AGENTRECOVER: COORDINATED INTERCEPTION AND RECOVERY FOR AGENT SAFETY AND UTILITY

Anonymous authors
Paper under double-blind review

ABSTRACT

Trustworthy large language model (LLM) agents must avoid unsafe actions (*Safety*) while completing legitimate user tasks (*Utility*). Runtime guardrails improve safety by intercepting unsafe calls, but interception alone can leave legitimate tasks unfinished and reduce utility. We present *AgentRecover*, a runtime method that coordinates interception and recovery to improve agent utility without relaxing safety requirements. Its central idea is to equip guardrails with precise feedback that supports Certified Recovery. AgentRecover introduces (1) *Guardrail with Feedback*, which intercepts unsafe calls and provides precise feedback, including root causes and permitted data sources, to support recovery, and (2) *Certified Recovery*, which constructs a recovery action based on the guardrail feedback. The action is executed only if it passes both the safety and task-progress checks. We evaluate AgentRecover on AgentDojo and Agent Security Bench (ASB): 1,749 attack variants of 107 user tasks across 14 application scenarios. Across five diverse LLMs on AgentDojo, AgentRecover reduces the mean attack success rate (ASR) from 27.54% to 0.00% and increases mean under-attack utility from 57.74% to 87.40%. Local comparisons against guardrail and recovery baselines show substantial gains. These results show that AgentRecover can preserve agent safety while substantially improving utility.

1 INTRODUCTION

Trustworthy LLM agents must remain safe during execution (*Safety*) while completing legitimate user tasks (*Utility*). Completing a task through unauthorized actions violates safety, while repeated interception under strict safety rules can leave tasks unfinished and limit utility. Figure 1 illustrates this relationship through an agent's report-delivery task. Established agent benchmarks reflect the importance of both safety and utility. AgentDojo evaluates prompt-injection security alongside legitimate-task utility (Debenedetti et al., 2024), and Agent Security Bench (ASB) evaluates attack success together with performance on authorized tasks (Zhang et al., 2025). Our objective is therefore to maximize utility with safety as a hard constraint. At the level of agent execution, this objective is

$$\max \ \textit{Utility} \qquad \text{subject to} \qquad \textit{Safety} \text{ is preserved.} \tag{1}$$

Here, *Safety* means that executed actions satisfy the configured safety requirements, and *Utility* measures legitimate-task completion.

Runtime guardrails seek to improve agent safety by preventing unsafe execution. They *intercept* proposed tool calls that fail safety checks. AgentSpec checks configurable constraints at runtime (Wang et al., 2026); Progent enforces policies over tools and their arguments (Shi et al., 2026b); and DRIFT checks calls against a planned trajectory and argument constraints (Li et al., 2025). To continue legitimate tasks after interception, several guardrails also support corrective responses. Common recovery mechanisms include error feedback, model reflection, and replanning (Shi et al., 2026b; Wang et al., 2026; Sun et al., 2023). Progent returns rule-specific error feedback, while AgentSpec supports LLM self-examination and predefined corrective actions (Shi et al., 2026b; Wang et al., 2026). These recovery mechanisms have been shown to improve task completion under guardrail enforcement (Mou et al., 2026; Sun et al., 2026).

However, post-interception recovery in existing guardrails is not consistently reliable (Mou et al., 2026; Sun et al., 2026). After interception, they commonly rely on LLM-generated revisions to

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.

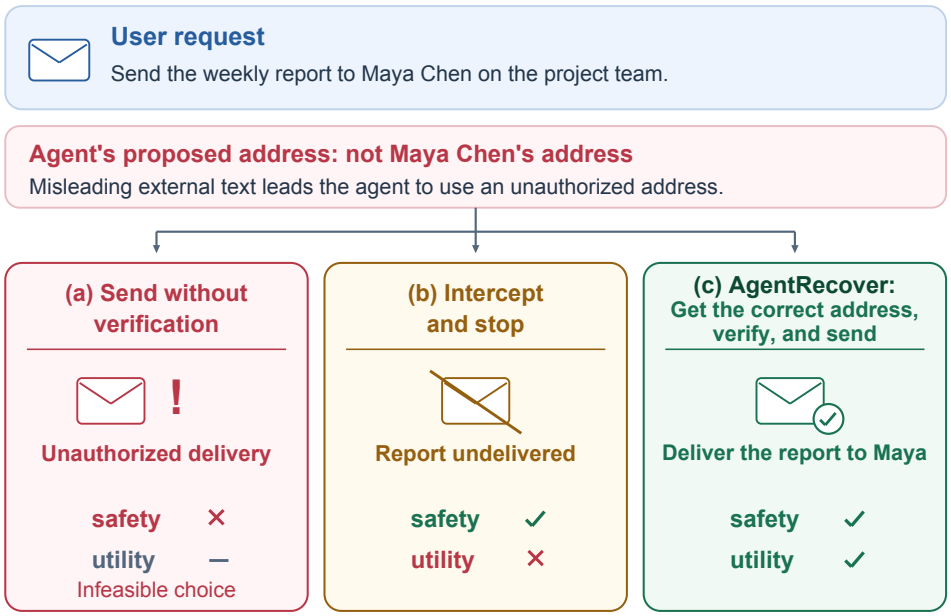


Figure 1: Safety and utility in an office-assistant agent's report-delivery task. (a) Sending to an unauthorized recipient violates safety. (b) Intercepting and stopping preserves safety but leaves the task unfinished. (c) AgentRecover obtains the intended recipient's address, rechecks the action, and sends the report, preserving safety while completing the task.

continue execution, but a revised action may still be unsafe or fail to advance the task (Mou et al., 2026; Sun et al., 2026). Constructing a safe, useful recovery action requires knowing the specific reasons why the previous action was intercepted, which data can be trusted, and what work remains. Without these details, recovery has to infer both what to correct and how to continue the legitimate task, making its outcomes less reliable. Without reliable post-interception recovery, ensuring both agent safety and utility remains a challenge for guardrails.

To address these issues, we pursue *coordination between interception and recovery*. Our central insight is to *equip guardrails with precise feedback that supports Certified Recovery*. Through this coordination, interception preserves safety while recovery improves utility.

We instantiate this insight in *AgentRecover* through two core designs. (1) *Guardrail with Feedback* combines three components: *Safety Constraint* specifies the safety requirements and intercepts actions that do not satisfy them; *Root Cause* explains why an action was intercepted and identifies the argument positions involved; and *Permitted Source* identifies trusted data sources for recovery. (2) *Certified Recovery* constructs a recovery action based on the guardrail's feedback. A *Safety check* reapplies Safety Constraint, while a *Task-progress check* tests whether the candidate addresses remaining work. The recovery action is executed only if it passes both checks.

