EDGEGUARD-AGENT: EDGE-LEVEL VERIFICATION AND POLICY LEARNING FOR TOOL-USING AGENTS UNDER INCOMPLETE INFORMATION

Anonymous authors
Paper under double-blind review

ABSTRACT

Tool-using agents are increasingly applied to stateful tasks such as order management, flight rebooking, and customer support, where requests are often incomplete, states are partially observable, and write actions may cause real side effects. Existing methods typically handle clarification, correction, and safety separately, making unified verification difficult. We propose **EdgeGuard-Agent**, an edge-level verification and policy learning framework for tool use under incomplete information. It treats each action as a state-transition edge and checks whether it is legal, grounded, executable, task-advancing, and non-redundant. At runtime, it focuses on risky or abnormal actions, including unsupported writes, invalid tools, missing arguments, state conflicts, and repeated actions, and then either accepts the action, replaces it with a safe executable repair, or blocks it if no safe option exists. Deterministic checks further prevent ungrounded arguments, unsafe writes, invalid confirmations, and cross-turn inconsistencies. During training, the same checks identify the first invalid action, restore the environment, apply a verified repair, and regenerate a causally consistent trajectory for supervised fine-tuning and GRPO with both task-level and edge-level rewards. Experiments on CAR-Bench, STATE-Bench, and τ^3-Bench-Ambiguous show improvements in task success of 29.60, 8.13, and 10.43 percentage points, respectively, with fewer premature actions and incorrect writes. On CAR-Bench, EdgeGuard-Agent-guided SFT improves task success from 27.69% to 35.30%, and GRPO further raises it to 40.42%. These results show that edge-level verification is a simple and effective way to improve both runtime reliability and policy learning in multi-turn tool-using agents.

1 INTRODUCTION

Tool-using agents increasingly operate in stateful applications such as order management, flight rebooking, account services, and customer support. Unlike isolated function calling, these tasks require multi-turn interaction with users and tools, compliance with domain policies, and updates to persistent environment states (Yao et al., 2024; Barres et al., 2025; Lu et al., 2025; Trivedi et al., 2024). Reliable execution is difficult because user requests are often incomplete, task-relevant environment states are only partially observed, and mutating actions may cause irreversible side effects.

An agent must therefore determine which facts are already supported, whether missing information should be obtained from the user or the environment, and whether a proposed write satisfies its execution conditions. Existing agents often overlook missing conditions, hallucinate tool arguments, or act without sufficient evidence (Yang et al., 2024; Kokane et al., 2024; Kirmayr et al., 2026). Clarification methods improve when to ask the user (Wang et al., 2025c; Suri et al., 2026), but do not fully address environment-recoverable uncertainty or write readiness.

Existing solutions approach these problems from two directions. At inference time, reflection and self-correction revise actions using model or environment feedback (Madaan et al., 2023; Shinn et al., 2023; Gou et al., 2024; Huang et al., 2024), while runtime safeguards inspect risky plans or tool calls before execution (Xiang et al., 2024; Chen et al., 2025; Wang et al., 2025a; Cuadron et al., 2025; Mou et al., 2026). However, open-ended replanning may overwrite valid local decisions, and

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

existing guards rarely verify information ownership, argument grounding, execution preconditions, and state progress jointly. At training time, multi-turn agent RL commonly relies on delayed task rewards (Qi et al., 2025; Wei et al., 2025; Wang et al., 2025d;b). Process and tool-specific rewards provide finer supervision (Qian et al., 2026; Choudhury, 2025; Xi et al., 2026; Li et al., 2026), but learned evaluators may be noisy, while directly editing a failed action can make subsequent observations inconsistent with the repaired trajectory.

Our key observation is that many failures under incomplete information are not global planning failures, but *local state-transition errors*. A proposed action may use the wrong information source, rely on unsupported evidence, violate a precondition, repeat an existing action, or introduce an unjustified side effect. We therefore treat each action as a candidate state-transition edge and audit it along six dimensions: *legality, source, grounding, risk, progress, and redundancy*.

Based on this view, we propose **EdgeGuard-Agent**, a unified framework for runtime verification and policy learning. At inference time, EdgeGuard-Agent selectively verifies risky or abnormal proposals, reconstructs the decision-relevant state, distinguishes user-side unknowns from environment-side unknowns, and accepts, locally repairs, or blocks the current edge. A deterministic gate further enforces schema validity, evidence grounding, write safety, and episode consistency before execution. During training, the same auditor identifies invalid edges, causally regenerates repaired trajectory suffixes through environment re-execution, and converts edge-level judgments into supervision for SFT and GRPO (Shao et al., 2024).

We evaluate EdgeGuard-Agent on CAR-Bench, STATE-Bench, and our incomplete-information extension, τ^3-Bench-Ambiguous. Runtime EdgeGuard-Agent improves task success by 29.60, 8.13, and 10.43 percentage points, respectively, while reducing premature execution and wrong writes. On CAR-Bench, EdgeGuard-Agent-curated SFT improves task success from 27.69% to 35.30%, and EdgeGuard-Agent-guided GRPO further increases it to 40.42%.

Contributions. Our contributions are threefold: (1) we formulate incomplete-information tool use as verifiable state transitions over ASKUSER, READTOOL, WRITETOOL, ANSWER, and ABORT; (2) we introduce a shared edge auditor for selective runtime verification, local repair, causal trajectory reconstruction, and process-level policy learning; and (3) we demonstrate consistent improvements in task success, stability, safety, and efficiency across three stateful tool-use benchmarks.

