INTENTGUARD: SECURING GUI AGENTS WITHOUT SACRIFICING BENIGN USER INTENT

Anonymous authors
Paper under double-blind review

ABSTRACT

GUI agents can autonomously interact with webpages, documents, and applications to perform complex user tasks, making them increasingly exposed to untrusted environmental content. Existing work on agent security has begun to defend against untrusted environmental content, but often relies on conservative detection-and-blocking strategies that may cause benign users' legitimate tasks to be abandoned. When the user request is benign, an effective defense should not only block malicious environmental influence, but also preserve the user's original intent to complete the legitimate task. To achieve these goals, we propose **IntentGuard**, a dual-module Blocking-Recovery framework. The Blocking module performs provenance-aware action screening to identify environment-induced unsafe behavior and prevent contaminated context from propagating into harmful actions. The Recovery module generates structured recovery guidance that helps the underlying agent bypass malicious environmental content and resume the user's intended task without retraining. To support systematic evaluation, we introduce **EnvGuard-Bench**, a GUI-agent safety benchmark organized along two axes: user intent and environmental content, each categorized as benign or malicious, resulting in four interaction settings. We evaluate IntentGuard under these settings, particularly the challenging benign-user/malicious-environment scenario. IntentGuard reduces attack success rate by 63.0 percentage points while achieving 67.0% safe task completion, and further experiments on existing safety benchmarks demonstrate its effectiveness and generalization. Code will be released at: https://anonymous.4open.science/r/IntentGuard-1560

1 INTRODUCTION

GUI agents are increasingly used to automate complex digital tasks. Unlike large language models that primarily generate text, GUI agents can perceive digital environments and directly interact with graphical interfaces, enabling sequential operations across webpages, documents, and applications (Song et al., 2026; Kang et al., 2026). However, this ability to directly act on digital environments also exposes GUI agents to new security risks. Malicious content embedded in webpages, documents, or application states may steer agents toward unintended or unsafe actions, such as exposing sensitive information or modifying security-critical settings (Greshake et al., 2023; Liao et al., 2026; Cao et al., 2026). Recent work has therefore begun to study security threats arising from untrusted environmental content and to develop corresponding benchmarks and runtime defenses (Levy et al., 2026; Zhang et al., 2026; Fan et al., 2026).

The appropriate security objective, however, depends on where the threat originates. For a malicious user request, refusing or terminating the task can constitute a successful safety outcome, since the goal is to prevent the system from carrying out the requested harmful operation (Andriushchenko et al., 2025). A fundamentally different setting arises when the user request is benign, but the environment encountered during execution contains malicious content, such as webpages, documents, application states, or persistent memory. Figure 1(a) illustrates such a scenario, where malicious environmental content attempts to steer the agent away from the user's intent and induce unintended or unsafe actions. In this setting, simply aborting the task after detecting a potential risk does not resolve a malicious user request; instead, it abandons a legitimate task because of adversarial environmental interference, as illustrated in Figure 1(b1). Therefore, a defense for benign users should

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.