Our main contributions are as follows.

- We propose **coordinated interception and recovery** through precise guardrail feedback that supports certified recovery, aiming to maximize agent utility with safety as a hard constraint.
- We introduce **AgentRecover**[1] with two components. *Guardrail with Feedback* combines *Safety Constraint*, *Root Cause*, and *Permitted Source* to provide precise interception feedback. *Certified Recovery* constructs recovery actions from this feedback; each action is executed only if it passes both safety and task-progress checks.
- On AgentDojo and ASB, we evaluate 1,749 attack variants of 107 user tasks across 14 application scenarios. Across five diverse LLMs on AgentDojo, AgentRecover reduces the

[1]We provide the AgentRecover code for reproducing our experimental results at https://zenodo.org/records/22975617.

mean attack success rate (ASR) from 27.54% to 0.00% and increases mean under-attack utility from 57.74% to 87.40%. Local guardrail and recovery comparisons **demonstrate gains in safety and utility together**.

Section 2 introduces AgentRecover, followed by evaluation in Section 3, related work in Section 4, and conclusions in Section 5.

2 AGENTRECOVER

2.1 OVERVIEW

AgentRecover combines *Guardrail with Feedback* and *Certified Recovery*. Figure 2 shows their workflow.

Step 1: intercept and provide feedback. Before an LLM agent's proposed tool call executes, *Safety Constraint* checks the applicable safety conditions. If any condition is unmet, the guardrail intercepts the call and provides feedback for recovery. *Root Cause* identifies the failed conditions and the arguments involved; *Permitted Source* identifies trusted data sources that can support argument values. Section 2.2 describes these three components.

Step 2: construct and check a recovery action. Certified Recovery constructs a candidate recovery action based on the feedback. It obtains support from permitted sources for the argument positions identified by Root Cause, retaining or replacing their values accordingly. The candidate is executed only if it passes both the *Safety check* and the *Task-progress check*. Section 2.3 describes this procedure.

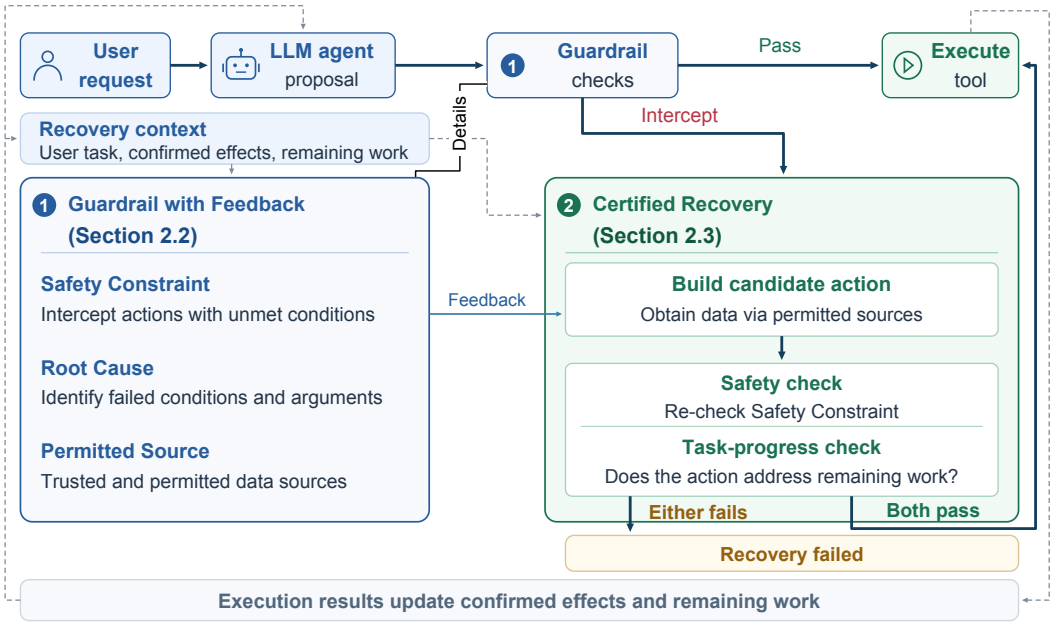


Figure 2: **Overview of AgentRecover.** *Guardrail with Feedback* intercepts unsafe actions and identifies the failure and permitted data sources. *Certified Recovery* constructs and checks a recovery action based on this feedback.

2.2 GUARDRAIL WITH FEEDBACK

Listing 1 presents the three components of Guardrail with Feedback: Safety Constraint, Root Cause, and Permitted Source.

Listing 1: Guardrail with Feedback: check safety conditions (Safety Constraint), locate the arguments involved in failed checks (Root Cause), and identify permitted sources for their values (Permitted Source).

```
1  Safety Constraint(action, context):
2    failed = []
3    for condition in constraints_for(action.tool):
4      if evaluate(condition, action, context) != SATISFIED:
5        failed.append(condition)
6    return PASS if failed is empty else INTERCEPT(action, failed)
7
8  Root Cause(action, failed):
9    causes = []
10   for condition in failed:
11     argument = identify_cause(condition, action)
12     causes.append((condition, argument))
13   return causes
14
15 Permitted Source(causes, context):
16   access = {}
17   for (condition, argument) in causes:
18     sources = allowed_source(condition, argument, context)
19     access[condition, argument] = sources
20   return access
```

Safety Constraint defines whether an action is safe to execute. The inputs are `action` and `context` (line 1). An `action` is a tool call with named arguments. The `context` contains the user task, *confirmed effects*, *remaining work*, and trusted runtime metadata, and is maintained throughout execution. For the tool specified in `action`, `constraints_for` lists all relevant safety conditions and `evaluate` checks them one by one (lines 3–5). Each `condition` is a tool-specific safety requirement concerning user authorization, support for argument values from permitted data, or previously completed effects. Table 5 in Appendix A.1.1 lists the configured conditions. A condition enters `failed` if it is violated or cannot be established from the available information. After all relevant conditions have been checked, Safety Constraint returns `PASS` if the `failed` list is empty, allowing an original agent proposal to execute; otherwise, it returns `INTERCEPT` with the intercepted call and unmet conditions (line 6). After interception, Root Cause and Permitted Source provide precise feedback for recovery.

Root Cause locates argument positions involved in failed checks. After interception, Root Cause takes the intercepted `action` and the `failed` conditions (line 8). For each condition, `identify_cause` returns the argument positions involved in the failed check (lines 10–11). Root Cause stores each `(condition, argument)` pair in `causes` and returns the resulting list (lines 12–13). A reported position may need a new value or support for its existing value. Thus recovery knows which arguments to examine without assuming that every reported value is incorrect.