2 RELATED WORK

2.1 STATEFUL AND INTERACTIVE TOOL-USING AGENTS

Tool-use evaluation has expanded from isolated API correctness to multi-turn interaction, environment feedback, and persistent state. MINT, AppWorld, ToolSandbox, and the τ-bench family evaluate increasingly realistic tool-use settings involving controllable applications, state dependencies, user simulation, and business policies (Wang et al., 2024; Trivedi et al., 2024; Lu et al., 2025; Yao et al., 2024; Barres et al., 2025). CAR-Bench further targets ambiguity, missing information, unavailable tools, and repeated-run consistency (Kirmayr et al., 2026), while STATE-Bench evaluates long-horizon enterprise workflows (Microsoft, 2026). These benchmarks expose intermediate decision failures, but do not provide a unified mechanism for verifying the source, evidence, preconditions, progress, and risk of each transition.

2.2 CLARIFICATION, SELF-CORRECTION, AND RUNTIME VERIFICATION

Clarification methods train agents to ask when user requests are ambiguous or incomplete (Wang et al., 2025c; Suri et al., 2026). Self-Refine, Reflexion, CRITIC, and QueryAgent revise decisions through self-generated or environment-grounded feedback (Madaan et al., 2023; Shinn et al., 2023; Gou et al., 2024; Huang et al., 2024). Runtime safeguards instead inspect plans or actions before execution (Xiang et al., 2024; Chen et al., 2025; Wang et al., 2025a; Cuadron et al., 2025; Mou et al., 2026). EdgeGuard-Agent differs by treating information recovery and safe execution as a single local edge-verification problem. It distinguishes user-owned from environment-owned unknowns, preserves valid base-agent proposals, and restricts correction to one executable edge rather than replanning the full trajectory.

2.3 Multi-Turn Agent RL and Process Supervision

Recent work applies online reinforcement learning to web and tool agents (Qi et al., 2025; Wei et al., 2025; Wang et al., 2025d;b). ToolRL, AgentPRM, and ToolPRMBench study fine-grained rewards and intermediate action evaluation (Qian et al., 2026; Choudhury, 2025; Xi et al., 2026; Li et al., 2026), while data-centric methods generate, filter, or repair multi-turn trajectories (Prabhakar et al., 2026; Liu et al., 2025; Chen et al., 2026; Xu et al., 2026). EdgeGuard-Agent derives process supervision from explicit, task-grounded edge constraints rather than a general learned quality score. It also re-executes repaired actions in the environment, ensuring that subsequent observations remain causally consistent with the corrected trajectory.

3 Method

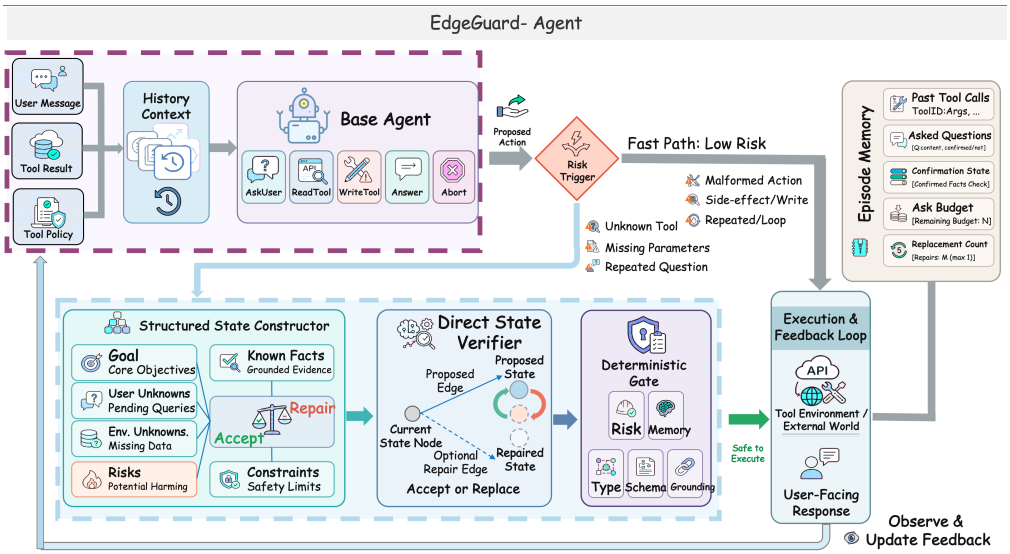

Figure 1: Overview of EdgeGuard-Agent. At inference time, EdgeGuard-Agent selectively verifies risky or abnormal actions and either accepts, repairs, or blocks the proposed transition. During training, the same edge auditor is used to identify invalid transitions, repair trajectories, and construct supervision signals for SFT and GRPO.

3.1 Problem Formulation and Framework Overview

We consider a stateful tool-use task defined by a user request x, a tool set $\mathcal{T}$, and domain policies $\mathcal{P}$. At turn t, the agent observes the current user message, previous tool outputs, and the interaction history $h_t = (o_1, a_1, \ldots, o_t)$. A base policy proposes the next action $\tilde{a}_t \sim \pi_{\text{base}}(a \mid h_t, \mathcal{T}, \mathcal{P})$.

We use the high-level action space $\mathcal{A} = \{\text{AskUser}, \text{ReadTool}, \text{WriteTool}, \text{Answer}, \text{Abort}\}$. These actions differ by both information source and execution risk: AskUser obtains user-owned information, ReadTool retrieves environment facts, WriteTool changes external state, and Answer/Abort terminate the interaction.

Our key idea is to treat each proposed action as a candidate state-transition edge $\tilde{e}_t : z_t \xrightarrow{\tilde{a}_t} \tilde{z}_{t+1}$. The question is therefore local: *is this edge executable from the current state?* This edge view is the main abstraction in EdgeGuard-Agent. Rather than treating failures only as trajectory-level planning errors, it localizes them to a specific transition that can be verified, repaired, or blocked.

