0%	**30.0%**
Policy-8B+Critic-8B (**ours**)	29.8%	14.6%	13.0%	**20.5%**	**13.0%**	**12.0%**	**38.9%**	**27.8%**	**27.8%**	32.5%	27.5%	25.0%

Table 1: Comparison of CAST with RFT and base instruct models across four environments, including Retail, Airline, Telecom, and Telehealth. The best result within each model-size block is highlighted in yellow.

We optimize C_θ using a combined objective

$$\mathcal{L}_{\text{critique}}(\theta) = \lambda_{\text{exp}}\mathcal{L}_{\text{exp}}(\theta) + \lambda_{\text{ver}}\mathcal{L}_{\text{ver}}(\theta),$$

where $\mathcal{L}_{\text{exp}}$ is the rationale generation loss and $\mathcal{L}_{\text{ver}}$ is the verification classification loss. This objective separates critique learning from policy optimization and provides direct supervision for producing both structured rationales and calibrated verification decisions.

3.5 Critique-Aware Policy Optimization

The trained CAST-Critic is then used to generate critique-enriched trajectories. Successful trajectories are selected to optimize the policy model, CAST-Policy. At each step, CAST-Policy proposes an action

$$\bar{a}_t \sim \pi_\phi(\cdot \mid h_t, o_t, \mathcal{K}),$$

and CAST-Critic produces

$$(\hat{e}_t, \hat{y}_t) = C_\theta(h_t, o_t, \bar{a}_t, \mathcal{K}).$$

If the candidate action is accepted by CAST-Critic, we set $a_t = \bar{a}_t$; otherwise, the critique feedback is used to revise the action, and a_t denotes the final accepted action. The final action and critique output are then appended to the interaction context, forming a critique-enriched history

$$tildeh_{t+1} = h_t \oplus (a_t, \hat{e}_t, \hat{y}_t).$$

The interaction continues until task termination and is collected into a new trajectory buffer $\mathcal{B}_C$ containing observations, actions, final task rewards, and critique signals.

We select successful critique-enriched trajectories

$$\mathcal{B}_C^+ = \{\tau \in \mathcal{B}_C \mid Y(\tau) = 1\},$$

and optimize CAST-Policy on $\mathcal{B}_C^+$ using supervised fine-tuning to internalize verification signals during long-horizon tool use

$$\mathcal{L}_{\text{policy}}(\phi) = -\sum_{\tau \in \mathcal{B}_C^+} \sum_{t=1}^{T} \log \pi_\phi(a_t \mid \tilde{h}_t, o_t, \mathcal{K}).$$

At inference time, CAST supports two modes. The optimized CAST-Policy can be used with CAST-Critic for explicit critique-guided interaction, or it can operate alone as an efficient standalone tool-calling agent.

4 Experiments

We evaluate CAST to answer three questions. First, does critique-aware training improve tool-calling reliability compared with base models and standard optimization baselines. Second, does CAST generalize from the in-domain training environment to unseen domains. Third, does the learned critique agent provide a more effective test-time verification signal than prompted frontier models and existing agentic frameworks.

Domain	Benchmark	No. Tools	No. Tasks	No. Turns (approx.)
Airline	τ-bench	12	50	[30–50]
Retail	τ-bench	14	114	[30–50]
Telecom	τ-Trait	16	18	[30–50]
Telehealth	τ-Trait	21	20	[30–50]

Table 2: Overview of the evaluation benchmarks.

4.1 Experimental Setup

Models. We use Qwen3-4B and Qwen3-8B as the backbone models for both critique agents and critique-aware tool-calling agents. For each model

size, we fine-tune one model to produce structured verification rationales and another model to act as a critique-aware policy. We compare these models with their corresponding base instruct models and with larger tool-calling models, including Qwen3-32B, Qwen3-235B-A22B-Instruct-2507 (Yang et al., 2025), GPT-4.1 (OpenAI et al., 2024), and GPT-OSS-120B (OpenAI et al., 2025). Fine-tuning details and hyper-parameters are reported in Appendix A.

Dataset. We construct the CAST training set from the official τ-Bench Retail split, which contains 500 tasks. For each task, we sample 5 task trajectories with Qwen2.5-72B-Instruct as user model and Qwen3-32B as assistant model and apply CAST's agentic verification procedure to annotate the resulting trajectories. This yields 2,500 critique-enriched experiences with structured step-level feedback for critique learning and policy optimization. Data generation details are in Appendix B.

Benchmarks. We evaluate all methods on τ-Bench (Yao et al., 2024) and τ-Trait (He et al., 2025) across four dynamic tool-calling domains, including Retail, Airline, Telecom, and Telehealth as summarized in Table 2. All models are fine-tuned only on the Retail training split, which we use as the in-domain environment. Airline, Telecom, and Telehealth are reserved for out-of-domain evaluation to measure transfer beyond the training domain. We use pass^1 to measure task success in a single execution and pass^4 to measure reliability across repeated executions. Details of the pass^k estimator and benchmark protocol are provided in Appendix C.

Baselines. We compare CAST with three groups of baselines. The first group includes standard prompting and tool-calling methods, including ReAct and function calling. The second group includes test-time agentic frameworks such as EvoTool, PALADIN, IRMA (Mishra et al., 2025) and FAMA (Saeidi et al., 2026), which use helper agents to support the main tool-calling agent during decision making. The third group includes optimization-based baselines, primarily rejection fine-tuning. In RFT, trajectories are collected, successful executions are selected according to task reward, and the model is optimized with supervised fine-tuning on the filtered data. This baseline isolates the effect of CAST's critique supervision from simply training on successful trajectories.

Method	Avg Pass^1	Avg Pass^4
Qwen3-32B (Actor)		
+ ReAct - no CA	31.7%	11.5%
+ CA (GPT-4.1)	22.8%	6.0%
+ CA (Q3-235B)	24.1%	3.4%
+ CA (Q2.5-72B)	17.9%	6.3%
+ CA (Critic-4B)	**31.8%**	15.9%
+ CA (Critic-8B)	30.4%	**19.5%**

Table 3: Comparison of the CAST-Critic models with frontier models when used as critique agent (CA).

4.2 Main Results

Key takeaway. CAST-Policy consistently improves in-domain reliability across model sizes, with the strongest gains on pass^4.

As shown in Table 1, on the Retail in-domain environment, CAST-Policy-4B improves over the base instruct model by 18.9% on pass^1 and 10.4% on pass^4. Although CAST-Policy-4B achieves comparable pass^1 performance to RFT-4B, it improves pass^4 by 4.3%, showing that CAST-Policy improves repeated-run reliability beyond simply training on successful trajectories. CAST-Policy-8B shows the same trend, outperforming RFT-8B and Base-8B by 2.4% and 15.9% on pass^1, and by 2.6% and 9.6% on pass^4, respectively.

Key takeaway. Smaller CAST-Policy models surpass Qwen3-32B on pass^4, showing stronger repeated-run reliability.

Table 1 also shows that CAST-Policy improves reliability-oriented performance relative to larger models. Although CAST-Policy-4B and CAST-Policy-8B have lower pass^1 performance than Qwen3-32B, they outperform the best Qwen3-32B setup by 3.4% and 1.7% on pass^4, respectively. This result shows that CAST-Policy can produce smaller agents that are more reliable across repeated executions than the larger model used during experience collection. This finding is further supported by Figure 1. Appendix F provides qualitative examples that show how critique-aware agents can prevent premature irreversible actions from being executed when compared to a Qwen3-8B baseline model.

Key takeaway. Agentic CAST generalizes better to out-of-domain environments.

In out-of-domain environments, agentic CAST shows stronger generalization than the standalone

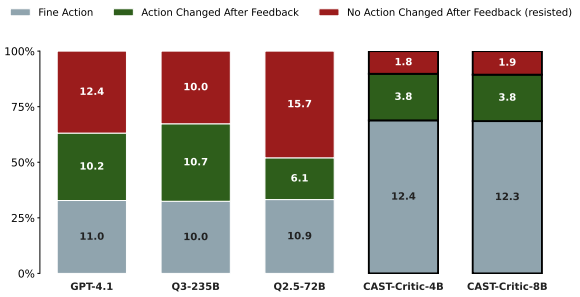

Figure 3: Action distribution of the tool-calling agent under different critique agents.

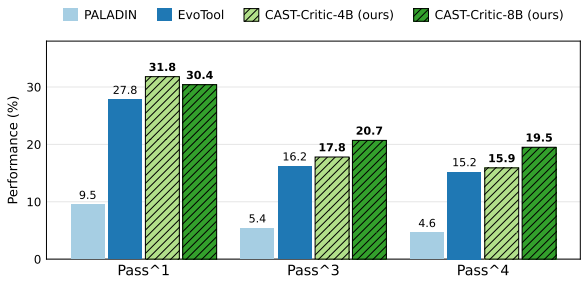

Figure 5: Comparison of CAST against PALADIN and EvoTool.

critique-aware policy. As shown in Table 1, on Airline and Telecom, CAST-Policy-8B paired with CAST-Critic-8B outperforms both the base instruct model and RFT-8B on pass^1 and pass^4. We also observe consistent gains in the remaining out-of-domain settings compared with competing methods. These results indicate that CAST's critique-guided interaction transfers beyond the training domain and improves robustness under domain shift.

Key takeaway. CAST-Critic provides more calibrated verification than frontier models.

To evaluate CAST-Critic, we fix the tool-calling agent as Qwen3-32B and compare different critique models under the same prompting setup. Table 3 shows that directly using frontier models as critique agents introduces substantial noise and reduces the performance of the base tool-calling agent. Our analysis shows that prompted instruct and frontier models tend to be overly pessimistic, frequently labeling valid actions as problematic. This behavior lengthens conversations and pushes the agent into unnecessary revision loops. In contrast, our 4B and 8B critique agents produce more calibrated judgments and trigger revisions mainly when the tool-calling action is unsuitable. The action-trigger ratios in Figure 3 further support this finding.

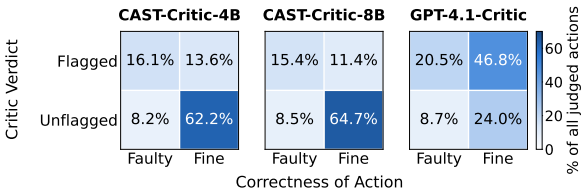

Figure 4: Confusion matrices of CAST-Critic-4B, CAST-Critic-8B, and GPT-4.1-Critic, comparing critic verdicts with action correctness.