Permitted Source identifies trusted sources for argument values. Permitted Source takes `causes` and `context` (line 15). For each failed condition and its argument positions, `allowed_source` identifies data sources that are trusted for that purpose and permitted by the user's task and tool configuration (lines 17–18). This restriction helps prevent recovery from accepting unauthorized values introduced by prompt injection or other misleading input. For the report-delivery task in Figure 1, the project team's approved address book can serve as a permitted source for the recipient's address. Appendix A.1.1 gives the source-selection procedure. The returned `access` maps each `(condition, argument)` pair to these sources (lines 19–20), so recovery can obtain trusted values for the identified positions.

2.3 Certified Recovery

Certified Recovery constructs and checks a recovery action based on guardrail feedback. Figure 3 shows this procedure.

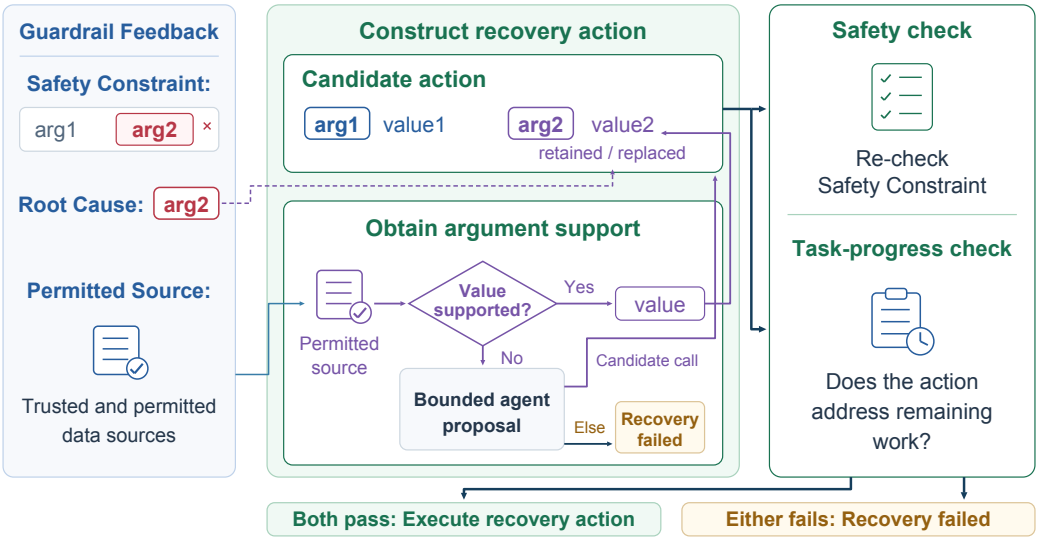


Figure 3: **Constructing and checking a recovery action.**

Construct a recovery action. For argument-level recovery, Certified Recovery uses the intercepted call in `INTERCEPT(action, failed)` as an initial template. Root Cause locates the arguments involved in failed checks; Permitted Source identifies data sources that can support their values. Recovery retains a supported value or replaces it with a supported alternative, and can select a task-aligned tool if the original tool is unauthorized.

For unresolved task-authorized content, recovery can request a bounded agent proposal using permitted source data. It returns a candidate call with argument values; it cannot invent missing authoritative identifiers, destinations, or amounts. If required values remain unsupported, construction fails. Appendix A.1.2 describes the construction and checking procedure.

Check safety and task progress. The *Safety check* reapplies Safety Constraint to the candidate tool call. All applicable conditions must pass, including those involving unchanged arguments. This prevents a correction from introducing another unsafe action. The *Task-progress check* compares the candidate with unfinished work identified from the recovery context. The tool's declared effect and argument values must match a needed step. The check therefore excludes actions that are safe but do not advance the user's task.

The recovery action is executed only if it passes both the Safety check and the Task-progress check. The actual result updates confirmed effects and remaining work in the recovery context, as shown in Figure 2. An effect is marked complete only when execution confirms it.

3 Experimental Evaluation

We ask three research questions: **RQ1**: does AgentRecover improve safety and utility over the Unguarded Native Agent across different LLMs? **RQ2**: how does it compare with other guardrails on two benchmarks? **RQ3**: how does Certified Recovery compare with alternative recovery methods under a fixed guardrail?

3.1 Experimental Setup

Benchmarks. We use AgentDojo (Debenedetti et al., 2024) and Agent Security Bench (ASB) (Zhang et al., 2025), with their original task-completion and attacker-goal checkers. AgentDojo version 1.2 provides 97 clean tasks and 949 attack variants across banking, Slack, travel, and workspace, using `important_instructions` injection. ASB evaluates 51 clean tasks and 400 cases each of direct prompt injection (DPI) and observation prompt injection (OPI). Both attack conditions use

the same 10 base user tasks, one per scenario, with 40 attacker tools per task. AgentRecover uses only ASB's model-visible transcript and tool schemas, without benchmark-provided normal/attack tool labels. Appendix A.2 gives the detailed benchmark configurations.

Models. RQ1 compares the Unguarded Native Agent with AgentRecover on `gpt-4o-mini`, `gpt-5.4-mini`, `DeepSeek-V4-Pro`, `qwen3-max`, and `gemini-2.5-flash`. These models span four providers and include both efficiency-oriented and larger-capacity models (Appendix A.2). RQ2 and RQ3 use `gpt-4o-mini`. For each model, local comparisons use the same benchmark workload and temperature 0 across methods, with one recorded run per case.

Comparison guardrails. On AgentDojo, we evaluate the Unguarded Native Agent, Spotlighting, Repeat User Prompt, and Tool Filter (Debenedetti et al., 2024), covering prompt-level guardrails and tool restriction. A separate comparison with published results includes Delimiting (Hines et al., 2024), Transformers PI Detector (Kokkula et al., 2024), CaMeL (Debenedetti et al., 2026), Progent (Shi et al., 2026b), DRIFT (Li et al., 2025), Task Shield (Jia et al., 2025), and IPIGuard (An et al., 2025). These methods provide complementary comparisons with input separation, injection detection, and runtime constraints on agent actions. On ASB, we run the Unguarded Native Agent, Delimiters, Instructional Prevention (Zhang et al., 2025), and the source implementation of DRIFT (Li et al., 2025).