As shown in Figure 1, EdgeGuard-Agent uses the same edge semantics in two settings: selective runtime verification at inference time, and trajectory auditing, repair, and supervision construction during training. Appendix A gives the full algorithmic flow.

3.2 Structured State Reconstruction and Edge Auditing

EdgeGuard-Agent is invoked only when an action needs verification. It reconstructs a compact decision state from *visible evidence*:

$$z_t = (G_t, K_t, U_t^{\text{user}}, U_t^{\text{env}}, R_t, \Gamma_t)\,.$$

Here, G_t is the unfinished goal, K_t the grounded known facts, U_t^{user} the missing information that only the user can provide, U_t^{env} the missing information recoverable through tools, R_t the target and risk-relevant state, and Γ_t the schemas, policies, and execution constraints. The complete schema is given in Appendix B.

Reconstructed facts must be grounded in visible evidence: $K_t \subseteq \text{GroundedFacts}(\mathcal{E}_t), \qquad \mathcal{E}_t = \mathcal{E}_t^{\text{user}} \cup \mathcal{E}_t^{\text{tool}} \cup \mathcal{E}_t^{\text{policy}}$. Thus, model guesses, hidden database fields, or benchmark oracles cannot enter K_t.

A key innovation is the split between U_t^{user} and U_t^{env}. Prior clarification work mainly asks whether the agent should ask; EdgeGuard-Agent also checks *which source owns the missing information*. For example, a replacement preference is user-owned and should be asked, whereas a latest order identifier is environment-owned and should be read. Source-routing rules and edge cases are detailed in Appendix B.

Given $(z_t, \tilde{a}_t)$, EdgeGuard-Agent audits the proposed edge along six dimensions: $q_t = \left(q_t^{\text{legal}}, q_t^{\text{source}}, q_t^{\text{ground}}, q_t^{\text{risk}}, q_t^{\text{progress}}, q_t^{\text{redundant}}\right)$, corresponding to legality, source correctness, grounding, execution risk, state progress, and redundancy. The same six-dimensional audit is shared by runtime decisions and training-time supervision; see Appendix D.

3.3 Runtime Edge Verification and Local Repair

Verifying every action is unnecessary. EdgeGuard-Agent therefore applies *selective* verification via a lightweight trigger $c_t = C\left(\tilde{a}_t, h_t, m_t, \mathcal{T}, \mathcal{P}\right)$, where m_t is episode memory. Verification is triggered for writes and other risky or abnormal proposals, including missing preconditions, malformed or unknown tools, repeated calls, repeated questions, state conflicts, and empty actions. Low-risk first-time reads or valid source-correct questions may take a fast path. The trigger taxonomy is summarized in Appendix C.

When triggered, the verifier returns $V(z_t, \tilde{a}_t, c_t) = (d_t, \rho_t, r_t, \hat{a}_t)$, where $d_t \in \{\text{accept}, \text{replace}, \text{block}\}$, ρ_t is confidence, r_t is a brief rationale, and $\hat{a}_t$ is at most one repair action. This *one-edge repair* restriction is deliberate: EdgeGuard-Agent corrects only the current transition rather than replanning the trajectory, preserving valid partial decisions whenever possible.

Typical repairs include asking for missing user information, reading a missing environment fact, requesting confirmation before a write, correcting an invalid tool call, or safely ending an infeasible task.

Any accepted replacement must pass a deterministic execution gate: $\text{Valid}(\hat{a}_t) = g_{\text{schema}} \wedge g_{\text{ground}} \wedge g_{\text{risk}} \wedge g_{\text{memory}}$. The gate enforces tool existence, schema compliance, evidence support for arguments, write preconditions and confirmation, and consistency with episode history. Gate rules and memory structure are given in Appendix E and Appendix F. We also impose $\text{Write}(\hat{a}_t) \Rightarrow \text{Write}(\tilde{a}_t)$, so the verifier cannot escalate a read, question, or answer into a write. If a risky write cannot be verified or safely repaired, EdgeGuard-Agent follows a fail-closed policy and blocks it. Appendix A gives the full runtime procedure.

3.4 Trajectory Auditing and Causal Repair

The same edge auditor is used to build training data. For each task, the current policy generates K trajectories $\tau^{(k)} = (o_1, a_1, \ldots, o_T), \qquad k = 1, \ldots, K$. EdgeGuard-Agent audits critical transitions and labels each trajectory as *clean*, *repairable*, *unsafe*, or *invalid*. Clean trajectories contain no invalid critical edge; repairable ones contain a locally fixable edge; unsafe ones contain unsupported or high-risk writes; invalid ones are discarded when reliable auditing is impossible. Appendix A gives the full procedure.

For a repairable trajectory, EdgeGuard-Agent identifies the first invalid edge $e_t = (z_t, a_t, z_{t+1})$ and produces a verified repair $\hat{a}_t = \mathrm{Repair}(z_t, a_t)$. Crucially, the repair is *re-executed* in the environment: $\hat{z}_{t+1} = \mathcal{F}(z_t, \hat{a}_t)$, and the remaining suffix is regenerated from $\hat{z}_{t+1}$.

This causal re-execution is essential. Editing the action only in text would leave later user messages and tool outputs conditioned on the original, incorrect action. EdgeGuard-Agent instead regenerates the suffix so that subsequent observations remain causally consistent with the repaired state. This distinguishes our method from simple trajectory editing or filtering.