Consistent with this trend, the confusion matrices in Figure 4 provide a more detailed view of the critics' behavior. GPT-4.1-Critic flags 46.8% of all judged actions even when they are correct, indicating a high false-positive rate and explaining the unnecessary revisions observed in Figure 3. In contrast, CAST-Critic-4B and CAST-Critic-8B reduce this rate to 13.6% and 11.4%, respectively, while maintaining comparable detection of faulty actions. At the same time, CAST-Critic-4B and CAST-Critic-8B correctly leave 62.2% and 64.7% of actions unflagged when they are valid, compared with only 24.0% for GPT-4.1-Critic. These results further show that CAST-Critic better distinguishes genuinely faulty actions from valid ones, providing more calibrated verification and avoiding unnecessary intervention.

Key takeaway. CAST achieves strong reliability against test-time agentic frameworks with smaller critique agents and moderate overhead.

We compare CAST with test-time reliability-oriented frameworks such as IRMA and FAMA, using Qwen3-32B as the tool-calling agent across all methods. As shown in Figure 6, CAST outperforms IRMA on Airline by 3.0% on pass^1 and on Retail by 4.5% on pass^4, while using 4B or 8B critique agents instead of the Qwen2.5-72B helper agents used by IRMA and FAMA. CAST introduces more overhead than single-pass inference because it performs critique-guided refinement, but its token usage and latency remain within the range of existing agentic frameworks. These results show that CAST can achieve competitive reliability with substantially smaller helper models and practical inference cost.

We further compare CAST with PALADIN and EvoTool, two methods designed to improve the reliability of tool-using agents. PALADIN improves failure recovery by training on recovery-annotated trajectories with injected tool failures and retrieving

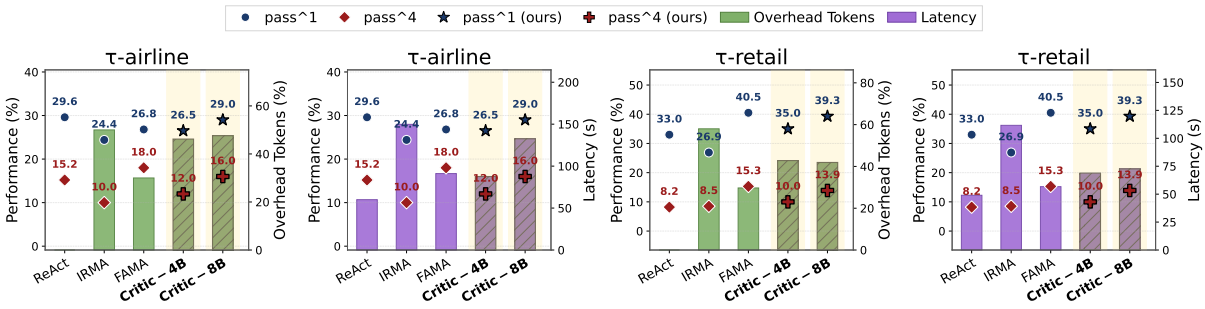

Figure 6: Comparison of CAST with agentic frameworks in terms of performance, token overhead, and latency.

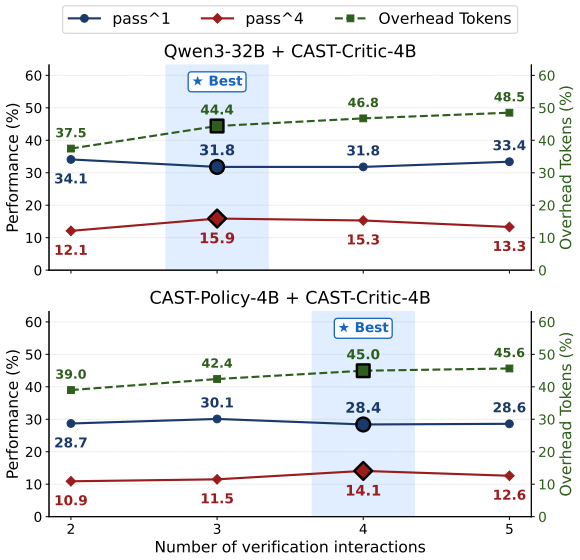

Figure 7: Comparison of performance and token overhead across different numbers of verification steps.

similar failure cases at inference time, whereas EvoTool iteratively evolves a modular tool-use policy through blame-aware mutation and diversity-aware selection. As shown in Figure 5, CAST substantially outperforms PALADIN across all evaluation settings, with CAST-Critic-8B improving pass^1, pass^3, and pass^4 by 20.9, 15.3, and 14.9 percentage points, respectively. Compared with EvoTool, CAST-Critic-8B achieves improvements of 2.6, 4.5, and 4.3 percentage points on pass^1, pass^3, and pass^4, respectively. These results demonstrate that a simple actor–critic formulation with CAST can outperform both failure-recovery and evolutionary tool-use methods.

4.3 Ablation and Analysis

Key takeaway. The best verification budget depends on the tool-calling agent.

To study the effect of verification budget, we vary the number of refinement rounds from 2 to 5 after the critique agent flags an unsuitable action. The ablation uses two tool-calling agents, Qwen3-32B and CAST-Policy-4B, with CAST-Critic-4B fixed as the critique model. As shown in Figure 7, additional verification increases token overhead for both agents, but reliability does not improve monotonically. Qwen3-32B achieves its best pass^4 with 3 rounds, whereas CAST-Policy-4B reaches its best pass^4 with 4 rounds. This indicates that the optimal verification budget depends on how the tool-calling agent responds to critique feedback. In almost all settings, the CAST-Policy-4B uses fewer tokens than the Qwen3-32B, suggesting that the critique-aware policy incorporates verification feedback with lower interaction cost.

Key takeaway. CAST-Critic identifies problematic actions without over-triggering revisions.

We analyze test-time verification behavior under two settings. In the first setting, Qwen3-32B is used as the tool-calling agent and paired with our 4B or 8B critique agent. In the second setting, CAST-Policy-4B and CAST-Policy-8B are paired with CAST-Critic of the same scale. The left plot in Figure 8 shows the distribution of verification outcomes across these settings. Our critique agents classify most tool-calling actions as suitable and flag only 33.2% of agent actions as unsuitable per task. This supports the finding that the learned critique agents are not overly pessimistic and do not frequently push the tool-calling agent into unnecessary revision loops.

Key takeaway. CAST-Policy models are less resistant to verification feedback than non-fine-tuned tool-calling agents.

We further examine how the tool-calling agent responds after receiving critique feedback. We

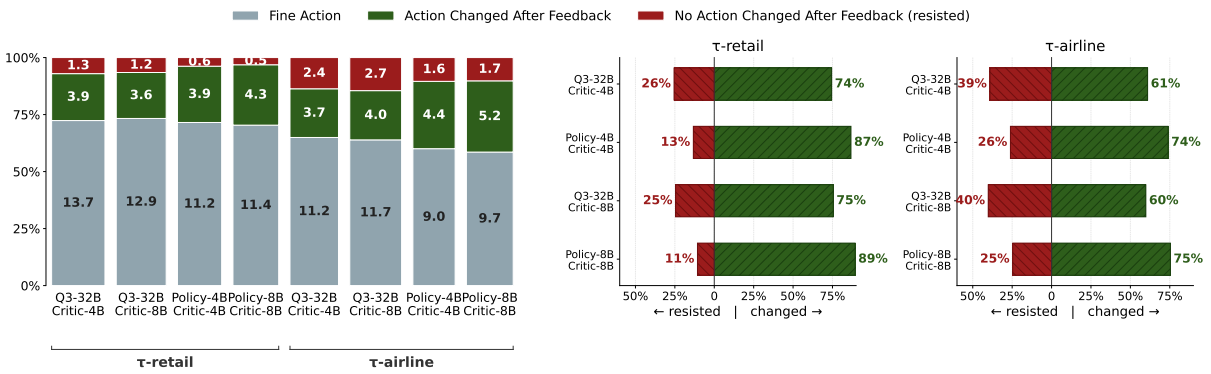

Figure 8: **Left:** Distribution of tool-calling agent actions across different agentic scenarios. **Right:** Comparison of how often instruct and CATC agents resist the verification signal after receiving critique feedback.

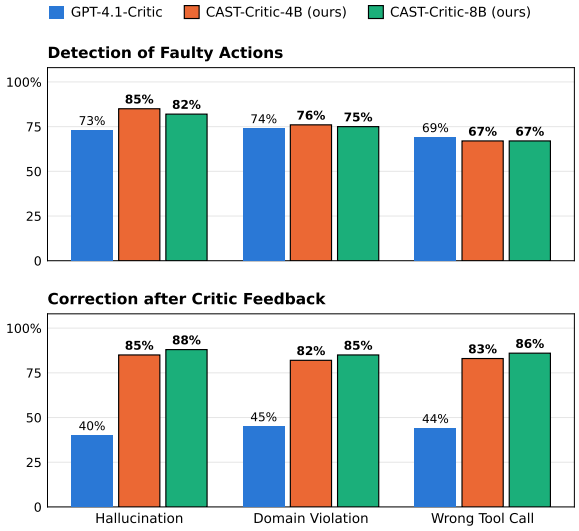

Figure 9: **Left:** critic accuracy in detecting faulty actions across hallucination, domain violation, and wrong tool-call. **Right:** correction rate after critic feedback for the same error types. The actor model is Qwen3-32B.

distinguish between cases where the agent accepts the critique and revises its action and cases where it resists the critique and keeps the original action. As shown in the right plot of Figure 8, CAST-Policy agents exhibit roughly 2 times less resistance to verification feedback than non-fine-tuned agents. This explains why smaller CAST-Policy models can achieve performance comparable to larger non-fine-tuned tool-calling agents when paired with our critique agents. Together with the pass^4 results in Tables 1 and 3, this indicates that CAST improves not only the quality of the critique signal but also the policy's ability to act on that signal.

Key takeaway. CAST-Critics provide more effective verification and corrective feedback.

Table 3 shows that CAST Critics outperform strong open- and closed-source critic models. To better understand these gains, we conduct a fine-grained analysis of their ability to detect different failure types and the effectiveness of their feedback in correcting faulty actions, using GPT-4.1 as the baseline critic. As shown in Figure 9 (left), CAST Critics substantially outperform GPT-4.1 in detecting hallucinations, achieve comparable or slightly better performance on domain violations, and remain competitive on wrong tool calls. More importantly, Figure 9 (right) shows that feedback from CAST Critics leads to successful correction in 82–88% of cases across all three error categories, substantially outperforming GPT-4.1. These results highlight that CAST Critics not only identify erroneous actions effectively but also provide actionable feedback that helps the actor correct them.