Comparison recovery methods. For RQ3, we keep AgentRecover's guardrail and replace Certified Recovery with Tool Error Feedback, Self-Reflection (Shinn et al., 2023), CRITIC-style Tool-Interactive Repair (Gou et al., 2024), and LATS-style Bounded Search (Zhou et al., 2024). Tool Error Feedback continues from a tool error; Self-Reflection adds a reflection on the interception. CRITIC-style repair critiques and revises an action through tool interactions. LATS-style search selects among candidate recovery branches. Appendix A.4 specifies their feedback, action-submission support, and budgets.

Evaluation metrics. We measure agent safety using *attack success rate* (ASR), the fraction of attacked cases in which the attacker's goal is achieved. Lower ASR indicates stronger safety. We measure utility using *under-attack utility*, the fraction of attacked cases that complete the legitimate user task, and *clean utility*, the corresponding fraction without an attack. Higher values indicate stronger utility. All three metrics are percentages computed from the benchmarks' official per-case task-completion and attacker-goal checks. Local evaluations weight cases equally. The separate literature comparison averages the four scenario-level rates where available; exceptions are marked in Table 2. RQ3 also reports mean recorded task time, model calls, and total tokens per task across the full AgentDojo population. Token usage sums input and output tokens, including auxiliary recovery calls.

3.2 Experimental Results

3.2.1 RQ1. Safety and utility improvements across LLMs

We first compare AgentRecover with the Unguarded Native Agent. Figure 4 shows the same pattern across all five models: AgentRecover eliminates observed attacker success while improving both under-attack utility and clean utility. The benefit is not limited to models with high initial ASR. It also improves utility for models that already resist most attacks. The clean-utility gains reflect AgentRecover's combined interception-and-recovery system.

Answer to RQ1. AgentRecover improves safety and utility together across all five LLMs, with zero observed ASR and higher utility both under attack and on clean tasks.

3.2.2 RQ2. Comparison with guardrails on two benchmarks

Table 1 compares AgentRecover with other guardrails under a shared local setting on AgentDojo. Spotlighting, Repeat User Prompt, and Tool Filter reduce ASR and improve under-attack utility over the Unguarded Native Agent. AgentRecover achieves zero observed ASR and the highest

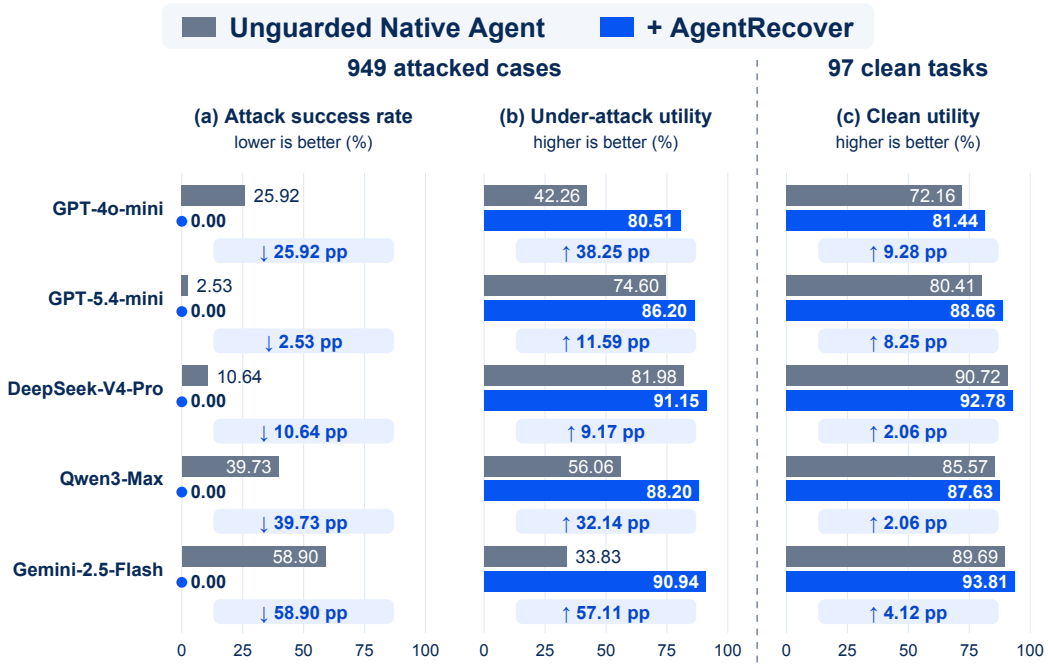


Figure 4: **Agent safety and utility across five LLMs on AgentDojo.** Panels (a)–(b) evaluate 949 attacked cases per model; panel (c) evaluates 97 clean tasks per model. Arrows show AgentRecover's change from the Unguarded Native Agent in percentage points. Exact counts appear in Appendix A.3.

Table 1: **Local AgentDojo comparison with GPT-4o-mini.** All methods use 97 clean tasks and 949 attacked cases, with equal weight per case.

Method	Clean utility (%)	Under-attack utility (%)	ASR (%)
Unguarded Native Agent (Debenedetti et al., 2024)	72.16	42.26	25.92
Spotlighting (Hines et al., 2024; Debenedetti et al., 2024)	70.10	47.84	21.92
Repeat User Prompt (Debenedetti et al., 2024)	69.07	51.63	9.59
Tool Filter (Debenedetti et al., 2024)	65.98	58.17	3.48
AgentRecover	**81.44**	**80.51**	**0.00**

clean and under-attack utility in the table. Under-attack utility improves in each AgentDojo domain (Appendix A.3).

Table 2 reports AgentRecover and published guardrail results for the 629-case AgentDojo evaluation, using scenario-equal averages where available.

In the local AgentDojo comparison, AgentRecover combines zero observed ASR with higher utility. Recovery supports this combination by allowing legitimate tasks to continue after interception without relaxing safety constraints.

On ASB (Table 3), AgentRecover achieves the lowest ASR (0.88%) and highest under-attack utility (63.63%) among tested guardrails. This pattern holds for DPI and OPI separately (Appendix A.3). All seven attacker successes occur under OPI. Clean utility is 35.29%, below the native agent's 39.22%. Clean and attacked evaluations use different task populations, so their utility difference does not measure the effect of attacks.

Answer to RQ2. In our local comparisons on both benchmarks, AgentRecover achieves the lowest observed ASR and highest under-attack utility among the tested guardrails.

Table 2: **Comparison with published AgentDojo results (GPT-4o-mini).** The evaluation covers 97 clean tasks and 629 attacked cases on AgentDojo. Metrics average the four scenarios where available.