3.5 EdgeGuard-Agent-Guided SFT and GRPO

The supervised training set combines $\mathcal{D}_{\mathrm{SFT}} = \mathcal{D}_{\mathrm{clean}} \cup \mathcal{D}_{\mathrm{repair}} \cup \mathcal{D}_{\text{safe-end}}$. These are verified successful trajectories, causally repaired trajectories, and correct safe-ending trajectories. We first optimize the standard action-level likelihood objective $\mathcal{L}_{\mathrm{SFT}} = -\sum_{\tau \in \mathcal{D}_{\mathrm{SFT}}} \sum_t \log \pi_\theta(a_t \mid h_t)$. Loss is applied only to agent actions. The implemented materialization of trajectory-derived edge records is described in Appendix M.

After SFT, we perform online GRPO with both task-level and edge-level rewards:

$$R(\tau) = R_{\mathrm{task}} + \alpha R_{\mathrm{legal}} + \beta R_{\mathrm{source}} + \gamma R_{\mathrm{ground}} + \eta R_{\mathrm{progress}} - \lambda C_{\mathrm{risk}} - \mu C_{\mathrm{redundancy}}.$$

The final task reward preserves the completion objective, while the edge terms reward legal, source-correct, grounded, and task-advancing actions, and penalize unsafe or redundant ones. This is the training-time analogue of runtime edge verification. Reward design and optimization details are given in Appendix N.

For the K rollouts sampled from the same task, the relative advantage is $A^{(k)} = \frac{R(\tau^{(k)}) - \mu_R}{\sigma_R + \epsilon}$, where μ_R and σ_R are the group mean and standard deviation, and the policy is updated with the GRPO objective (Shao et al., 2024).

After training, the policy can be deployed alone, with no runtime verification overhead, or combined with runtime EdgeGuard-Agent in higher-risk settings. This dual use is a final benefit of the shared edge formulation: the same edge semantics support both explicit execution-time protection and policy internalization through learning.

Table 1: Comparison with inference-time methods on CAR-Bench. Block A reports success, repeated-run stability, write safety, and cost efficiency. Block B reports policy compliance, ending quality, and representative task categories. Best results are bold and second-best results are underlined.

Block A: Overall Performance, Stability, and Safety

Method	TSR↑	Pass^3 ↑	Pass@3↑	Premature↓	Wrong Write↓	RBW↑	SCR↑
No Method	0.229	0.168	0.312	0.699	0.669	0.521	1.700
Ask-when-Needed	0.208	0.136	0.288	0.666	0.648	0.523	1.451
ERASER	0.277	0.176	0.368	0.591	0.576	0.600	1.373
Self-Refine	0.301	0.160	0.440	0.633	0.702	0.593	0.874
Reflexion	0.339	0.224	0.472	0.551	0.586	0.669	1.516
GuardAgent	0.384	0.232	0.544	0.600	0.545	0.581	1.680
SAGE	<u>0.515</u>	<u>0.328</u>	**0.696**	<u>0.386</u>	<u>0.453</u>	<u>0.791</u>	<u>2.668</u>
EdgeGuard-Agent	**0.525**	**0.384**	<u>0.656</u>	**0.367**	**0.434**	**0.796**	**3.615**

Block B: Policy, Ending Quality, and Task Types

Method	Policy↑	User End↑	Hall TSR↑	Hall P^3 ↑	Base P^3 ↑	Disambig TSR↑	Disambig P^3 ↑
No Method	0.670	0.594	0.107	0.060	0.260	0.253	0.200
Ask-when-Needed	0.612	0.601	0.073	0.000	0.260	0.213	0.160
ERASER	0.676	0.707	0.233	0.100	0.280	0.227	0.120
Self-Refine	0.628	0.713	0.253	0.140	0.220	0.307	0.080
Reflexion	0.701	0.672	0.200	0.040	0.340	0.427	**0.360**
GuardAgent	0.758	0.707	0.293	0.180	0.300	0.387	0.200
SAGE	<u>0.852</u>	<u>0.775</u>	<u>0.433</u>	<u>0.260</u>	<u>0.380</u>	**0.480**	**0.360**
EdgeGuard-Agent	**0.862**	**0.836**	**0.540**	**0.380**	**0.400**	<u>0.467</u>	**0.360**

4 EXPERIMENTS

4.1 EXPERIMENTAL SETUP

Benchmarks. We evaluate EdgeGuard-Agent on three complementary stateful tool-use benchmarks. CAR-Bench is our primary testbed and covers Base, Disambiguation, Hallucination, Missing Tool, Missing Tool Parameter, and Missing Tool Response scenarios (Kirmayr et al., 2026). STATE-Bench evaluates long-horizon enterprise workflows with persistent environment states (Microsoft, 2026). We further construct τ^3-Bench-Ambiguous from the τ-bench family (Yao et al., 2024; Barres et al., 2025) by hiding task-critical fields and rewriting explicit requests into ambiguous ones. The resulting tasks require agents to recover user-side information through clarification, query environment-side facts through tools, and execute write operations only after the required conditions have been satisfied.

Model Configurations. We use two agent backbones for the runtime and policy-learning experiments. For the runtime evaluation reported in Tables 1–3 and Figure 2, the base agent is Qwen3-Coder-30B-A3B-Instruct. Qwen3.6-Plus is used as the user simulator and environment model throughout these experiments. For the policy-learning experiments reported in Table 4, including the untrained baseline, EdgeGuard-Agent-curated SFT, and EdgeGuard-Agent-guided GRPO, the base agent is Qwen3.5-27B, while Qwen3.6-Plus remains the user simulator and environment model. Therefore, the results in Table 4 should be interpreted as a separate within-backbone comparison and should not be directly compared in absolute value with the runtime results in Tables 1–3. Each CAR-Bench task is evaluated over three independent runs, whereas STATE-Bench uses five runs per task.