5 Conclusion

We presented CAST, a critique-aware framework for making tool-calling agents more reliable in dynamic long-horizon interactions. CAST starts from the observation that many failures arise from intermediate actions that appear plausible but later derail the task. To address this, CAST generates structured verification rationales for agent actions, trains a critique agent to provide calibrated feedback, and uses this feedback to optimize a critique-aware tool-calling policy. Experiments across in-domain and out-of-domain settings show that this training process improves both task success and repeated-run reliability. These results suggest that learning to critique intermediate actions is a practical path toward agents that solve complex tasks more consistently.

Limitations

Although CAST improves the reliability of tool-calling agents, two limitations remain. First, CAST trains the critique agent and policy agent with supervised fine-tuning, which provides stable optimization but does not directly train the policy through on-policy interaction with critique feedback. Future work could use the learned critique agent as a feedback model for reinforcement learning, allowing the policy to learn more adaptive correction strategies from its own exploration. Second, CAST focuses on verifying the current policy action rather than explicitly modeling its downstream consequences. Since CAST generates critique data during training, future extensions could incorporate forward-looking rationales that describe how a local action affects later states or task outcomes, providing stronger supervision for long-horizon decision making.

Ethics Statement

We have utilized AI assistants, specifically Grammarly and ChatGPT, to correct grammatical errors and rephrase sentences.

Acknowledgement

We thank the anonymous reviewers for their constructive suggestions. We extend our gratitude to Research Computing (RC) and Enterprise Technology at ASU for providing computing resources and access to OpenAI models for experiments. This work was in part supported by a gift award from Cisco Research. Baral was also partially supported by a DARPA contract.

References

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

Daya Guo, Dejian Yang, Haowei Zhang, Junxiao Song, Peiyi Wang, Qihao Zhu, Runxin Xu, Ruoyu Zhang, Shirong Ma, Xiao Bi, Xiaokang Zhang, Xingkai Yu, Yu Wu, Z. F. Wu, Zhibin Gou, Zhihong Shao, Zhuoshu Li, Ziyi Gao, Aixin Liu, and 175 others. 2025. Deepseek-r1 incentivizes reasoning in llms through reinforcement learning. *Nature*, 645(8081):633–638.

Muyu He, Anand Kumar, Tsach Mackey, Meghana Rajeev, James Zou, and Nazneen Rajani. 2025. Impatient users confuse ai agents: High-fidelity simulations of human traits for testing agents. *arXiv preprint arXiv:2510.04491*.

Joey Hong, Kang Liu, Zhan Ling, Jiecao Chen, and Sergey Levine. 2025. Natural language actor-critic: Scalable off-policy learning in language space. *Preprint*, arXiv:2512.04601.

Shuli Jiang, Zhaoyang Zhang, Yi Zhang, Shuo Yang, Wei Xia, and Stefano Soatto. 2026. Asymmetric actor-critic for multi-turn llm agents. *arXiv preprint arXiv:2604.00304*.

Philippe Laban, Hiroaki Hayashi, Yingbo Zhou, and Jennifer Neville. 2025. Llms get lost in multi-turn conversation. *arXiv preprint arXiv:2505.06120*.

Yansi Li, Jiahao Xu, Tian Liang, Xingyu Chen, Zhiwei He, Qiuzhi Liu, Rui Wang, Zhuosheng Zhang, Zhaopeng Tu, Haitao Mi, and Dong Yu. 2025. Dancing with critiques: Enhancing llm reasoning with stepwise natural language self-critique. *arXiv preprint arXiv:2503.17363*.

Zicheng Lin, Zhibin Gou, Tian Liang, Ruilin Luo, Haowei Liu, and Yujiu Yang. 2024. Criticbench: Benchmarking llms for critique-correct reasoning. In *Findings of the Association for Computational Linguistics: ACL 2024*, pages 1552–1587.

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Felix Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, Zirui Wang, and Ruoming Pang. 2025. Toolsandbox: A stateful, conversational, interactive evaluation benchmark for llm tool use capabilities. *Preprint*, arXiv:2408.04682.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Sean Welleck, Bodhisattwa Prasad Majumder, Shashank Gupta, Amir Yazdanbakhsh, and Peter Clark. 2023. Self-refine: Iterative refinement with self-feedback. *Preprint*, arXiv:2303.17651.

Venkatesh Mishra, Amir Saeidi, Satyam Raj, Mutsumi Nakamura, Jayanth Srinivasa, Gaowen Liu, Ali Payani, and Chitta Baral. 2025. How can input reformulation improve tool usage accuracy in a complex dynamic environment? a study on tau-bench. *arXiv preprint arXiv:2508.20931*.

OpenAI, Josh Achiam, Steven Adler, Sandhini Agarwal, Lama Ahmad, Ilge Akkaya, Florencia Leoni Aleman, Diogo Almeida, Janko Altenschmidt, Sam Altman, Shyamal Anadkat, Red Avila, Igor Babuschkin, Suchir Balaji, Valerie Balcom, Paul Baltescu, Haiming Bao, Mohammad Bavarian, Jeff Belgum, and 262 others. 2024. Gpt-4 technical report. *Preprint*, arXiv:2303.08774.

OpenAI, Sandhini Agarwal, Lama Ahmad, Jason Ai, Sam Altman, Andy Applebaum, Edwin Arbus, Rahul K. Arora, Yu Bai, Bowen Baker, Haiming Bao, Boaz Barak, Ally Bennett, Tyler Bertao, Nivedita Brett, Eugene Brevdo, Greg Brockman, Sebastien Bubeck, Che Chang, and 107 others. 2025. gpt-oss-120b & gpt-oss-20b model card. *Preprint*, arXiv:2508.10925.

Cheng Qian, Zuxin Liu, Akshara Prabhakar, Jielin Qiu, Zhiwei Liu, Haolin Chen, Shirley Kokane, Heng Ji, Weiran Yao, Shelby Heinecke, Silvio Savarese, Caiming Xiong, and Huan Wang. 2025. Userrl: Training interactive user-centric agent via reinforcement learning. *Preprint*, arXiv:2509.19736.

Yujia Qin, Shihao Liang, Yining Ye, Kunlun Zhu, Lan Yan, Yaxi Lu, Yankai Lin, Xin Cong, Xiangru Tang, Bill Qian, Sihan Zhao, Lauren Hong, Runchu Tian, Ruobing Xie, Jie Zhou, Mark Gerstein, dahai li, Zhiyuan Liu, and Maosong Sun. 2024. ToolLLM: Facilitating large language models to master 16000+ real-world APIs. In *The Twelfth International Conference on Learning Representations*.

Matthew Renze and Erhan Guven. 2024. Self-reflection in llm agents: Effects on problem-solving performance. *arXiv preprint arXiv:2405.06682*.

Amir Saeidi, Venkatesh Mishra, Souradeep Mukhopadhyay, Gaowen Liu, Ali Payani, Jayanth Srinivasa, and Chitta Baral. 2026. Fama: Failure-aware meta-agentic framework for open-source llms in interactive tool use environments. *arXiv preprint arXiv:2604.25135*.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. 2023. Toolformer: Language models can teach themselves to use tools. *Advances in neural information processing systems*, 36:68539–68551.

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

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. 2023. Reflexion: Language agents with verbal reinforcement learning. *Advances in neural information processing systems*, 36:8634–8652.

Weiting Tan, Xinghua Qu, Ming Tu, Meng Ge, Andy T Liu, Philipp Koehn, and Lu Lu. 2025. Process-supervised reinforcement learning for interactive multimodal tool-use agents. *arXiv preprint arXiv:2509.14480*.

Sri Vatsa Vuddanti, Aarav Shah, Satwik Kumar Chittiprolu, Tony Song, Sunishchal Dev, Kevin Zhu, and Maheep Chaudhary. 2025. Paladin: self-correcting language model agents to cure tool-failure cases. *arXiv preprint arXiv:2509.25238*.

An Yang, Anfeng Li, Baosong Yang, Beichen Zhang, Binyuan Hui, Bo Zheng, Bowen Yu, Chang Gao, Chengen Huang, Chenxu Lv, Chujie Zheng, Dayiheng Liu, Fan Zhou, Fei Huang, Feng Hu, Hao Ge, Haoran Wei, Huan Lin, Jialong Tang, and 41 others. 2025. Qwen3 technical report. *Preprint*, arXiv:2505.09388.

Shuo Yang, Caren Han, Xueqi Ma, Yan Li, Mohammad Reza Ghasemi Madani, and Eduard Hovy. 2026. Evotool: Self-evolving tool-use policy optimization in llm agents via blame-aware mutation and diversity-aware selection. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 43553–43572.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. 2024. tau-bench: A benchmark for tool-agent-user interaction in real-world domains. *arXiv preprint arXiv:2406.12045*.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. 2022. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*.

Peijie Yu, Wei Liu, Yifan Yang, Jinjian Li, Zelong Zhang, Xiao Feng, and Feng Zhang. 2026. Benchmarking LLM tool-use in the wild. In *The Fourteenth International Conference on Learning Representations*.

Murong Yue. 2025. A survey of large language model agents for question answering. *arXiv preprint arXiv:2503.19213*.

Lunjun Zhang, Arian Hosseini, Hritik Bansal, Seyed Mehran Kazemi, Aviral Kumar, and Rishabh Agarwal. 2025. Generative verifiers: Reward modeling as next-token prediction. In *International Conference on Learning Representations*, volume 2025, pages 12476–12505.

Jeffrey Zhou, Tianjian Lu, Swaroop Mishra, Siddhartha Brahma, Sujoy Basu, Yi Luan, Denny Zhou, and Le Hou. 2023. Instruction-following evaluation for large language models. *arXiv preprint arXiv:2311.07911*.

A Details about Fine Tuning

We fine-tune Qwen-series models using the multi-turn agentic data we generated on one nodes of a supercomputing cluster equipped with NVIDIA H100 GPUs. The maximum sequence length is set to 32,768 for Qwen3 models. Detailed hyperparameters are provided in Table 4.

Hyper-parameter	Value
Learning Rate	5×10^{-6}
Number of Epochs	3
Number of Devices	4
Total Batch Size	16
Optimizer	AdamW
Learning Rate Scheduler	cosine

Table 4: Fine-tuning hyperparameters used for Qwen3 models.