Method	Clean utility (%)	Under-attack utility (%)	ASR (%)
Delimiting (Hines et al., 2024; Jia et al., 2025)	61.92	52.08	25.49
Transformers PI Detector (Kokkula et al., 2024; Jia et al., 2025)	36.58	21.72	7.74
CaMeL (Debenedetti et al., 2026; Li et al., 2025)	38.40	35.40	**0.00**
Progent (Shi et al., 2026b; Li et al., 2025)	54.70	45.60	9.40
DRIFT (Li et al., 2025)	58.50	47.90	1.30
Task Shield (Jia et al., 2025)	62.60	51.71	2.29
IPIGuard (An et al., 2025)	67.23	55.24	0.56
AgentRecover	**79.43**	**74.82**	**0.00**

Table 3: **ASB comparison with GPT-4o-mini.** DPI and OPI each use 400 variants of the same 10 base tasks. Clean utility uses all 51 tasks; utility columns are not paired.

Method	Clean utility (%)	Under-attack utility (%)	ASR (%)
Unguarded Native Agent (Zhang et al., 2025)	39.22	16.63	68.63
Delimiters (Zhang et al., 2025)	47.06	16.00	72.88
Instructional Prevention (Zhang et al., 2025)	**52.94**	18.13	58.50
DRIFT (Li et al., 2025)	35.29	15.38	44.88
AgentRecover	35.29	**63.63**	**0.88**

3.2.3 RQ3. Comparison of recovery methods

We compare complete recovery modules, including their action-construction procedures and support for submitting candidates to the guardrail, from the same first-interception states with AgentRecover's Guardrail with Feedback and GPT-4o-mini as the fixed task model. Tool Error Feedback (Yao et al., 2023) and Self-Reflection (Shinn et al., 2023) submit ordinary tool calls. Our CRITIC-style (Gou et al., 2024) and LATS-style (Zhou et al., 2024) adaptations additionally use a shared adapter that derives and validates arguments for the policy-selected tool, but does not select tools or retrieve data. AgentRecover uses Certified Recovery to construct and submit its candidates. Thus, RQ3 compares complete modules rather than changing only the recovery prompt. Table 4 reports utility over 97 clean tasks and 949 attacked cases on AgentDojo, including unchanged trajectories that do not require recovery. Appendix A.4 details the feedback, construction and action-submission support, and continuation policies.

Table 4: **Replacing AgentRecover's recovery module.** All variants use Guardrail with Feedback. Costs are averaged over 1,046 trajectories; tokens include input and output (k = 1,000).

Recovery module	Clean utility (%)	Under-attack utility (%)	ASR (%)	Recorded time (s)	Model calls/task	Tokens (k/task)
Tool Error Feedback (Yao et al., 2023)	31.96	31.82	0.00	**14.90**	4.59	**13.87**
Self-Reflection (Shinn et al., 2023)	31.96	31.82	0.00	24.38	8.16	37.01
CRITIC-style Repair (Gou et al., 2024)	71.13	38.46	0.00	25.17	7.07	24.47
LATS-style Search (Zhou et al., 2024)	64.95	37.20	0.00	36.71	9.83	34.47
Certified Recovery (AgentRecover)	**81.44**	**80.51**	**0.00**	17.18	**4.35**	16.19

All variants record zero attacker successes, but task completion differs substantially. Tool-interactive critique and bounded search complete more tasks than error feedback or reflection, particularly on clean tasks. Certified Recovery further raises under-attack utility to 80.51%, compared with 38.46% for the strongest recovery baseline, and achieves the highest clean utility. These results support the effectiveness of the evaluated Certified Recovery module.

AgentRecover uses the fewest model calls, averaging 4.35 per task. Its token usage averages 16.19k per task, lower than reflection, critique, and search, but higher than simple error feedback (13.87k).

Answer to RQ3. With the guardrail fixed, Certified Recovery achieves the highest clean and under-attack utility among the tested recovery modules, while retaining zero observed ASR and using the fewest model calls.

4 Related Work

We first review the development of tool-using agents, then discuss runtime guardrails and recovery from errors as approaches to safe behavior and reliable task completion.

Tool-using agents. Tool use enables LLMs to gather information and act on their environment. ReAct interleaves reasoning with actions (Yao et al., 2023), Toolformer learns when and how to call external APIs (Schick et al., 2023), and AutoGen supports collaboration among agents equipped with tools (Wu et al., 2024). As agents take on a wider range of tasks, their safety and reliable task completion become increasingly important. AgentDojo and ASB evaluate both legitimate-task performance and attacker success (Debenedetti et al., 2024; Zhang et al., 2025), while τ-bench examines task completion under domain policies (Yao et al., 2025). AgentRecover likewise aims to preserve agent safety while improving utility.

Runtime guardrails for agents. Runtime guardrails check proposed actions before they affect the environment. Policy-based methods specify permissible behavior: AgentSpec supports configurable constraints and corrective interventions, while Progent controls tool privileges (Wang et al., 2026; Shi et al., 2026b). Task-oriented methods use the legitimate goal to constrain execution: DRIFT checks planned trajectories, and Task Shield checks task alignment (Li et al., 2025; Jia et al., 2025). Other designs limit the influence of untrusted data through provenance tracking in CaMeL, tool dependencies in IPIGuard, and isolated planning in Tool-Guard (Debenedetti et al., 2026; An et al., 2025; Shi et al., 2026a). These methods strengthen runtime safety. AgentRecover builds on their use of guardrails to intercept unsafe actions and studies how to recover after interception so that legitimate tasks can continue, improving utility without relaxing safety requirements.

Recovery from errors. Feedback-based recovery helps agents revise unsuccessful decisions and continue their tasks. Reflexion and Self-Refine use verbal feedback to improve subsequent attempts (Shinn et al., 2023; Madaan et al., 2023). CRITIC grounds correction in tool interactions, AdaPlanner adapts plans to environmental feedback, and LATS searches alternative action paths (Gou et al., 2024; Sun et al., 2023; Zhou et al., 2024). For safety-related failures, ToolSafe returns explanations of detected tool-use risks to guide subsequent actions (Mou et al., 2026), while TRIAD uses structured critiques to guide plan revision (Sun et al., 2026). AgentSentry repairs compromised context (Zhang et al., 2026). These methods improve agents' ability to recover from errors. Building on the value of feedback, AgentRecover coordinates interception and recovery through precise feedback that supports Certified Recovery, with the aim of making post-interception recovery more reliable.

5 Conclusion and Future Work

We presented AgentRecover, which coordinates interception and recovery through precise guardrail feedback and checked recovery actions, without relaxing safety requirements. AgentRecover achieves zero observed ASR on AgentDojo and 0.88% ASR on ASB, while improving under-attack utility over the corresponding Unguarded Native Agent baselines. Local guardrail and recovery comparisons support these safety and utility gains.