Table 2: Vanilla and Runtime EdgeGuard-Agent across three benchmarks. Δ is the absolute change introduced by EdgeGuard-Agent. Repeated Success denotes Pass^3 on τ^3-Bench-Ambiguous and CAR-Bench, and Pass^5 on STATE-Bench.

Metric	τ^3-Bench-Ambiguous			CAR-Bench			STATE-Bench		
	Vanilla	EdgeGuard-Agent	Δ	Vanilla	EdgeGuard-Agent	Δ	Vanilla	EdgeGuard-Agent	Δ
Core success									
Task Success Rate (%) ↑	26.98	**37.41**	**+10.43**	22.90	**52.50**	**+29.60**	35.60	**43.73**	**+8.13**
Repeated Success (%) ↑	16.19	**21.58**	**+5.40**	16.80	**38.40**	**+21.60**	14.67	**21.33**	**+6.67**
Pass@2 (%) ↑	34.53	**48.20**	**+13.67**	28.80	**64.00**	**+35.20**	—	—	—
Pass@3 (%) ↑	39.57	**53.60**	**+14.03**	31.20	**65.60**	**+34.40**	—	—	—
Information acquisition									
Ask Precision (%) ↑	**56.97**	53.43	-3.54	20.40	**26.10**	**+5.70**	**51.66**	49.96	-1.70
Ask Recall (%) ↑	59.54	**72.41**	**+12.87**	50.70	**61.90**	**+11.20**	24.31	**30.20**	**+5.88**
Over-asking Rate (%) ↓	**43.03**	46.57	+3.54	79.60	**73.90**	**-5.70**	**48.34**	50.04	+1.70
Information Gain / Turn ↑	**0.041**	0.038	-0.003	0.527	**0.674**	**+0.147**	0.0390	0.0392	+0.0001
Safety and state transitions									
Premature Tool-call Rate (%) ↓	6.52	**3.39**	**-3.14**	69.90	**36.70**	**-33.20**	40.76	**20.89**	**-19.86**
Wrong Write Rate (%) ↓	87.71	**77.58**	**-10.13**	66.90	**43.40**	**-23.50**	80.63	**64.03**	**-16.60**
Read-before-Write Rate (%) ↑	99.84	**100.00**	**+0.16**	52.10	**79.60**	**+27.50**	98.84	**99.59**	**+0.75**
State Gain / Action ↑	0.067	**0.134**	**+0.067**	0.312	**0.490**	**+0.178**	-0.126	**-0.004**	**+0.123**
Cost and burden									
User Burden ↓	**3.182**	3.972	+0.790	0.992	**0.938**	**-0.054**	1.528	**1.497**	**-0.031**
Success–Cost Ratio ↑	0.513	**0.666**	**+0.153**	1.700	**3.615**	**+1.915**	0.874	**1.444**	**+0.570**

Baselines and Metrics. We compare against Ask-when-Needed (Wang et al., 2025c), ERASER, Self-Refine (Madaan et al., 2023), Reflexion (Shinn et al., 2023), GuardAgent (Xiang et al., 2024), and SAGE (Suri et al., 2026), under the same backbone, tools, policies, simulator, environment, and evaluation protocol within each setting. We report Task Success Rate (TSR), repeated-run success (Pass^3/Pass^5), Pass@3, Ask Precision/Recall, Premature Tool-call Rate, Wrong Write Rate, Read-before-Write (RBW), State Gain, User Burden, and Success–Cost Ratio (SCR). Full baseline and metric details are provided in Appendix H and Appendix J.

We study four questions: runtime reliability, cross-benchmark and cross-model generalization, component contributions, and the usefulness of EdgeGuard-Agent supervision for SFT and GRPO.

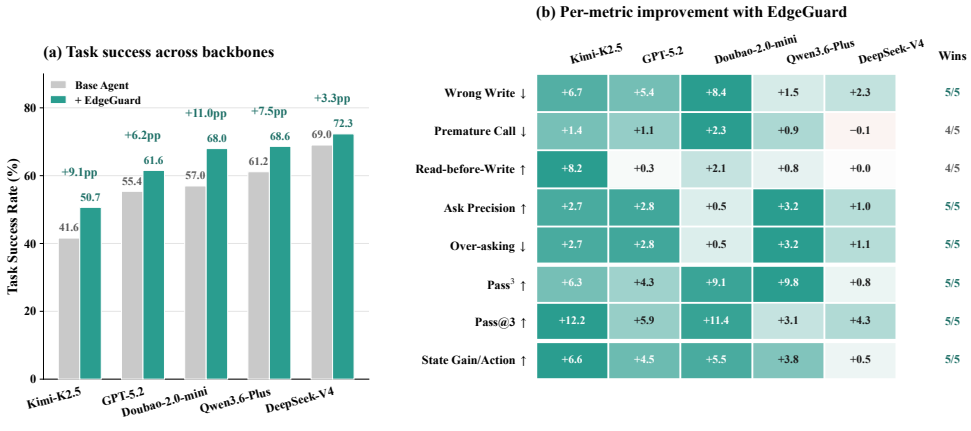

Figure 2: Cross-model generalization on CAR-Bench. Each row compares the vanilla base agent with EdgeGuard-Agent. EdgeGuard-Agent consistently improves task success, state progress, and cost efficiency while reducing wrong writes. † denotes a partial run.

Table 3: Component ablations on CAR-Bench. Ask R denotes Ask Recall, RBW denotes Read-before-Write, and State Gain is measured per action. Higher is better unless otherwise indicated.