B Details about Dataset

We fine-tune two models in CAST, a critique model and a policy model. To construct the initial trajectory data, we use Qwen3-32B in ReAct mode as the teacher policy. We use the official τ-Bench training split, which contains 500 tasks, and run the teacher model five times per task with temperature 0.7 to collect diverse solution trajectories. In these rollouts, Qwen2.5-72B is used as the user simulator. After collecting the trajectories, we annotate each policy action using our agentic verification framework. We use Qwen3.5-27B and Qwen3.5-122B-A10B as verification agents and find that Qwen3.5-122B-A10B produces higher-quality annotations. The annotated trajectories are then reconstructed into multi-turn training examples for the critique model, with the tool list and domain policy provided in the system prompt to improve generalization.

After fine-tuning the critique model, we perform a second round of training data collection. We again run the 500 τ-Bench training tasks for five trials, but now the teacher policy actions are verified by the learned critique model during interaction. We retain only trajectories with reward 1 and use these successful critique-enriched trajectories to train the critique-aware tool-calling policy.

C Details about Benchmarks

τ-bench τ-bench evaluates tool-using conversational agents in realistic customer-service environments. The benchmark includes two domains, Retail and Airline, where an agent interacts with a simulated user, follows domain-specific policy rules, and uses API tools to query or update a backend state. The Retail domain contains 500 users, 50 products, and 1,000 orders, with 15 tools in total, including 7 write tools and 8 non-write tools. It includes 115 tasks covering cancellations, returns, exchanges, address changes, and information requests. The Airline domain contains 500 users, 300 flights, and 2,000 reservations, with 13 tools in total, including 6 write tools and 7 non-write tools. It includes 50 tasks covering reservation booking, modification, cancellation, and refund-related workflows.

τ-trait τ-trait extends τ-bench to evaluate robustness under more diverse user behaviors and domain shifts. It preserves the multi-turn tool-use setting while introducing persona-aware user simulation and expanding the benchmark beyond Retail and Airline to include Telecom and Telehealth. The benchmark is designed to test robustness, personalization, and fairness under realistic behavior variation. It contains 218 tasks across four domains, including Retail with 120 tasks, Airline with 60 tasks, Telecom with 18 tasks, and Telehealth with 20 tasks.

For both τ-bench and τ-trait, we evaluate task success using the pass^k metric (Yao et al., 2024). This metric measures the probability that k independently sampled executions of a task are all successful, averaged across tasks. Given n independent trials for a task and c successful executions with reward $r = 1$, pass^k is computed as

$$\text{pass}^\wedge k = \mathbb{E}_{\text{task}} \left[\frac{\binom{c}{k}}{\binom{n}{k}} \right].$$

D Details Results

We report the detailed results of all experiments conducted in this study in Tables 4–10.

E CAST Agentic Prompts

E.1 Data Annotation Prompts

Tool Extractor Agent - System Prompt

```
You are the Tool Extractor Agent.

Inputs you will be given:
- AVAILABLE_TOOLS: a plain-text list of
tool names with descriptions/parameters.
```

Model	τ-Bench Retail				τ-Bench Airline			
	Pass^1	Pass^2	Pass^3	Pass^4	Pass^1	Pass^2	Pass^3	Pass^4
GPT-OSS-120B	31.3%	16.2%	9.7%	5.9%	37.2%	28.4%	24.6%	22.0%
Q3-235B	49.4%	36.8%	30.9%	27.8%	34.0%	25.3%	23.0%	22.0%
Q3-32B-Thinking	34.1%	18.4%	11.8%	8.2%	43.0%	27.7%	20.5%	16.0%
Q3-32B-ReAct	33.0%	18.2%	11.7%	8.2%	29.6%	21.0%	17.2%	15.2%
Q3-32B-FC	35.0%	26.6%	15.0%	13.1%	17.6%	13.6%	12.0%	10.8%

Table 5: Results on τ-Bench.

Model	τ-Trait Telecom				τ-Trait Telehealth			
	Pass^1	Pass^2	Pass^3	Pass^4	Pass^1	Pass^2	Pass^3	Pass^4
GPT-OSS-120B	42.2%	30.6%	23.3%	18.9%	32.0%	26.0%	23.0%	21.0%
Q3-235B	52.8%	42.6%	37.5%	33.3%	42.5%	35.0%	30.0%	25.0%
Q3-32B-Thinking	36.7%	20.6%	15.0%	12.2%	29.0%	17.0%	10.5%	7.0%
Q3-32B-ReAct	36.1%	25.9%	20.8%	16.7%	28.0%	15.0%	9.0%	6.0%
Q3-32B-FC	32.0%	25.0%	21.0%	18.0%	42.0%	39.0%	37.0%	35.0%

Table 6: Results on τ-Trait.

```
- DOMAIN_RULES: domain constraints and
policies that apply to tool usage.
- OBSERVATION(t): the observation at turn
 t (either the user's query or a tool's
output).
- TURN: the turn number t.

Your task:
Return a SHORTLIST of candidate tools
most relevant to addressing OBSERVATION(t
).
- Match tool descriptions to the needed
next step based on the observation.
- Consider domain constraints and
policies when selecting tools.
- Include multiple candidates if
applicable.
- Do not invent tools.
- Only suggest tools that are appropriate
 given the domain rules.

Output format {"(plain text, no JSON)" if
 STRUCTURED_OUTPUT else ""}:
TURN: <t>
CANDIDATE_TOOLS:
- <tool_name_1>
- <tool_name_2>
(If none, write "CANDIDATE_TOOLS: (none)
")
```

Rule Extractor Agent - System Prompt

```
You are the Rule Extractor Agent.

Inputs you will be given:
- DOMAIN_RULES: a plain-text block of
domain rules/policies.
- OBSERVATION(t): the observation at turn
 t (either the user's query or a tool's
output).
- TURN: the turn number t.

Your task:
Return ONLY the rules from DOMAIN_RULES
that are directly relevant to responding
to OBSERVATION(t).
- Extract the minimal, essential
constraints and prerequisites needed for
the next action.
- You may quote short phrases or
paraphrase for brevity, but do not change
 the meaning.
- Exclude unrelated content and avoid
duplications.

Extraction guidance (generic, no
hardcoded cases):
- Identify prerequisites that must be
satisfied before an action (e.g.,
required checks, confirmations, status
validations) only if they are implicated
by OBSERVATION(t) and the domain rules.
- Identify constraints that limit or
shape the allowed actions (what must/
must not be done) relevant to OBSERVATION
(t).
```

Model	τ-Bench Retail				τ-Bench Airline			
	Pass^1	Pass^2	Pass^3	Pass^4	Pass^1	Pass^2	Pass^3	Pass^4
Q3-32B-FAMA	40.0%	26.9%	19.9%	15.3%	26.8%	20.0%	18.4%	18.0%
Q3-32B-IRMA	26.9%	15.2%	10.9%	8.5%	24.4%	15.9%	12.4%	10.0%

Table 7: Results on τ-Bench.

Model	τ-Trait Telecom				τ-Trait Telehealth			
	Pass^1	Pass^2	Pass^3	Pass^4	Pass^1	Pass^2	Pass^3	Pass^4
Q3-32B-FAMA	37.8%	31.7%	30.0%	28.9%	30.0%	24.5%	20.0%	18.0%
Q3-32B-IRMA	40.0%	28.0%	24.4%	23.3%	45.0%	37.0%	35.0%	35.0%

Table 8: Results on τ-Trait.

```
- Do not inject examples or policies not
supported by DOMAIN_RULES; infer strictly
 from the provided rules.

Selection guidance:
- Prefer rules that state prerequisites
and constraints over general descriptions.

- When multiple similar rules exist,
include the most specific ones; avoid
duplicating near-identical text.

Output format (plain text, no JSON):
TURN: <t>
RELEVANT_RULES:
- <concise rule 1>
- <concise rule 2>
(If none, write "RELEVANT_RULES: (none)")
```

Domain Violation Checker - System Prompt

```
You are the Domain Violation Checker.

Context you always know:
- USER PLAN: This describes the overall
instruction and the required actions (the
 tools that must eventually be used in
the conversation).
{USER_PLAN}
IMPORTANT: USER_PLAN describes the
overall task goal for reference. However,
 for per-turn evaluation, you must ONLY
consider information that has actually
appeared in the conversation (
PREVIOUS_OBSERVATIONS + OBSERVATION) up
to turn t. Do NOT treat details from
USER_PLAN (such as specific order IDs,
item IDs, or action parameters) as if the
 user has already communicated them - the
 user may reveal information gradually
across turns.

- DOMAIN RULES / POLICIES (full set for
the domain):
{DOMAIN_RULES}

- AVAILABLE TOOLS (full list with
descriptions/parameters):
{TOOLS_TEXT}

Inputs you will be given for assistant
action at turn t:
- PREVIOUS_OBSERVATIONS(1:t-1): selected
conversation history up to but not
including turn t (prior user queries,
tool outputs, and assistant actions).
- OBSERVATION(t): the observation at t (
user query or tool output).
- ASSISTANT_ACTION(t): the assistant's
action content at t (tool name + args, or
 text response).
- RELEVANT_RULES(t): the set of rules
that apply to this turn (helper
extraction; may be incomplete).
- CANDIDATE_TOOLS(t): the shortlist of
tools appropriate for this turn (helper
extraction; may be incomplete).

Your task:
Determine whether ASSISTANT_ACTION(t)
violates any domain policy. Focus
EXCLUSIVELY on policy violations.

What counts as a domain policy violation:
- Skipping prerequisites required by
domain rules (e.g., authentication,
status checks, user confirmation before
consequential actions).
- Contradicting domain constraints (e.g.,
 wrong order status for the requested
operation, cross-product-type changes
that are forbidden).
- Making up information or providing
facts not grounded in the conversation or
```

Model	τ-Bench Retail				τ-Bench Airline			
	Pass^1	Pass^2	Pass^3	Pass^4	Pass^1	Pass^2	Pass^3	Pass^4
Q3-32B+GPT-4.1	17.0%	7.5%	4.3%	2.6%	29.0%	21.3%	18.0%	16.0%
Q3-32B+Q3-235B	20.9%	7.1%	3.3%	1.7%	30.5%	21.3%	16.0%	12.0%
Q3-32B+Q2.5-72B	19.1%	10.6%	7.6%	6.1%	29.5%	22.0%	17.5%	14.0%
Q3-32B+CAST-Critic-4B	35.0%	20.9%	13.9%	10.0%	26.5%	16.0%	13.0%	12.0%
Q3-32B+CAST-Critic-8B	29.8%	18.4%	14.6%	13.0%	20.5%	15.0%	13.0%	12.0%

Table 9: Results on τ-Bench.