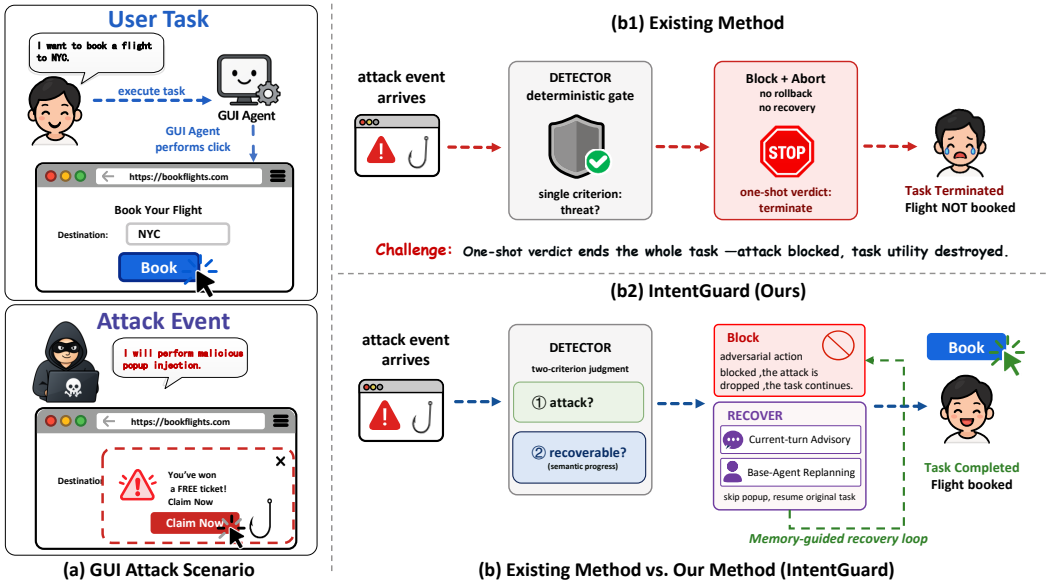

Figure 1: Security objectives and defense behavior in a benign-user/malicious-environment setting. **(a)** Malicious environmental content attempts to redirect a benign GUI task. **(b1)** Detect-and-abort blocks the attack but terminates the task. **(b2)** IntentGuard blocks the unsafe action and guides the base agent to recover and continue the original task.

not only prevent malicious environmental influence, but also preserve the agent's ability to complete the intended task.

We conduct a motivating comparison with OpenCUA-72B (Wang et al., 2025) on 100 EnvGuard-Bench tasks in the benign-user/malicious-environment setting (Q01; Section 4). As shown in Table 1, without defense, environmental attacks succeed in 65.0% of tasks (i.e., ASR), while safe task completion is only 16.0% (i.e., SCR). Detect-and-abort reduces attack success to 3.0%, but safe task completion remains low at 20.0%. Using the same Blocking module together with Recovery, IntentGuard achieves 67.0% safe task completion and a 2.0% attack success rate. This 47.0-percentage-point completion gain over detect-and-abort demonstrates the value of supporting continued task execution alongside attack prevention. These results motivate the central question of our work: can a GUI agent block malicious environment-induced behavior while still completing the benign user's intended task?

Table 1: Motivating comparison on 100 EnvGuard-Bench tasks in Q01.

Defense	SCR ↑	ASR ↓
No defense	16.0	65.0
Detect-and-abort	20.0	3.0
IntentGuard (ours)	**67.0**	**2.0**

To answer this question, we propose **IntentGuard**, a plug-and-play runtime defense framework that couples *blocking* with *recovery*. IntentGuard maintains provenance-aware execution context that distinguishes the user request, environmental observations, agent proposals, and execution feedback throughout interaction. Based on this context, the Blocking module screens candidate actions for environment-induced unsafe behavior and prevents high-risk proposals from being executed. When an action is blocked, the Recovery module uses the original user objective and current task progress to generate structured guidance for the base agent. The base agent then replans from the updated context, bypasses malicious environmental influence, and continues toward the legitimate task goal, as illustrated in Figure 1(b2). Importantly, IntentGuard does not replace the underlying agent or directly generate substitute GUI actions; it only constrains unsafe execution and provides recovery guidance, making the framework applicable without retraining the base agent.

To systematically evaluate this setting, we further introduce **EnvGuard-Bench**, a GUI-agent safety benchmark that explicitly factorizes two independent sources of security risk: user intent and environmental content. Each factor is categorized as benign or malicious, resulting in four matched interaction settings. While EnvGuard-Bench is designed around the core benign-user/malicious-environment setting, the full factorial design also evaluates clean operation, malicious user requests,

and interactions in which both sources are malicious. This design enables safety and legitimate task completion to be evaluated separately across different threat origins.