Variant	Core Success			Safety and Interaction				Progress and Efficiency		Outcome Quality	
	TSR	Pass^3	Pass@3	Ask R	Premature↓	Wrong Write↓	RBW	State Gain	Success–Cost	Policy	User End
EdgeGuard-Agent	**52.50**	**38.40**	**65.60**	**61.90**	36.70	**43.40**	79.60	**0.490**	**3.615**	**86.20**	**83.60**
No Method	22.90	16.80	31.20	50.70	69.90	66.90	52.10	0.312	1.700	67.00	59.40
w/o Structured State	46.04	32.74	60.18	42.91	11.93	75.34	85.57	0.188	1.861	78.97	78.48
w/o Source Split	45.05	31.42	59.29	38.87	13.17	76.12	84.87	0.173	1.838	76.58	78.17
Write-Only Guard	39.76	25.22	54.42	40.23	14.48	78.23	83.99	0.132	1.469	73.07	71.55
w/o Local Repair	33.38	21.24	46.90	22.92	12.09	77.35	84.61	0.118	1.418	77.78	59.21
Full Replanning	47.68	34.51	60.62	35.03	14.65	72.62	**87.81**	0.149	1.888	79.67	78.62
w/o Deterministic Gate	41.61	27.88	55.31	44.71	15.66	75.93	86.45	0.132	1.543	80.61	73.28
w/o Episode Memory	43.26	28.32	61.06	41.98	**11.39**	75.70	84.92	0.163	1.651	76.58	74.65
w/o Fail-Closed	43.60	28.32	60.62	39.82	13.20	75.92	84.56	0.150	1.762	74.07	78.90
Allow Write Escalation	46.12	29.20	65.04	40.08	14.55	72.00	85.12	0.158	1.834	77.34	79.18

Table 4: EdgeGuard-Agent-guided policy learning on CAR-Bench. Parentheses report absolute changes relative to the untrained baseline. Rep. Succ. denotes Pass^3.

Metric	Baseline	SFT	GRPO
Core			
Succ. Rate (%) ↑	27.69	35.30 (+7.61)	**40.42 (+12.73)**
Rep. Succ. (%) ↑	21.65	23.62 (+1.97)	**29.13 (+7.48)**
Pass@3 (%) ↑	33.86	49.21 (+15.35)	**53.54 (+19.68)**
Information and Safety			
Ask Recall (%) ↑	51.72	53.44 (+1.72)	**59.74 (+8.02)**
Premature (%) ↓	11.52	**10.19 (-1.33)**	11.03 (-0.49)
Wrong Write (%) ↓	72.88	**68.14 (-4.74)**	71.57 (-1.31)
Read→Write (%) ↑	68.43	78.65 (+10.22)	**79.82 (+11.39)**
State and Cost			
State Gain ↑	0.1476	**0.2101 (+0.0625)**	0.1944 (+0.0468)
Succ.–Cost ↑	1.2444	**1.6603 (+0.4159)**	1.6445 (+0.4001)
Burden ↓	1.5814	**1.4383 (-0.1431)**	1.5131 (-0.0683)

4.2 Runtime Comparison on CAR-Bench

Table 1 compares EdgeGuard-Agent with representative clarification, reflection, and runtime-guard methods. EdgeGuard-Agent obtains the highest TSR, Pass^3, Policy, User End, RBW, and SCR, while producing the lowest Premature and Wrong Write rates. Its TSR is statistically comparable to SAGE, but EdgeGuard-Agent is more stable and safer across repeated runs. The largest task-level advantage appears on Hallucination cases, where grounding and execution checks directly prevent unsupported actions. SAGE remains slightly better on Disambiguation TSR, consistent with its focus on clarification.

4.3 Cross-Benchmark and Cross-Model Generalization

Table 2 shows consistent gains across all three benchmarks. EdgeGuard-Agent improves TSR by 10.43, 29.60, and 8.13 percentage points on τ^3-Bench-Ambiguous, CAR-Bench, and STATE-Bench, respectively. More importantly, it reduces both premature execution and wrong writes in every environment. The small losses in Ask Precision on τ^3-Bench-Ambiguous and STATE-Bench indicate a more conservative clarification policy, but this trade-off is accompanied by higher Ask Recall and substantially safer state transitions.

Figure 2 evaluates five model families on CAR-Bench. EdgeGuard-Agent improves TSR by 3.3–11.0 points, lowers Wrong Write by 1.5–8.4 points, and improves Pass^3, Pass@3, and State Gain/Action for every model. The gains are not restricted to weaker agents: GPT-5.2 and Qwen3.6+ also improve, indicating that EdgeGuard-Agent supplies task-state constraints that remain useful after base-model capability increases.

4.4 Component Ablations

Table 3 confirms that the gains do not come from conservative blocking alone. Removing structured state or the user–environment source split reduces success and state progress. A write-only guard cannot repair earlier information-acquisition errors, while removing local repair causes the largest drop in task completion. Full replanning achieves high RBW but lower State Gain and SCR, supporting the use of local rather than global correction. Deterministic gating, episode memory, fail-closed behavior, and the no-write-escalation rule each contribute to reliable execution.

Several ablations show lower Premature rates than the full method. This is caused by excessive blocking: these variants avoid acting, but also fail to advance the task, as reflected by lower TSR and State Gain.

4.5 EdgeGuard-Agent-Guided Policy Learning

Table 4 evaluates whether EdgeGuard-Agent can provide useful training supervision in addition to runtime protection. EdgeGuard-Agent-curated SFT improves success from 27.69% to 35.30%, while EdgeGuard-Agent-guided GRPO reaches 40.42%. GRPO produces the strongest TSR, repeated success, Pass@3, Ask Recall, and RBW. SFT is better on Wrong Write, State Gain, User Burden, and SCR, suggesting that supervised learning absorbs conservative safe behavior more directly, while online RL improves exploration and final task completion. This division is consistent with recent work on fine-grained tool rewards and process supervision (Qian et al., 2026; Wang et al., 2025d; Xi et al., 2026; Li et al., 2026).