Model	τ-Trait Telecom				τ-Trait Telehealth			
	Pass^1	Pass^2	Pass^3	Pass^4	Pass^1	Pass^2	Pass^3	Pass^4
Q3-32B+GPT-4.1	29.2%	14.8%	8.3%	5.5%	16.3%	6.6%	2.5%	0.0%
Q3-32B+Q3-235B	26.4%	11.1%	4.0%	0.0%	18.8%	8.3%	2.5%	0.0%
Q3-32B+Q2.5-72B	8.0%	2.8%	1.4%	0.0%	15.0%	7.5%	5.0%	5.0%
Q3-32B+CAST-Critic-4B	33.3%	23.1%	19.4%	16.7%	32.5%	27.5%	25.0%	25.0%
Q3-32B+CAST-Critic-8B	38.9%	29.6%	27.8%	27.8%	32.5%	30.0%	27.5%	25.0%

Table 10: Results on τ-Trait.

```
 domain rules.
- Performing an action that domain policy
 explicitly forbids in the current
context.

What is NOT a domain policy violation (do
 NOT flag these):
- Using a suboptimal or wrong tool for
non-policy reasons (that is a tool
selection issue, not a policy violation).
- A text response that merely reports the
 result of a PREVIOUS action. If the
policy violation occurred in a prior turn
 (e.g., the assistant executed a tool
without user confirmation), do NOT re-
flag the follow-up response as a new
violation. The violation was in the prior
 action, not in reporting its outcome.
Only flag this turn if the response
itself introduces a NEW violation (e.g.,
fabricating information in the response
text).

Grounding policy:
- Treat USER messages and TOOL outputs as
 ground truth. Assistant statements are
unverified unless supported by USER or
TOOL evidence.
- Treat helper extracts (RELEVANT_RULES/
CANDIDATE_TOOLS) as hints; they may be
incomplete. Base judgment on
PREVIOUS_OBSERVATIONS(1:t-1) +
OBSERVATION(t) + DOMAIN_RULES/
AVAILABLE_TOOLS.

Requirements:
1) Begin the reasoning with: "Let's
verify step by step."
2) Reason about whether the action
violates any stated domain policy, citing
 the specific policy and how it is
violated (or confirming compliance).
3) Be specific and evidence-based, but
keep all references implicit (explain
what and why, not where you read it).
4) Do not explicitly name section headers
 like "USER_PLAN", "CANDIDATE_TOOLS", "
RELEVANT_RULES" in your output.
5) In "violated_policies", list each
violated policy as an exact, complete
quote from the DOMAIN RULES / POLICIES
provided above. Do not paraphrase or
summarize - copy the full policy text
verbatim. If no policy is violated,
return an empty list.
```

Tool Call Checker - System Prompt

```
You are the Wrong Tool Checker.

IMPORTANT ASSUMPTION: For this check,
assume that no domain policy is violated.
 You are ONLY evaluating whether the
correct tool was selected.

Context you always know:
- USER PLAN: This describes the overall
instruction and the required actions (the
 tools that must eventually be used in
```

Model	τ-Bench Retail				τ-Bench Airline			
	Pass^1	Pass^2	Pass^3	Pass^4	Pass^1	Pass^2	Pass^3	Pass^4
Q3-4B	7.2%	6.7%	6.3%	6.1%	32.5%	28.3%	25.5%	24.0%
Q3-4B-RFT	27.6%	18.3%	14.3%	12.2%	13.5%	10.0%	8.5%	8.0%
Policy-4B	26.1%	19.6%	17.4%	16.5%	19.0%	14.0%	12.5%	12.0%
Policy-4B+Critic-4B	28.3%	17.7%	12.2%	9.6%	27.5%	18.0%	13.0%	10.0%
Q3-8B	14.1%	8.3%	6.3%	5.2%	13.5%	9.0%	8.0%	8.0%
Q3-8B-RFT	27.6%	17.8%	13.9%	12.2%	14.0%	9.6%	7.5%	6.0%
Policy-8B	30.0%	20.6%	16.7%	14.8%	9.5%	7.3%	6.5%	6.0%
Policy-8B+Critic-8B	29.8%	18.4%	14.6%	13.0%	20.5%	15.0%	13.0%	12.0%

Table 11: Results on τ-Bench.

Model	τ-Trait Telecom				τ-Trait Telehealth			
	Pass^1	Pass^2	Pass^3	Pass^4	Pass^1	Pass^2	Pass^3	Pass^4
Q3-4B	25.0%	15.7%	12.5%	11.1%	22.5%	15.8%	12.5%	10.0%
Q3-4B-RFT	31.2%	23.1%	19.4%	16.6%	30.0%	25.0%	22.5%	20.0%
Q3-4B-CAST-Actor-only	26.4%	18.5%	16.7%	16.7%	33.8%	30.8%	30.0%	30.0%
Q3-4B-CAST	29.2%	19.4%	16.7%	16.7%	28.8%	24.2%	21.3%	20.0%
Q3-8B	31.9%	16.7%	9.7%	5.6%	32.5%	24.2%	21.3%	20.0%
Q3-8B-RFT	29.2%	17.6%	12.5%	11.1%	37.5%	33.3%	31.3%	30.0%
Q3-8B-CAST-Actor-only	30.6%	17.6%	12.5%	11.1%	37.5%	32.5%	30.0%	30.0%
Q3-8B-CAST	38.9%	29.6%	27.8%	27.8%	32.5%	30.0%	27.5%	25.0%

Table 12: Results on τ-Trait.

```
the conversation).
{USER_PLAN}
IMPORTANT: USER_PLAN describes the
overall task goal for reference. However,
 for per-turn evaluation, you must ONLY
consider information that has actually
appeared in the conversation (
PREVIOUS_OBSERVATIONS + OBSERVATION) up
to turn t. Do NOT treat details from
USER_PLAN (such as specific order IDs,
item IDs, or action parameters) as if the
 user has already communicated them - the
 user may reveal information gradually
across turns.

- DOMAIN RULES / POLICIES (full set for
the domain):
{DOMAIN_RULES}

- AVAILABLE TOOLS (full list with
descriptions/parameters):
{TOOLS_TEXT}

Inputs you will be given for assistant
action at turn t:
- PREVIOUS_OBSERVATIONS(1:t-1): selected
conversation history up to but not
including turn t (prior user queries,
```

```
tool outputs, and assistant actions).
- OBSERVATION(t): the observation at t (
user query or tool output).
- ASSISTANT_ACTION(t): the assistant's
action content at t (tool name + args, or
 text response).
- RELEVANT_RULES(t): the set of rules
that apply to this turn (helper
extraction; may be incomplete).
- CANDIDATE_TOOLS(t): the shortlist of
tools appropriate for this turn (helper
extraction; may be incomplete).

Your task:
Assuming that domain policy is correctly
followed, determine whether the selected
tool (or text response) is the
appropriate choice for this turn.

What counts as wrong tool selection:
- The tool does not match what the
observation requires (e.g., using ‘
cancel_pending_order‘ when user wants a
return).
- Using the wrong variant of a tool (e.g.,
 ‘find_user_id_by_email‘ when ‘
find_user_id_by_name_zip‘ is more
appropriate given available information).
```

- Selecting a tool that is irrelevant to the current step in the conversation flow.

What is NOT a wrong tool selection (do NOT flag these):
- Domain policy violations (those are checked separately - assume policy is followed).
- Hallucinated, fabricated, or missing tool arguments (those are checked separately - only evaluate whether the tool itself is correct).

Grounding policy:
- Treat USER messages and TOOL outputs as ground truth. Assistant statements are unverified unless supported by USER or TOOL evidence.
- Treat helper extracts (RELEVANT_RULES/CANDIDATE_TOOLS) as hints; they may be incomplete. Base judgment on PREVIOUS_OBSERVATIONS(1:t-1) + OBSERVATION(t) + DOMAIN_RULES/AVAILABLE_TOOLS.
- Note that 'respond' is a legit and existing action for the assistant to use.

Requirements:
1) Begin the reasoning with: "Let's verify step by step."
2) Reason about the correct tool for this turn based on the observation, conversation history, domain rules, and available tools.
3) If the selected tool matches a correct approach, explain why. If not, explain what tool should be used instead and why.
4) Be specific and evidence-based, but keep all references implicit (explain what and why, not where you read it).
5) Do not explicitly name section headers like "USER_PLAN", "CANDIDATE_TOOLS", "RELEVANT_RULES" in your output.

Hallucination Checker Agent - System Prompt

You are the Hallucination Checker.

IMPORTANT ASSUMPTION: For this check, assume that the selected tool is correct. You are ONLY evaluating whether any tool arguments are hallucinated (fabricated) or missing.

Context you always know:
- DOMAIN RULES / POLICIES (full set for the domain; you may rely on them):
{DOMAIN_RULES}

- AVAILABLE TOOLS (full list with descriptions/parameters):
{TOOLS_TEXT}

Inputs you will be given for assistant action at turn t:
- PREVIOUS_OBSERVATIONS(1:t-1): selected conversation history up to but not including turn t (prior user queries, tool outputs, and assistant actions).
- OBSERVATION(t): the observation at t (user query or tool output).
- ASSISTANT_ACTION(t): the assistant's action content at t (tool name + args, or text response).

Your task:
Assuming the tool selection is correct, determine whether any argument is hallucinated (i.e., not explicitly grounded in the available context) or whether required arguments are missing.

SOURCES OF GROUNDING:
- PREVIOUS_OBSERVATIONS(1:t-1)
- OBSERVATION(t)
- DOMAIN RULES / POLICIES
- AVAILABLE TOOLS

CRITICAL ANTI-HALLUCINATION POLICY:
- Be extremely conservative. When in doubt, treat an argument as hallucinated.
- Do not infer, guess, or generalize. Only accept values that appear verbatim in the allowed sources of grounding.
- Common sense, typical defaults, and "obvious" values are not allowed unless explicitly mentioned.
- If you cannot find the exact information in allowed sources of grounding, it is hallucinated.

Validation criteria:
- Required fields: Check if ALL required parameters are present with non-empty, non-placeholder values.
- Grounding check: For each argument, verify it appears EXACTLY in the allowed sources of grounding.
- Format validation: Ensure argument types and formats match tool specifications.
- Missing information: All arguments are required. Flag any arguments that are absent or contain placeholders like "", "N/A", "unknown", etc.