Future work will extend AgentRecover to real-world settings with more complex tasks and a broader range of attacks. We also plan to refine the coordination between interception and recovery to further improve agent safety and utility.

AI Use Statement

In this work, we used generative AI tools for improving the readability and language quality of the manuscript and assisting with the search and retrieval of references. All AI-assisted content was reviewed and verified by the authors, who take full responsibility for the final content of this submission.

REFERENCES

Hengyu An, Jinghuai Zhang, Tianyu Du, Chunyi Zhou, Qingming Li, Tao Lin, and Shouling Ji. IPIGuard: A novel tool dependency graph-based defense against indirect prompt injection in LLM agents. In *Proceedings of the 2025 Conference on Empirical Methods in Natural Language Processing*, pp. 1023–1039. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.emnlp-main.53. URL https://aclanthology.org/2025.emnlp-main.53/.

Edoardo Debenedetti, Jie Zhang, Mislav Balunović, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. AgentDojo: A dynamic environment to evaluate prompt injection attacks and defenses for LLM agents. In *Advances in Neural Information Processing Systems*, volume 37, pp. 82895–82920, 2024. doi: 10.52202/079017-2636. URL https://proceedings.neurips.cc/paper_files/paper/2024/hash/97091a5177d8dc64b1da8bf3e1f6fb54-Abstract-Datasets_and_Benchmarks_Track.html. Datasets and Benchmarks Track.

Edoardo Debenedetti, Ilia Shumailov, Tianqi Fan, Jamie Hayes, Nicholas Carlini, Daniel Fabian, Christoph Kern, Chongyang Shi, Andreas Terzis, and Florian Tramèr. Defeating prompt injections by design. In *2026 IEEE Conference on Secure and Trustworthy Machine Learning*, SaTML '26, 2026. URL https://arxiv.org/abs/2503.18813.

DeepSeek. DeepSeek V4 preview release. https://api-docs.deepseek.com/news/news260424/, 2026.

Google. Gemini 2.5 Flash. https://ai.google.dev/gemini-api/docs/models/gemini-2.5-flash, 2026. Documentation for the June 2025 model release, accessed September 25, 2026.

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. In *The Twelfth International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=Sx038qxjek.

Keegan Hines, Gary Lopez, Matthew Hall, Federico Zarfati, Yonatan Zunger, and Emre Kiciman. Defending against indirect prompt injection attacks with spotlighting, 2024. URL https://arxiv.org/abs/2403.14720.

Feiran Jia, Tong Wu, Xin Qin, and Anna Squicciarini. The task shield: Enforcing task alignment to defend against indirect prompt injection in LLM agents. In *Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pp. 29680–29697. Association for Computational Linguistics, 2025. doi: 10.18653/v1/2025.acl-long.1435. URL https://aclanthology.org/2025.acl-long.1435/.

Sahasra Kokkula, Somanathan R, Nandavardhan R, Aashishkumar, and G Divya. Palisade – prompt injection detection framework, 2024. URL https://arxiv.org/abs/2410.21146.

Hao Li, Xiaogeng Liu, Hung-Chun Chiu, Dianqi Li, Ning Zhang, and Chaowei Xiao. DRIFT: Dynamic rule-based defense with injection isolation for securing LLM agents. In *Advances in Neural Information Processing Systems*, volume 38, pp. 83262–83290, 2025. doi: 10.52202/085713-2791. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/77f3b26c7907aa27b207df9b9d43f29a-Abstract-Conference.html.

Aman Madaan, Niket Tandon, Prakhar Gupta, Skyler Hallinan, Luyu Gao, Sarah Wiegreffe, Uri Alon, Nouha Dziri, Shrimai Prabhumoye, Yiming Yang, Shashank Gupta, Bodhisattwa Prasad Majumder, Katherine Hermann, Sean Welleck, Amir Yazdanbakhsh, and Peter Clark. Self-refine: Iterative refinement with self-feedback. In *Advances in Neural Information Processing Systems*, volume 36, pp. 46534–46594, 2023. doi: 10.52202/075280-2019. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/91edff07232fb1b55a505a9e9f6c0ff3-Abstract-Conference.html.

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. In *Findings of the Association for Computational Linguistics: ACL 2026*, pp. 37125–37153. Association for Computational Linguistics, 2026. doi: 10.18653/v1/2026.findings-acl.1850. URL https://aclanthology.org/2026.findings-acl.1850/.

OpenAI. GPT-4o mini: Advancing cost-efficient intelligence. https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/, 2024.

OpenAI. Introducing GPT-5.4 mini and nano. https://openai.com/index/introducing-gpt-5-4-mini-and-nano/, 2026.

Qwen Team. Qwen3-Max: Just scale it. https://qwen.ai/blog?id=241398b9cd6353de490b0f82806c7848c5d2777d, 2025.

Timo Schick, Jane Dwivedi-Yu, Roberto Dessì, Roberta Raileanu, Maria Lomeli, Eric Hambro, Luke Zettlemoyer, Nicola Cancedda, and Thomas Scialom. Toolformer: Language models can teach themselves to use tools. In *Advances in Neural Information Processing Systems*, volume 36, pp. 68539–68551, 2023. doi: 10.52202/075280-2997. URL https://proceedings.neurips.cc/paper/2023/hash/d842425e4bf79ba039352da0f658a906-Abstract-Conference.html.

Shanghao Shi, Xiao Wang, Chaoyu Zhang, Hao Li, Wenjing Lou, Thomas Hou, Yevgeniy Vorobeychik, Chongjie Zhang, and Ning Zhang. Think twice before you act: Protecting LLM agents against tool description poisoning via isolated planning. *arXiv preprint arXiv:2606.20922*, 2026a. doi: 10.48550/arXiv.2606.20922. URL https://arxiv.org/abs/2606.20922. Accepted to ICML 2026.

Tianneng Shi, Jingxuan He, Zhun Wang, Hongwei Li, Linyu Wu, Wenbo Guo, and Dawn Song. Progent: Securing AI agents with privilege control, 2026b. URL https://arxiv.org/abs/2504.11703v3. Version 3, revised May 14, 2026.

Noah Shinn, Federico Cassano, Ashwin Gopinath, Karthik Narasimhan, and Shunyu Yao. Reflexion: Language agents with verbal reinforcement learning. In *Advances in Neural Information Processing Systems*, volume 36, pp. 8634–8652, 2023. doi: 10.52202/075280-0377. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/1b44b878bb782e6954cd888628510e90-Abstract-Conference.html.