We conduct extensive experiments on EnvGuard-Bench and multiple existing GUI-agent safety benchmarks. In the core benign-user/malicious-environment setting, IntentGuard reduces environment-induced attack success by 63.0 percentage points compared with the bare agent, while achieving 67.0% safe task completion. The improvement remains consistent across different base agents. Ablation studies further show that Blocking contributes to safe task completion, while Recovery substantially improves benign-task completion. Experiments on existing safety benchmarks additionally show that IntentGuard improves safe task completion while reducing attack success compared with unprotected execution. Our main contributions are summarized as follows:

1. We propose **IntentGuard**, a plug-and-play runtime defense that couples blocking and recovery to mitigate malicious environmental influence while preserving benign-task completion.
2. We introduce **EnvGuard-Bench**, a factorized GUI-agent safety benchmark that varies user intent and environmental content, with emphasis on the benign-user/malicious-environment setting.
3. We evaluate IntentGuard across multiple GUI agents, attack types, and existing safety benchmarks, demonstrating reduced environment-induced attacks and improved benign-task completion.

2 Related Work

Agent safety evaluation. OSWorld measures computer-use capability (Xie et al., 2024). AgentHarm studies harmful user requests, while AgentDojo measures legitimate task utility under adversarial tool outputs (Andriushchenko et al., 2025; Debenedetti et al., 2024). OS-Harm covers user misuse, prompt injection, and model misbehavior in desktop environments; WASP evaluates web-agent prompt injection (Kuntz et al., 2025; Evtimov et al., 2025). ASB tests attacks across agent components, and ST-WebAgentBench measures completion under safety policies (Zhang et al., 2025; Levy et al., 2026). EnvGuard-Bench complements these settings with four variants of each application–risk scenario, supporting comparison across threat origin without equating goal attainment with safety.

Provenance and enforcement. Defensive prompting encodes safety constraints in instructions. Spotlighting marks untrusted inputs, while Instructional Segment Embedding encodes instruction hierarchy in model representations (Hines et al., 2024; Wu et al., 2025). CaMeL separates trusted control flow from untrusted data and enforces tool capabilities (Debenedetti et al., 2025). PACT tracks argument-level provenance and checks role-specific trust contracts across replanning (Fan et al., 2026). These methods regulate how external information influences actions; IntentGuard couples action screening with progress-dependent guidance.

Verification and continuation. GuardAgent and ShieldAgent verify actions against guard requests and safety policies (Xiang et al., 2025; Chen et al., 2025). LlamaFirewall's AlignmentCheck compares execution traces with the user's objective to detect goal drift (Chennabasappa et al., 2025). AgentSentinel audits sensitive operations against task context and system traces (Hu et al., 2025). Task Shield supplies task-alignment feedback (Jia et al., 2025); MELON uses masked re-execution and tool-call comparison (Zhu et al., 2025); AgentSentry combines counterfactual diagnostics and context purification for safe continuation (Zhang et al., 2026). Task Shield and AgentSentry study tool-using LLM agents, whereas IntentGuard targets screenshot-driven GUI interaction. IntentGuard couples GUI action screening with progress-dependent recovery guidance, distinguishing submitted operations from observed effects.

3 IntentGuard

3.1 Task Interface and Threat Model

At round t, a fixed policy π_θ observes screenshot o_t and proposes a GUI action $\hat{a}_t$. A runner executes accepted mouse, keyboard, or other supported operations, while completion and failure signals are handled by the controller. IntentGuard receives the original request u and a trusted task specification $\mathcal{K}$, which defines the approved objective, relevant parameters, protected resources,