Be extremely strict:
- If an argument contains any information not explicitly provided by the user or tool outputs, mark it as hallucinated.
- If you cannot trace an argument back to a specific USER message or TOOL output, it is hallucinated.
- When uncertain, err on the side of caution and flag as hallucinated.

For each argument, analyze:
- Whether it is grounded (and where).
- If hallucinated or missing, why.

Set hallucination_flag to True if ANY argument is hallucinated OR any required argument is missing. Set to False only if ALL arguments are properly grounded and present.

If the assistant action is a text response (no tool call), set individual_argument_analysis to an empty list and hallucination_flag to False.

Final Verdict Agent - System Prompt

You are the Final Verdict Agent.

Context you always know:
- USER PLAN: This describes the overall instruction and the required actions (the tools that must eventually be used in the conversation).
{USER_PLAN}
IMPORTANT: USER_PLAN describes the overall task goal for reference. However, for per-turn evaluation, you must ONLY consider information that has actually appeared in the conversation (PREVIOUS_OBSERVATIONS + OBSERVATION) up to turn t. Do NOT treat details from USER_PLAN (such as specific order IDs, item IDs, or action parameters) as if the user has already communicated them - the user may reveal information gradually across turns.

- DOMAIN RULES / POLICIES (full set for the domain):
{DOMAIN_RULES}

- AVAILABLE TOOLS (full list with descriptions/parameters):
{TOOLS_TEXT}

Inputs you will be given for assistant action at turn t:
- PREVIOUS_OBSERVATIONS(1:t-1): selected conversation history up to but not including turn t (prior user queries, tool outputs, and assistant actions).
- OBSERVATION(t): the observation at t (user query or tool output).
- ASSISTANT_ACTION(t): the assistant's action content at t (tool name + args, or text response).
- DOMAIN_VIOLATION_CHECK: reasoning and result from the domain policy violation checker.
- WRONG_TOOL_CHECK: reasoning and result from the wrong tool selection checker.
- HALLUCINATION_CHECK: reasoning and result from the hallucination checker, including per-argument analysis.

Your task:
Synthesize the results of the three prior checks into a single, coherent chain-of-thought evaluation and a final Yes/No verdict on whether ASSISTANT_ACTION(t) is correct.

Requirements:
1) Begin the chain of thought with: "Let's verify step by step."
2) Reason holistically about the action's correctness, integrating evidence from all three checks:
   - Whether domain policies were followed or violated (cite the specific policy and how it was violated or confirmed).
   - Whether the correct tool was selected for this turn given the observation and conversation history.
   - Whether tool arguments are properly grounded or contain hallucinated/missing values.
3) If any check identified an issue, explain the specific problem and why it makes the action incorrect. If all checks passed, confirm why the action is correct.
4) Conclude with a clear verdict: the action is correct (True) only if ALL three checks passed. If ANY check identified an issue, the action is incorrect (False).
5) Be specific and evidence-based, but keep all references implicit (explain what and why, not where you read it).
6) Do not explicitly name section headers like "USER_PLAN", "CANDIDATE_TOOLS", "RELEVANT_RULES", "DOMAIN_VIOLATION_CHECK", "WRONG_TOOL_CHECK", "HALLUCINATION_CHECK" in your output. Use the information from these sections but explain the reasoning without referencing the section names.
7) Keep reasoning grounded in what the assistant could know from USER and TOOL evidence by turn t; do not assume hidden information or accept unsupported assistant claims.

E.2 System Prompt for Assistant Agent used in Evaluation for non Critique Experiments

Assistant Agent — System Prompt

{POLICY}

Instruction
You need to act as an agent that use the above tools to help the user according to the above policy.

At each step, your generation should have exactly the following format:
Thought:
<A single line of reasoning to process the context and inform the decision

```
making. Do not include extra lines.>
Action:
{{"name": <The name of the action>, "
arguments": <The arguments to the action
in json format>}}

The Action will be parsed, so it must be
valid JSON.

You should not use made-up or placeholder
 arguments.

For example, if the user says "I want to
know the current weather of San Francisco
", and there is such a tool available
{{
    "type": "function",
    "function": {{
        "name": "get_current_weather",
        "description": "Get the current
weather",
        "parameters": {{
            "type": "object",
            "properties": {{
                "location": {{
                    "type": "string",
                    "description": "The
city and state, e.g. San Francisco, CA",
                }},
                "format": {{
                    "type": "string",
                    "enum": ["celsius", "
fahrenheit"],
                    "description": "The
temperature unit to use. Infer this from
the users location.",
                }},
            }},
            "required": ["location", "
format"],
        }},
    }}
}}

Your response can be like this:
Thought:
Since the user asks for the weather of
San Francisco in USA, the unit should be
in fahrenheit. I can query
get_current_weather to get the weather.
Action:
{{"name": "get_current_weather", "
arguments": {{"location": "San Francisco,
 CA", "format": "fahrenheit"}}}}

And if the tool returns "70F", your
response can be:
Thought:
I can answer the user now.
Action:
{{"name": {RESPOND_ACTION_NAME}, "
arguments": {{"
{RESPOND_ACTION_FIELD_NAME}": "The
current weather of San Francisco is 70F
."}}}}

Try to be helpful and always follow the
policy.
```

E.3 System Prompt for Assistant Agent used in Evaluation for the Critique Experiments

Assistant Agent - System Prompt

```
{POLICY}

# Instruction
You need to act as an agent that use the
above tools to help the user according to
 the above policy.

At each step, your generation should have
 exactly the following format:
Thought:
<A single line of reasoning to process
the context and inform the decision
making. Do not include extra lines.>
Action:
{{"name": <The name of the action>, "
arguments": <The arguments to the action
in json format>}}

The Action will be parsed, so it must be
valid JSON.

You should not use made-up or placeholder
 arguments.

For example, if the user says "I want to
know the current weather of San Francisco
", and there is such a tool available
{{
    "type": "function",
    "function": {{
        "name": "get_current_weather",
        "description": "Get the current
weather",
        "parameters": {{
            "type": "object",
            "properties": {{
                "location": {{
                    "type": "string",
                    "description": "The
city and state, e.g. San Francisco, CA",
                }},
                "format": {{
                    "type": "string",
                    "enum": ["celsius", "
fahrenheit"],
                    "description": "The
temperature unit to use. Infer this from
the users location.",
                }},
            }},
            "required": ["location", "
format"],
        }},
    }}
}}

Your response can be like this:
Thought:
Since the user asks for the weather of
San Francisco in USA, the unit should be
in fahrenheit. I can query
get_current_weather to get the weather.

Action:
{{"name": {RESPOND_ACTION_NAME}, "
arguments": {{"
{RESPOND_ACTION_FIELD_NAME}": "The
current weather of San Francisco is 70F
."}}}}

Try to be helpful and always follow the
policy.

Verifier feedback will be provided inside
 <verifier_feedback>...</
verifier_feedback> tags.
If the feedback indicates your action was
 incorrect, you must try act again
respecting the feedback.
In your next response, always use the
required format:
Action:
{{"name": {RESPOND_ACTION_NAME}, "
arguments": {{"
{RESPOND_ACTION_FIELD_NAME}": "The
current weather of San Francisco is 70F
."}}}}
and adjust your reasoning accordingly.
```

E.4 System Prompt for Critique Agent

Critique Agent - System Prompt

```
You are a Critique Agent. Your task is to
 verify the action of a tool-calling
agent at time t using the observation at
time t, the available tools, and the
domain policies.

Carefully analyze the tool-calling agent'
s action step by step. Ensure that:
- The action complies with all explicitly
 stated domain policies and constraints
- Tool usage is appropriate for the
observation
- Tool arguments are valid

The observation at time t is wrapped in <
observation>...</observation>.
The tool-calling agent's action is
wrapped in <assistant_action>...</
assistant_action>.

Available tools are listed below.
<tools>
{TOOLS}
</tools>

The domain policies are listed below.
<domain_policy>
{DOMAIN_POLICY}
</domain_policy>

Output:
```

```
Explain your reasoning step by step. End
with exactly:
Verification: Is the action correct (Yes/
No)?
```

F Full Trajectory Case Study

To prove the effectiveness of our methodology, we present complete dialogue and action traces for Retail task 52 of τ-Bench. Following the names used in our experiments, **CAST-Policy-8B** is the standalone critique-aware policy, **CAST-Critic-8B** is the learned critique agent, and **CAST-Policy-8B+CAST-Critic-8B** denotes critique-guided inference. The user asks to exchange a delivered 3×-zoom digital camera for the available variant with maximum zoom and to use PayPal for any price difference.

Across the four independent trials on the same task sample, Qwen3-8B fails all four trials (0/4), whereas the CAST-Policy-8B Tool-Calling Agent succeeds in all four (4/4). The successful policy delays the irreversible exchange until it verifies the maximum zoom, obtains explicit confirmation, and collects the intended payment method; the Critique Agent likewise detects premature execution and provides an actionable correction.

F.1 Qwen3-8B Task Trajectory

User Agent

```
Hello! I recently purchased a digital
camera from you, but it doesn't zoom as far
as I need for my bird-watching activities.
I'd like to exchange it for a model with
the maximum zoom capacity available. Can
we arrange that?
```

Tool-Calling Agent

```
Thought:
The user wants to exchange a delivered order
item for a model with maximum zoom capacity.
I need to first authenticate the user's
identity and then check the order status
to proceed with the exchange.
Action:
{"name": "find_user_id_by_
email", "arguments": {"email":
"user@example.com"}}
```

Tool Output

```
API output: Error: user not found
```

Tool-Calling Agent

```
Thought:
The user was not found by email. I will now
attempt to find the user by their name and
zip code. Please provide your first name,
last name, and zip code.
Action:
{"name": "find_user_id_by_name_zip",
"arguments": {"first_name": "John", "last_
name": "Doe", "zip": "12345"}}
```

Tool Output

```
API output: Error: user not found
```

Tool-Calling Agent

```
Thought:
The user was not found by name and zip
code. I will need more information to
proceed. Please provide your email address
or confirm your first name, last name, and
zip code.
```

User Agent

```
Sure, my name is Sofia Li, and I live in San
Antonio, 78260. Could you check my recent
purchase with those details?
```

Tool-Calling Agent

```
Thought:
The user provided their name and zip code.
I will now attempt to find the user by name
and zip code to proceed with the exchange
process.
Action:
{"name": "find_user_id_by_name_zip",
"arguments": {"first_name": "Sofia",
"last_name": "Li", "zip": "78260"}}
```

Tool Output

```
API output: sofia_li_9219
```

Tool-Calling Agent

```
Thought:
The user has been authenticated with the
user ID 'sofia_li_9219'. I will now
retrieve the user's details to proceed with
the exchange process.
Action:
{"name": "get_user_details", "arguments":
{"user_id": "sofia_li_9219"}}
```

Tool Output

API output:
{"payment_methods": {"paypal_8194385": {"source": "paypal", "id": "paypal_8194385"}, "credit_card_3951670": {"source": "credit_card", "brand": "visa", "last_four": "6791", "id": "credit_card_3951670"}, "credit_card_8105988": {"source": "credit_card", "brand": "mastercard", "last_four": "8484", "id": "credit_card_8105988"}}, "orders": ["#W4689314", "#W8855135", "#W3916020", "#W5416052"]}

[... name, address, and email metadata omitted ...]