Haotian Sun, Yuchen Zhuang, Lingkai Kong, Bo Dai, and Chao Zhang. AdaPlanner: Adaptive planning from feedback with language models. In *Advances in Neural Information Processing Systems*, volume 36, pp. 58202–58245, 2023. doi: 10.52202/075280-2537. URL https://proceedings.neurips.cc/paper_files/paper/2023/hash/b5c8c1c117618267944b2617add0a766-Abstract.html.

Yuhao Sun, Jiacheng Zhang, Shaanan Cohney, Zhexin Zhang, Feng Liu, and Xingliang Yuan. From risk classification to action plan remediation: A guardrail feedback driven framework for LLM agents, 2026. URL https://arxiv.org/abs/2606.05805.

Haoyu Wang, Christopher M. Poskitt, and Jun Sun. AgentSpec: Customizable runtime enforcement for safe and reliable LLM agents. In *Proceedings of the 48th IEEE/ACM International Conference on Software Engineering*, ICSE '26. Association for Computing Machinery, 2026. doi: 10.1145/3744916.3764546. URL https://cposkitt.github.io/files/publications/agentspec_llm_enforcement_icse26.pdf.

Qingyun Wu, Gagan Bansal, Jieyu Zhang, Yiran Wu, Beibin Li, Erkang Zhu, Li Jiang, Xiaoyun Zhang, Shaokun Zhang, Jiale Liu, Ahmed Hassan Awadallah, Ryen W. White, Doug Burger, and Chi Wang. AutoGen: Enabling next-gen LLM applications via multi-agent conversations. In *First Conference on Language Modeling*, 2024. URL https://openreview.net/forum?id=BAakY1hNKS.

Shunyu Yao, Jeffrey Zhao, Dian Yu, Nan Du, Izhak Shafran, Karthik Narasimhan, and Yuan Cao. ReAct: Synergizing reasoning and acting in language models. In *The Eleventh International Conference on Learning Representations*, 2023. URL https://openreview.net/forum?id=WE_vluYUL-X.

Shunyu Yao, Noah Shinn, Pedram Razavi, and Karthik Narasimhan. τ-bench: A benchmark for tool-agent-user interaction in real-world domains. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=roNSXZpUDN.

Hanrong Zhang, Jingyuan Huang, Kai Mei, Yifei Yao, Zhenting Wang, Chenlu Zhan, Hongwei Wang, and Yongfeng Zhang. Agent security bench (ASB): Formalizing and benchmarking attacks and defenses in LLM-based agents. In *The Thirteenth International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=V4y0CpX4hK.

Tian Zhang, Yiwei Xu, Juan Wang, Keyan Guo, Xiaoyang Xu, Bowen Xiao, Quanlong Guan, Jinlin Fan, Jiawei Liu, Zhiquan Liu, and Hongxin Hu. AgentSentry: Mitigating indirect prompt injection in LLM agents via temporal causal diagnostics and context purification, 2026. URL https://arxiv.org/abs/2602.22724.

Andy Zhou, Kai Yan, Michal Shlapentokh-Rothman, Haohan Wang, and Yu-Xiong Wang. Language agent tree search unifies reasoning, acting, and planning in language models. In *Proceedings of the 41st International Conference on Machine Learning*, volume 235 of *Proceedings of Machine Learning Research*, pp. 62138–62160. PMLR, 2024. URL https://proceedings.mlr.press/v235/zhou24r.html.

A Appendix

A.1 Method Supplement

A.1.1 Safety checks and feedback

Safety Constraint. `constraints_for` selects conditions by tool effect and argument role. `evaluate` applies the corresponding operations in Table 5. A condition fails when it is violated or its required support cannot be established.

Table 5: **Safety conditions.**

Condition	Executable check
Operation authorization	Match the tool's effect to supported request patterns; retrieved instructions require explicit user delegation.
Explicit permission	Require explicit user permission for destructive operations or password updates.
Incidental effects	Require authorization for a declared mutation, either explicitly or as an inseparable part of the requested operation.
Argument support	Bind argument values to the request or eligible records through a role-compatible selection or calculation.
Required values	Reject empty supplied required values; read-only wildcard queries require another bound selector or an explicit full-enumeration contract.
Resource access	Check that a read-only resource locator occurs in the request or eligible data, excluding occurrences found only in recognized instruction spans.
Duplicate effects	Reject an already completed effect or repetition of an operation authorized only once.

Root Cause. `identify_cause` maps an unmet condition to the argument positions used by that check. Operation-level failures can involve the entire call. A reported position may need a different value or support for its existing value; it is not necessarily incorrect.

Permitted Source. `allowed_source` first selects records or retrievals whose declared outputs match the argument's role. It then binds the selector to the user request or an already supported record and checks source permissions. Related arguments must refer to the same selected entity; single-record construction requires a unique match. If existing records are insufficient, a permitted retrieval can become a candidate action, provided its query arguments are supported. It must pass both checks before execution. Otherwise, the argument remains unresolved.

A.1.2 CONSTRUCTION AND CHECKS

Constructing candidates. The constructor selects templates compatible with Root Cause and Permitted Source. These combine record filtering, unique selection, field projection, and arithmetic on supported operands. Candidates with unmet preconditions are discarded; unresolved or conflicting bindings do not justify a guessed value. Construction can retain a supported argument, replace it with a supported alternative, or select a task-aligned tool when the intercepted tool is unauthorized.

Admission and execution. The Safety check reapplies all applicable conditions to the candidate, including unchanged arguments. Critical derived values are checked against their sources and the supported selection or calculation. The Task-progress check accepts an action that implements unfinished work or obtains information required to construct that action. It checks the tool effect and argument bindings against the pending item; an unrelated lookup does not qualify. Both checks must pass before execution, and only the observed result updates confirmed effects. Construction can screen multiple candidates, but a failed final check does not trigger an automatic queue of alternative calls.

A.2 EXPERIMENTAL POPULATIONS AND RUNTIME CONFIGURATION

The evaluated task populations were available during method development. Experiments use configured tool permissions and construction templates.

AgentDojo populations. We use package version 0.1.35, benchmark version 1.2, and `important_instructions` injection. Banking, Slack, travel, and workspace contain 16/144, 21/105, 20/140, and 40/560 clean/attacked cases, respectively. RQ1, local RQ2, and RQ3 weight the 949 attacked cases equally; RQ1 averages the five model-level rates with equal weight. For comparison with published guardrails in Table 2, we retain all non-workspace cases and workspace injection tasks 0–5 for each user task, yielding 144, 105, 140, and 240 attacked cases (629 total). The four scenario rates receive equal weight.