Overall, the runtime and training results support the same conclusion: explicit state-edge supervision improves immediate execution and also provides a useful learning signal. Runtime EdgeGuard-Agent offers direct fail-closed protection, whereas SFT and GRPO partially internalize the same decision boundaries into the base policy.

5 DISCUSSION

5.1 GUARDING AND LEARNING THROUGH SHARED EDGE SEMANTICS

EdgeGuard-Agent uses the same state-edge semantics for two complementary purposes. At inference time, it verifies whether a proposed action is executable and, when necessary, replaces it with one locally valid repair. At training time, it identifies the first invalid transition and converts the corresponding audit into causally consistent supervision and process rewards. This shared formulation is useful because many tool-use failures are local: an action may query the wrong source, use an unsupported argument, skip a precondition, reuse stale confirmation, or repeat an operation without advancing the task. Verifying the current edge preserves valid decisions made by the base agent while providing finer credit assignment than trajectory-level success alone.

Runtime verification and policy learning serve different deployment needs. Runtime EdgeGuard-Agent provides explicit fail-closed protection for high-risk operations, whereas EdgeGuard-Agent-curated SFT and EdgeGuard-Agent-guided GRPO partially internalize the same evidence, source, and safety constraints into the policy. The trained policy can therefore be deployed independently for lower overhead or combined with runtime EdgeGuard-Agent when explicit execution-time protection is required.

5.2 LIMITATIONS

EdgeGuard-Agent relies on sufficiently complete tool schemas, policies, and observable environment feedback. Errors in state reconstruction may still lead to incorrect audits, and deterministic gating reduces practical risk rather than providing a formal safety guarantee. Causal re-execution increases training cost, while reward design influences the trade-off between task completion and conservative behavior. Empirically, higher Ask Recall can reduce Ask Precision, and GRPO does not uniformly outperform SFT on write safety, state efficiency, or user burden. Finally, our evaluation focuses on business-oriented tool environments; broader validation on web, code, and embodied agents remains future work.

6 CONCLUSION

We introduced EdgeGuard-Agent, a unified framework that treats state-transition edges as the shared unit for runtime verification and policy learning under incomplete information. EdgeGuard-Agent selectively verifies and locally repairs risky actions at inference time, while using the same audit semantics to construct causally consistent SFT data and GRPO rewards.

Across CAR-Bench, STATE-Bench, and τ^3-Bench-Ambiguous, EdgeGuard-Agent improves task success, repeated-run reliability, and execution safety. These results show that local edge auditing provides a practical basis for runtime protection, trajectory repair, and multi-turn agent learning.

AI USE STATEMENT

Generative AI tools were used to assist with manuscript preparation and formatting. The use of language models as agents, user simulators, environment models, verifiers, and evaluators is described in the main text and appendices.

REPRODUCIBILITY STATEMENT

Appendices A–F describe the algorithms, structured state, trigger rules, verifier, deterministic gate, and episode memory. Appendices G–J document benchmark construction, baseline implementations, inference cost, and evaluation metrics. Appendices M–P describe training, rewards, case studies, and additional results, including the boundaries of the retained evidence.

REFERENCES

Victor Barres, Honghua Dong, Soham Ray, Xujie Si, and Karthik Narasimhan. τ^2-Bench: Evaluating Conversational Agents in a Dual-Control Environment. *arXiv preprint arXiv:2506.07982*, 2025.

Jiaao Chen, Jingyuan Qi, Mingye Gao, Wei-Chen Wang, Hanrui Wang, and Di Jin. Eigendata: A self-evolving multi-agent platform for function-calling data synthesis, auditing, and repair. *arXiv preprint arXiv:2603.05553*, 2026.

Zhaorun Chen, Mintong Kang, and Bo Li. Shieldagent: Shielding agents via verifiable safety policy reasoning. *arXiv preprint arXiv:2503.22738*, 2025.

Sanjiban Choudhury. Process reward models for llm agents: Practical framework and directions. *arXiv preprint arXiv:2502.10325*, 2025.

Alejandro Cuadron, Pengfei Yu, Yang Liu, and Arpit Gupta. Saber: Small actions, big errors–safeguarding mutating steps in llm agents. *arXiv preprint arXiv:2512.07850*, 2025.

Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Nan Duan, and Weizhu Chen. Critic: Large language models can self-correct with tool-interactive critiquing, 2024. *URL https://arxiv. org/abs/2305.11738*, 2024.

Xiang Huang, Sitao Cheng, Shanshan Huang, Jiayu Shen, Yong Xu, Chaoyun Zhang, and Yuzhong Qu. Queryagent: A reliable and efficient reasoning framework with environmental feedback based self-correction. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 5014–5035, 2024.

Johannes Kirmayr, Lukas Stappen, and Elisabeth André. Car-bench: Evaluating the consistency and limit-awareness of llm agents under real-world uncertainty. *arXiv preprint arXiv:2601.22027*, 2026.

Shirley Kokane, Ming Zhu, Tulika Awalgaonkar, Jianguo Zhang, Thai Hoang, Akshara Prabhakar, Zuxin Liu, Tian Lan, Liangwei Yang, Juntao Tan, et al. Spectool: A benchmark for characterizing errors in tool-use llms. *Preprint*, 2024.

Dawei Li, Yuguang Yao, Zhen Tan, Huan Liu, and Ruocheng Guo. Toolprmbench: Evaluating and advancing process reward models for tool-using agents. *arXiv preprint arXiv:2601.12294*, 2026.