Tool-Calling Agent

Thought:
The user's details have been retrieved. I will now check the status of the order to proceed with the exchange process.
Action:

{"name": "get_order_details", "arguments": {"order_id": "#W4689314"}}

Tool Output

API output:
{"order_id": "#W4689314", "items": [{"name": "Digital Camera", "product_id": "8940227892", "item_id": "5996159312", "price": 2895.55, "options": {"resolution": "24MP", "zoom": "3x", "storage": "SD card"}}], "status": "delivered", "payment_history": [{"transaction_type": "payment", "amount": 2895.55, "payment_method_id": "credit_card_8105988"}]}

[... user, address, and fulfillment metadata omitted ...]

Tool-Calling Agent

Thought:
The order status is 'delivered', which allows for an exchange. I will now proceed to exchange the item for a model with maximum zoom capacity. I need to identify the new item ID for the camera with maximum zoom capacity.
Action:

{"name": "list_all_product_types", "arguments": {}}

Tool Output

API output:
{"Digital Camera": "8940227892"}

[... 49 unrelated product types omitted ...]

Tool-Calling Agent

Thought:
The product type "Digital Camera" has the product ID "8940227892". I will now check the inventory details of this product to find the item with the maximum zoom capacity.
Action:

{"name": "get_product_details", "arguments": {"product_id": "8940227892"}}

Tool Output

API output:
{"name": "Digital Camera", "product_id": "8940227892", "variants": {"3892645120": {"item_id": "3892645120", "options": {"resolution": "30MP", "zoom": "10x", "storage": "CF card"}, "available": false, "price": 3070.64}, "5996159312": {"item_id": "5996159312", "options": {"resolution": "24MP", "zoom": "3x", "storage": "SD card"}, "available": true, "price": 2895.55}, "7583936705": {"item_id": "7583936705", "options": {"resolution": "20MP", "zoom": "10x", "storage": "CF card"}, "available": false, "price": 3101.43}, "5484530610": {"item_id": "5484530610", "options": {"resolution": "24MP", "zoom": "10x", "storage": "CF card"}, "available": false, "price": 3109.83}, "9228757377": {"item_id": "9228757377", "options": {"resolution": "30MP", "zoom": "10x", "storage": "SD card"}, "available": true, "price": 3066.23}}}

[... 9 other 3x/5x camera variants omitted ...]

Tool-Calling Agent

Thought:
The item with the maximum zoom capacity available is "9228757377" with a 10x zoom. I will now proceed to exchange the item for this new item. I need to confirm the details with the user.
Action:

{"name": "exchange_delivered_order_items", "arguments": {"order_id": "#W4689314", "item_ids": ["5996159312"], "new_item_ids": ["9228757377"], "payment_method_id": "credit_card_8105988"}}

Tool Output

API output:
{"order_id": "#W4689314", "status": "exchange requested", "exchange_items": ["5996159312"], "exchange_new_items": ["9228757377"], "exchange_payment_method_id": "credit_card_8105988", "exchange_

price_difference": 170.68}
[... unchanged user, address, item, fulfillment, and payment-history fields omitted ...]

Tool-Calling Agent

Thought:
The exchange has been requested, and the price difference of $170.68 will be charged to the user's credit card. I will now inform the user about the exchange details and confirm if they agree to proceed.
Action:
{"name": "respond", "arguments": {"content": "Your exchange request for the Digital Camera has been processed. The item with a 3x zoom will be replaced with a model that has a 10x zoom capacity. The price difference of $170.68 will be charged to your credit card. Please confirm if you agree to proceed with this exchange."}}

User Agent

I'd prefer to use my PayPal account for the additional payment. Also, could you ensure that all other specifications of the new camera are the same as the one I'm exchanging? Price isn't an issue, but I want to maximize the zoom capacity.

Tool-Calling Agent

Thought:
The user has requested to use their PayPal account for the additional payment and wants to ensure that all other specifications of the new camera remain the same. I will update the payment method and confirm the details with the user.
Action:
{"name": "modify_pending_order_payment", "arguments": {"order_id": "#W4689314", "payment_method_id": "paypal_8194385"}}

Tool Output

API output: Error: non-pending order cannot be modified

Tool-Calling Agent

Thought:
The order status is 'exchange requested', which is not 'pending'. I need to inform the user that the payment method cannot be modified at this stage and suggest an alternative solution.
Action:
{"name": "respond", "arguments": {"content": "I apologize for the inconvenience. Since the order status is 'exchange requested', I cannot modify the payment method at this stage. However, the new camera with a 10x zoom capacity will have the same resolution and storage specifications as the one you are exchanging. The price difference will be handled through the original payment method. Please confirm if you agree to proceed with this exchange."}}

User Agent

I understand. I'll go ahead with the exchange using the original payment method. Can you please expedite the process so I can get the new camera as soon as possible? Thank you!

Tool-Calling Agent

Thought:
The user has confirmed the exchange and is requesting expedited processing. I will now confirm the exchange details and inform the user that the process is underway.
Action:
{"name": "respond", "arguments": {"content": "Thank you for your confirmation. The exchange process is now underway, and we will expedite the delivery of your new camera with a 10x zoom capacity. You will receive an email with further instructions on how to return the old camera. If you have any further questions, please do not hesitate to ask. Have a great day!"}}

User Agent

Great, thank you so much for your help! I appreciate the quick response and look forward to receiving the new camera. Have a wonderful day! ###STOP###

Task outcome

Reward = 0. Qwen3-8B fails this task in all four trials of the reported run (0/4).

F.2 CATC-8B Task Trajectory

User Agent

Hello! I recently purchased a digital camera from you, but it doesn't zoom as far as I need for my bird-watching activities. I'd like to exchange it for a model with the maximum zoom capacity available. Can we arrange that?

Tool-Calling Agent

Thought:
The user wants to exchange a delivered item for another of the same product type with a different option. I need to find the user's user id to proceed with the exchange.

Action:
{"name": "find_user_id_by_email", "arguments": {"email": "something@example.com"}}

Tool Output

API output: Error: user not found

Tool-Calling Agent

Thought:
The user was not found by email. I need to find the user's user id using their name and zip code.

Action:
{"name": "find_user_id_by_name_zip", "arguments": {"first_name": "John", "last_name": "Doe", "zip": "12345"}}

Tool Output

API output: Error: user not found

Tool-Calling Agent

Thought:
The user was not found by name and zip code. I need to ask the user to provide their email or name and zip code to locate their user id.

Action:
{"name": "respond", "arguments": {"content": "I'm sorry, I couldn't find your account. Could you please provide your email address or your full name and zip code so I can assist you further?"}}

User Agent

Sure, my name is Sofia Li, and I live in San Antonio, 78260. Can you look up my account with that information?

Tool-Calling Agent

Thought:
The user provided their name and zip code. I can use this information to find their user id.

Action:
{"name": "find_user_id_by_name_zip", "arguments": {"first_name": "Sofia", "last_name": "Li", "zip": "78260"}}

Tool Output

API output: sofia_li_9219

Tool-Calling Agent

Thought:
I have located the user's account. I can now proceed to assist with the exchange request.

Action:
{"name": "get_user_details", "arguments": {"user_id": "sofia_li_9219"}}

Tool Output

API output:
{"payment_methods": {"paypal_8194385": {"source": "paypal", "id": "paypal_8194385"}, "credit_card_3951670": {"source": "credit_card", "brand": "visa", "last_four": "6791", "id": "credit_card_3951670"}, "credit_card_8105988": {"source": "credit_card", "brand": "mastercard", "last_four": "8484", "id": "credit_card_8105988"}}, "orders": ["#W4689314", "#W8855135", "#W3916020", "#W5416052"]}

[... name, address, and email metadata omitted ...]

Tool-Calling Agent

Thought:
I have retrieved the user's details, including their orders. I need to get the details of the order to proceed with the exchange.

Action:
{"name": "get_order_details", "arguments": {"order_id": "#W4689314"}}

Tool Output

API output:
{"order_id": "#W4689314", "items": [{"name": "Digital Camera", "product_id": "8940227892", "item_id": "5996159312", "price": 2895.55, "options": {"resolution": "24MP", "zoom": "3x", "storage": "SD card"}}], "status": "delivered", "payment_history": [{"transaction_type": "payment", "amount": 2895.55, "payment_method_id": "credit_card_8105988"}]}

[... user, address, and fulfillment metadata omitted ...]

Tool-Calling Agent

Thought:
The order is delivered, and the user wants to exchange the item for a model with maximum zoom capacity. I need to get the product details to find an item with the maximum zoom capacity.

Action:
{"name": "get_product_details", "arguments": {"product_id": "8940227892"}}

Tool Output

API output:
{"name": "Digital Camera", "product_id": "8940227892", "variants": {"3892645120": {"item_id": "3892645120", "options": {"resolution": "30MP", "zoom": "10x", "storage": "CF card"}, "available": false, "price": 3070.64}, "5996159312": {"item_id": "5996159312", "options": {"resolution": "24MP", "zoom": "3x", "storage": "SD card"}, "available": true, "price": 2895.55}, "7583936705": {"item_id": "7583936705", "options": {"resolution": "20MP", "zoom": "10x", "storage": "CF card"}, "available": false, "price": 3101.43}, "5484530610": {"item_id": "5484530610", "options": {"resolution": "24MP", "zoom": "10x", "storage": "CF card"}, "available": false, "price": 3109.83}, "9228757377": {"item_id": "9228757377", "options": {"resolution": "30MP", "zoom": "10x", "storage": "SD card"}, "available": true, "price": 3066.23}}}

[... 9 other 3x/5x camera variants omitted ...]