ASB populations. We use commit `1f561dccf92d55302368fa67679b4ba9d9c8fdc4`. Each attack workload pairs the first task in each of ten scenarios with 40 compatible attacker tools. DPI uses `naive`; OPI uses `context_ignoring` (indirect prompt injection in the ASB paper). Both use the same ten base tasks, producing 800 attack variants in total. The main table pools these variants with equal case weights. Clean utility uses all 51 tasks, including these ten. The utility columns are therefore not paired measurements of attack-induced change. Clean, DPI, and OPI runs use the same controller. ASB evaluates tool-level recovery with parameter-free tools and model-based authorization and task-progress judgments; AgentDojo also tests argument-value construction. Across both benchmarks, clean evaluation covers 148 distinct tasks, and the 1,749 attack variants derive from 107 of them.

Models and run control. The five API aliases span OpenAI, DeepSeek, Alibaba, and Google, including efficiency-oriented and larger-capacity model families (OpenAI, 2024; 2026; Google, 2026; DeepSeek, 2026; Qwen Team, 2025). Each native/AgentRecover pair shares its endpoint profile, temperature, and tool interface; DeepSeek uses non-thinking tool mode in both arms. Each case has one recorded execution; transport retries address failed requests, not semantic failures. Ten ASB Instructional Prevention OPI cases affected by connection errors were rerun, and their completed reruns are used in the reported counts.

A.3 EXACT COUNTS AND DOMAIN-LEVEL RESULTS

Table 6 gives the integer numerators for Figure 4. Across five models, the Unguarded Native Agent records 1,307 attacker successes and 2,740 completed tasks on 4,745 attacked trajectories; AgentRecover records zero and 4,147, respectively. Clean-task completions are 406 and 431 out of 485. We compute differences before rounding: the mean under-attack utility gain is $100(4147 - 2740)/4745 = 29.6523\ldots$ percentage points.

Table 6: **Five-model integer counts.** Separate rows report Unguarded Native Agent (Debenedetti et al., 2024) and AgentRecover. Attack columns use 949 cases per model; the clean column uses 97 tasks.

Model	Method	Attacker successes	Under-attack task successes	Clean-task successes
GPT-4o-mini	Unguarded Native Agent	246	401	70
	AgentRecover	0	764	79
GPT-5.4-mini	Unguarded Native Agent	24	708	78
	AgentRecover	0	818	86
DeepSeek-V4-Pro	Unguarded Native Agent	101	778	88
	AgentRecover	0	865	90
Qwen3-Max	Unguarded Native Agent	377	532	83
	AgentRecover	0	837	85
Gemini-2.5-Flash	Unguarded Native Agent	559	321	87
	AgentRecover	0	863	91

Table 7: **AgentDojo domain-level results with GPT-4o-mini.** Separate rows report Unguarded Native Agent (Debenedetti et al., 2024) and AgentRecover.

Suite (case counts)	Method	Clean utility (%)	Under-attack utility (%)	ASR (%)
Banking	Unguarded Native Agent	50.00	41.67	42.36
(16 clean, 144 attacked)	AgentRecover	75.00	60.42	0.00
Slack	Unguarded Native Agent	80.95	51.43	59.05
(21 clean, 105 attacked)	AgentRecover	95.24	94.29	0.00
Travel	Unguarded Native Agent	60.00	39.29	24.29
(20 clean, 140 attacked)	AgentRecover	60.00	55.00	0.00
Workspace	Unguarded Native Agent	82.50	41.43	15.89
(40 clean, 560 attacked)	AgentRecover	87.50	89.46	0.00

For Table 1, the local counts of clean-task successes, under-attack task successes, and attacker successes, in that order, are (70, 401, 246) for Unguarded Native Agent, (68, 454, 208) for Spotlighting, (67, 490, 91) for Repeat User Prompt, (64, 552, 33) for Tool Filter, and (79, 764, 0) for AgentRecover. For AgentRecover in Table 2, clean-task successes are 12/16, 20/21, 12/20, and 35/40 in banking, Slack, travel, and workspace, respectively. Under-attack task successes are 87/144, 99/105, 77/140, and 215/240, with zero attacker successes in each scenario. Averaging these rates gives 79.43% clean utility, 74.82% under-attack utility, and 0.00% ASR. Table 8 provides the counts used to combine ASB's attack conditions. AgentRecover completes $265 + 244 = 509$ of 800 attacked cases and 18 of 51 clean tasks. DPI utility and ASR are 66.25% and 0.00%; OPI utility and ASR are 61.00% and 1.75%. The seven OPI attacker successes yield a combined ASR of $7/800 = 0.875\%$. Percentages are computed from counts and rounded to two decimal places, with half values rounded upward.

Table 8: **ASB integer counts underlying the combined comparison.** Each attack condition uses 40 variants per task over the same 10 base user tasks (400 cases). Clean-task successes are out of all 51 tasks.

Method	Clean-task successes	DPI		OPI	
		Task successes	Attacker successes	Task successes	Attacker successes
Unguarded Native Agent (Zhang et al., 2025)	20	0	376	133	173
Delimiters (Zhang et al., 2025)	24	1	393	127	190
Instructional Prevention (Zhang et al., 2025)	27	7	337	138	131
DRIFT (Li et al., 2025)	18	0	342	123	17
AgentRecover	18	265	0	244	7

A.4 Recovery-Module Replacement Protocol

Replacement strategies. Each alternative receives the same public feedback: the intercepted call and arguments, failed rules, argument positions, failure reasons, and admission conditions.

- **Tool Error Feedback** returns interception as a tool error. The task model continues without an auxiliary call (Yao et al., 2023).
- **Self-Reflection** adds one auxiliary model call to explain the interception and suggest a safer step (Shinn et al., 2023).
- **CRITIC-style Repair** uses one auxiliary call to select a checking interaction; the task model executes it and revises its action from the result (Gou et al., 2024).
- **LATS-style Search** uses three auxiliary calls to propose branches and a fourth to select one by expected safety and progress; the task model executes the selected step (Zhou et al., 2024).

These strategies adapt the cited recovery ideas to this setting; LATS-style search uses bounded branches rather than full tree search. Auxiliary calls cannot execute tools.

Submitting actions to the guardrail. Tool Error Feedback and Self-Reflection submit ordinary calls. CRITIC-style and LATS-style recovery use a shared adapter to derive and validate arguments for the policy-selected tool after the policy responds to the first interception. The adapter cannot select tools, retrieve data, or invoke AgentRecover's controller.

Continuation. All four alternatives use GPT-4o-mini (temperature 0), with at most five strategy activations across successive interceptions and 24 task-model turns. AgentRecover retains its recorded continuation from the same boundary, including action and final-response checks.