Weiwen Liu, Xu Huang, Xingshan Zeng, Shuai Yu, Dexun Li, Shuai Wang, Weinan Gan, Zhengying Liu, Yuanqing Yu, Zezhong WANG, et al. Toolace: Winning the points of llm function calling. In *International conference on learning representations*, volume 2025, pp. 41359–41381, 2025.

Jiarui Lu, Thomas Holleis, Yizhe Zhang, Bernhard Aumayer, Feng Nan, Haoping Bai, Shuang Ma, Shen Ma, Mengyu Li, Guoli Yin, et al. Toolsandbox: A stateful, conversational, interactive evaluation benchmark for llm tool use capabilities. In *Findings of the Association for Computational Linguistics: NAACL 2025*, pp. 1160–1183, 2025.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, et al. Self-refine: Iterative refinement with self-feedback. *Advances in neural information processing systems*, 36:46534–46594, 2023.

Microsoft. STATE-Bench: Benchmark for Enterprise Workflows. `https://github.com/microsoft/STATE-Bench`, 2026. GitHub repository, accessed 2026-07-24.

Yutao Mou, Zhangchi Xue, Lijun Li, Peiyang Liu, Shikun Zhang, Wei Ye, and Jing Shao. Toolsafe: Enhancing tool invocation safety of llm-based agents via proactive step-level guardrail and feedback. *arXiv preprint arXiv:2601.10156*, 2026.

Akshara Prabhakar, Zuxin Liu, Ming Zhu, Jianguo Zhang, Tulika Manoj Awalgaonkar, Shiyu Wang, Zhiwei Liu, Haolin Chen, Thai Hoang, Juan Carlos Niebles, et al. Apigen-mt: Agentic pipeline for multi-turn data generation via simulated agent-human interplay. *Advances in Neural Information Processing Systems*, 38, 2026.

Zehan Qi, Xiao Liu, Iat Long Iong, Hanyu Lai, Xueqiao Sun, Jiadai Sun, Xinyue Yang, Yu Yang, Shuntian Yao, Wei Xu, et al. Webrl: Training llm web agents via self-evolving online curriculum reinforcement learning. In *International Conference on Learning Representations*, volume 2025, pp. 79791–79821, 2025.

Cheng Qian, Emre Can Acikgoz, Qi He, Hongru Wang, Xiusi Chen, Dilek Hakkani-Tur, Gokhan Tur, and Heng Ji. Toolrl: Reward is all tool learning needs. *Advances in Neural Information Processing Systems*, 38:105523–105553, 2026.

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

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

Manan Suri, Puneet Mathur, Nedim Lipka, Franck Dernoncourt, Ryan A Rossi, and Dinesh Manocha. Structured uncertainty guided clarification for llm agents. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 40811–40838, 2026.

Harsh Trivedi, Tushar Khot, Mareike Hartmann, Ruskin Manku, Vinty Dong, Edward Li, Shashank Gupta, Ashish Sabharwal, and Niranjan Balasubramanian. Appworld: A controllable world of apps and people for benchmarking interactive coding agents. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 16022–16076, 2024.

Haoyu Wang, Christopher M Poskitt, and Jun Sun. Agentspec: Customizable runtime enforcement for safe and reliable llm agents. *arXiv preprint arXiv:2503.18666*, 2025a.

Hongru Wang, Cheng Qian, Wanjun Zhong, Xiusi Chen, Jiahao Qiu, Shijue Huang, Bowen Jin, Mengdi Wang, Kam-Fai Wong, and Heng Ji. Otc: Optimal tool calls via reinforcement learning. *arXiv e-prints*, pp. arXiv–2504, 2025b.

Wenxuan Wang, Shi Juluan, Zixuan Ling, Yuk-Kit Chan, Chaozheng Wang, Cheryl Lee, Youliang Yuan, Jen-tse Huang, Wenxiang Jiao, and Michael R Lyu. Learning to ask: When llm agents meet unclear instruction. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 21784–21795, 2025c.

Xingyao Wang, Zihan Wang, Jiateng Liu, Yangyi Chen, Lifan Yuan, Hao Peng, and Heng Ji. Mint: Evaluating llms in multi-turn interaction with tools and language feedback. In *International Conference on Learning Representations*, volume 2024, pp. 32593–32627, 2024.

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

Zhepei Wei, Wenlin Yao, Yao Liu, Weizhi Zhang, Qin Lu, Liang Qiu, Changlong Yu, Puyang Xu, Chao Zhang, Bing Yin, et al. Webagent-r1: Training web agents via end-to-end multi-turn reinforcement learning. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 7920–7939, 2025.

Zhiheng Xi, Chenyang Liao, Guanyu Li, Zhihao Zhang, Wenxiang Chen, Binghai Wang, Senjie Jin, Yuhao Zhou, Jian Guan, Wei Wu, et al. Agentprm: Process reward models for llm agents via step-wise promise and progress. In *Proceedings of the ACM Web Conference 2026*, pp. 4184–4195, 2026.

Zhen Xiang, Linzhi Zheng, Yanjie Li, Junyuan Hong, Qinbin Li, Han Xie, Jiawei Zhang, Zidi Xiong, Chulin Xie, Carl Yang, et al. Guardagent: Safeguard llm agents by a guard agent via knowledge-enabled reasoning. *arXiv preprint arXiv:2406.09187*, 2024.

Tianshi Xu, Yuteng Chen, and Meng Li. Cleaner: Self-purified trajectories boost agentic reinforcement learning. *arXiv preprint arXiv:2601.15141*, 2026.

Seungbin Yang, ChaeHun Park, Taehee Kim, and Jaegul Choo. Can tool-augmented large language models be aware of incomplete conditions? *arXiv preprint arXiv:2406.12307*, 2024.

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