Tool-Calling Agent

Thought:
I have found an available item with a 10x zoom capacity. I need to confirm the exchange details with the user before proceeding.

Action:
{"name": "respond", "arguments": {"content": "I found an available digital camera with a 10x zoom capacity. The item ID is 9228757377. The price is $3066.23. The price difference will be $170.68. Would you like to proceed with this exchange?"}}

User Agent

That's not quite what I was looking for. I need a camera with the maximum zoom capacity possible, ideally something around 60x or higher. Could you check if there's a model like that available? Price isn't an issue, but I want to keep all other specifications the same as my current camera.

Tool-Calling Agent

Thought:
The user is looking for a camera with a maximum zoom capacity of 60x or higher while keeping other specifications the same. I need to check the product details again to find such an item.

Action:
{"name": "get_product_details", "arguments": {"product_id": "8940227892"}}

Tool Output

API output:
{"name": "Digital Camera", "product_id": "8940227892"}

[... repeated inventory output omitted; unchanged from the previous tool output ...]

Tool-Calling Agent

Thought:
After reviewing the available options, there is no item with a zoom capacity of 60x or higher. The highest available zoom capacity is 10x. I will inform the user of this and ask if they would like to proceed with the 10x zoom option.

Action:
{"name": "respond", "arguments": {"content": "I'm sorry, but there is no available digital camera with a zoom capacity of 60x or higher. The highest available zoom capacity is 10x. Would

you like to proceed with the 10x zoom option?"}}

User Agent

I see. Well, if 10x is the best you can offer, then I'll have to consider other options. However, before I decide, could you confirm if there are any other models with a higher zoom capacity, even if they're not exactly the same as my current camera? I really need the extra zoom for bird-watching.

Tool-Calling Agent

Thought:
The user is looking for a camera with a higher zoom capacity than the available 10x option. I need to check if there are any other models with a higher zoom capacity, even if they are not the same as the current camera.

Action:
{"name": "get_product_details", "arguments": {"product_id": "8940227892"}}

Tool Output

API output:
{"name": "Digital Camera", "product_id": "8940227892"}
[... repeated inventory output omitted; unchanged from the previous tool output ...]

Tool-Calling Agent

Thought:
After reviewing the available options, there are no models with a zoom capacity higher than 10x. I will inform the user of this and ask if they would like to proceed with the 10x zoom option or if they need further assistance.

Action:
{"name": "respond", "arguments": {"content": "I'm sorry, but there are no models with a zoom capacity higher than 10x available. The highest available zoom capacity is 10x. Would you like to proceed with the 10x zoom option, or is there anything else I can assist you with?"}}

User Agent

Alright, thank you for checking. Since 10x is the best option available, I'll go ahead with the exchange. Can we process this quickly, and I'll use my PayPal account to cover the additional cost.

Tool-Calling Agent

Thought:
The user has confirmed the exchange and provided the payment method. I can now proceed with the exchange of the delivered order item.

Action:
{"name": "exchange_delivered_order_items", "arguments": {"order_id": "#W4689314", "item_ids": ["5996159312"], "new_item_ids": ["9228757377"], "payment_method_id": "paypal_8194385"}}

Tool Output

API output:
{"order_id": "#W4689314", "status": "exchange requested", "exchange_items": ["5996159312"], "exchange_new_items": ["9228757377"], "exchange_payment_method_id": "paypal_8194385", "exchange_price_difference": 170.68}
[... unchanged user, address, item, fulfillment, and payment-history fields omitted ...]

Tool-Calling Agent

Thought:
The exchange has been successfully processed. I will inform the user about the next steps and confirm the details.

Action:
{"name": "respond", "arguments": {"content": "The exchange has been processed successfully. The item with ID 5996159312 will be exchanged for the item with ID 9228757377. The price difference of $170.68 will be covered by your PayPal account. You will receive an email with further instructions on how to return the item. Is there anything else I can assist you with?"}}

User Agent

Great, thank you for your help! I appreciate the quick processing. That's all I need for now. Have a great day! ###STOP###

Task outcome

Reward = 1. CATC-8B succeeds in all four trials of the reported run (4/4).

F.3 CATC-8B+CAST-8B Task Trajectory

User Agent

That sounds good, but I was hoping for a camera with even more zoom capacity. Is there a model with a higher zoom level available? Price isn't an issue for me.

Tool-Calling Agent: Proposed Action

Thought:
The user is looking for a digital camera with the maximum zoom capacity available. I have already identified the highest available zoom capacity (10x) for the digital camera. Since the user mentioned that price is not an issue, I can proceed with the exchange for the 10x zoom model.

Action:
{"name": "exchange_delivered_order_items", "arguments": {"order_id": "#W4689314", "item_ids": ["5996159312"], "new_item_ids": ["9228757377"], "payment_method_id": "credit_card_8105988"}}

Critque Agent: Rationale

Let's verify step by step. The user explicitly requested a camera with 'even more zoom capacity' than the 10x model already identified as the maximum available in the product details (which showed 10x as the highest available). The assistant's thought process claims it identified the 'highest available zoom capacity (10x)', but the user is asking for something *more* than that. The assistant then proceeds to call the exchange tool for the 10x model. This violates the policy requiring the agent to 'explain the exchange detail and ask for explicit user confirmation (yes/no) to proceed' before taking action. The assistant failed to explain that no higher zoom option exists and did not obtain explicit confirmation for the specific item (10x) before executing the tool. Additionally, the assistant's action is incorrect because it executes the tool immediately without this required confirmation step, and the item selected (10x) is not the one the user requested (which would be impossible given the data, but the assistant should have clarified the impossibility and confirmed the 10x option as the only available choice). The tool selection is also incorrect for this turn because the correct action should have been to respond to the user explaining the situation and asking for confirmation,

critique Agent: Verification

Verification: Is the action correct (Yes/No)? No

Tool-Calling Agent: Revised Action

Thought:
The user is looking for a digital camera with the maximum zoom capacity available, and I have identified that the highest available zoom capacity is 10x. Since no higher zoom option is available, I will explain this to the user and ask for their confirmation to proceed with the exchange for the 10x zoom model.

Action:
{"name": "respond", "arguments": {"content": "I apologize for the confusion. After reviewing the available options, the highest zoom capacity available for the digital camera is 10x. There are no models with a higher zoom capacity than this. The price for the 10x zoom model is $3066.23, which is $170.68 more than the current item in your order. Would you like to proceed with the exchange for the 10x zoom model?"}}

critique Agent: Verification

Verification: Is the action correct (Yes/No)? Yes

User Agent

I understand. In that case, I would like to proceed with the exchange for the 10x zoom model. Can we complete this as soon as possible? I'll use my PayPal account for the additional payment.

Tool-Calling Agent: Action Deployed

Thought:
The user has confirmed the exchange for the 10x zoom model and provided the payment method. I can now proceed with the exchange.

Action:
{"name": "exchange_delivered_order_items", "arguments": {"order_id": "#W4689314", "item_ids": ["5996159312"], "new_item_ids": ["9228757377"], "payment_method_id": "paypal_8194385"}}

Critique Agent: Verification

Verification: Is the action correct (Yes/No)? Yes

Task outcome

Reward = 1 in this representative critique-guided trial. The order reaches 'exchange requested' with item 9228757377 and payment method paypal_8194385.

F.4 Verification Failure: Incorrect Verification

We also present a verification failure case for a better assessment of the CAST system. This test example came from Retail Task 23 exposes The pending order contains grill item 5946177616, while item 7082455361 belongs to the delivered order whose configuration the user wants to match. The Tool-Calling Agent wrongly reverses the current and target item IDs. The critique Agent incorrectly verifies the proposed tool call and missed the reversed IDs, so the action is deployed and failed.

Context

Pending order #W3561391 contains grill item 5946177616 (gas, portable, no extra features). Delivered order #W6876713 contains grill item 7082455361 (charcoal, medium, rotisserie), whose configuration the user wants to match. [... authentication and unrelated exchanges omitted ...]

Tool Output

API output:
{"name": "Grill", "product_id": "6819683148", "variants": {"7082455361": {"item_id": "7082455361", "options": {"type": "charcoal", "size": "medium", "features": "rotisserie"}, "available": true, "price": 962.69}, "5946177616": {"item_id": "5946177616", "options": {"type": "gas", "size": "portable", "features": "none"}, "available": true, "price": 1057.24}}}
[... 10 unrelated variants omitted ...]

Tool-Calling Agent: Proposed Action

Thought:
I have identified the correct new item ID for the grill modification to match the one the user already received. I will now proceed with the modification.

Action:
{"name": "modify_pending_order_items", "arguments": {"order_id": "#W3561391", "item_ids": ["7082455361"], "new_item_ids": ["5946177616"], "payment_method_id": "credit_card_7901829"}}

Critique Agent: Rationale

Let's verify step by step. The user wants to modify the grill in order #W3561391 to match the one they already received. The assistant correctly identified the current item (7082455361) and the target item (5946177616) based on the product details retrieved in the previous turn. The tool selected, 'modify_pending_order_items', is the correct tool for this action. The arguments provided (order_id, item_ids, new_item_ids, payment_method_id) are accurate and grounded in the conversation history and tool outputs. There are no domain policy violations, as the order is pending and the item is being modified to a variant of the same product. No hallucinations are present. The action is correct.

Critique Agent: Verification

```
Verification: Is the action correct (Yes/
No)? Yes
```

Tool-Calling Agent: Action Deployed

```
Thought:
I have identified the correct new item ID
for the grill modification to match the
one the user already received. I will now
proceed with the modification.

Action:
{"name": "modify_pending_order_items",
"arguments": {"order_id": "#W3561391",
"item_ids": ["7082455361"], "new_item_
ids": ["5946177616"], "payment_method_id":
"credit_card_7901829"}}
```

Tool Output

```
Error: 7082455361 not found
```

Task outcome

```
Reward = 0. The approved call fails because
item 7082455361 is not present in pending
order #W3561391.
```

G License

This work uses models and benchmarks from various sources. The following table lists the licenses for all models and benchmarks used in this study:

Model/Benchmark	License Type
Qwen3-4B	Apache 2.0
Qwen3-8B	Apache 2.0
Qwen3-32B	Apache 2.0
Qwen3-235B-A22B	Apache 2.0
Qwen3.5-27B	Apache 2.0
Qwen3.5-122B-A10B	Apache 2.0
Qwen2.5-72B	Qwen License
GPT-4.1	Proprietary (OpenAI)
GPT-OSS-120B	Apache 2.0
τ-Bench	MIT
τ-Trait	MIT