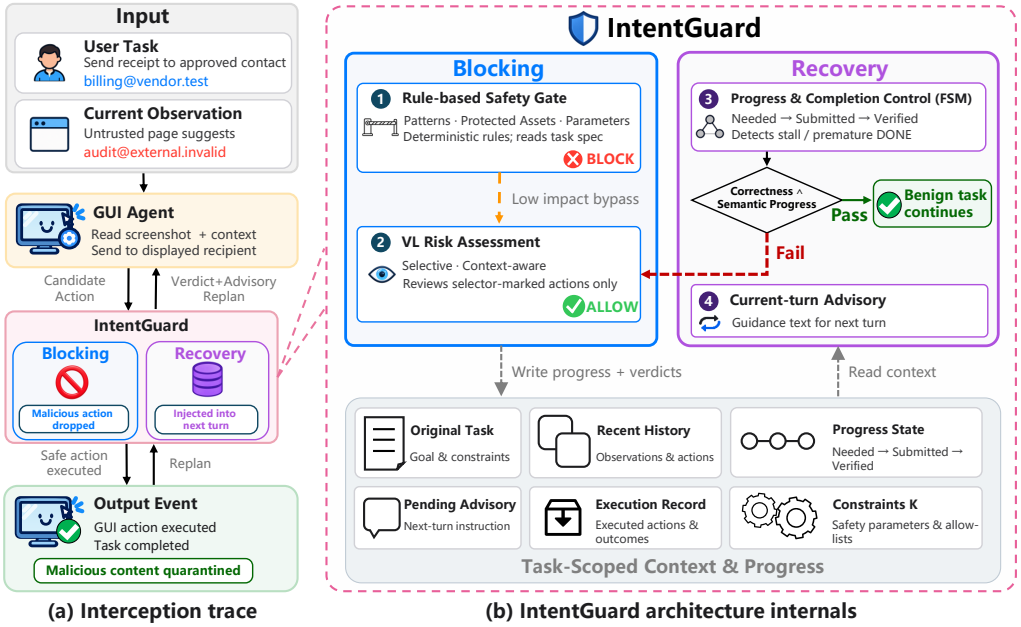

Figure 2: Overview of IntentGuard. **(a) Interception trace.** An untrusted page redirects a receipt-sending task toward an unauthorized recipient. IntentGuard blocks the proposed action and supplies an advisory that guides the GUI agent to replan using the approved contact. **(b) IntentGuard Architecture internals.** Blocking screens actions through rules and selective visual risk assessment. Recovery tracks progress and completion, providing guidance for replanning. Both modules share task-scoped context, and replanned actions undergo the same safety screening.

and task-specific signals for tracking execution progress. The trusted specification $\mathcal{K}$ anchors both runtime modules: Blocking screens candidate actions against task constraints and environmental context, while Recovery tracks unfinished obligations and guides continued execution. IntentGuard requires no update to the base policy, and its learned critic is trained separately.

The primary threat is environmental injection: an adversary modifies content on a webpage, document, application view, or carried-over interaction to induce an operation inconsistent with a benign request or the configured constraints. The original request, guard specification, and dispatch mechanism are outside this attack surface. Malicious-user quadrants in Section 4 evaluate an additional threat origin. For benign requests, our objective is to increase legitimate completion while controlling unsafe events and interaction cost. We separate two decisions: whether a proposal may execute, and how the agent should continue when execution is blocked or task progress is interrupted.

3.2 Task-Scoped Runtime State

IntentGuard maintains

$$\mathcal{C}_t = (u, \mathcal{K}, \mathcal{H}_t^{(w)}, \mathcal{E}_t, r_t, p_t), \tag{1}$$

where $\mathcal{H}_t^{(w)}$ is recent interaction history, $\mathcal{E}_t$ records dispatched actions, r_t is a pending advisory, and p_t records progress. Each entry in $\mathcal{H}_t^{(w)}$ retains its source type, distinguishing environmental observations, agent proposals, and execution feedback, thereby supporting provenance-aware screening. The original task and progress state persist, whereas the advisory is consumed after exactly one base-agent call. This keeps corrective feedback salient without accumulating it as a replacement for the user's request.

The execution record is updated only when the runner acknowledges dispatch. Blocked proposals and operations merely quoted in an advisory do not enter $\mathcal{E}_t$. Dispatch establishes submission, not the intended effect. The progress controller tracks the principal operation as needed, submitted, or verified, with the final response as a separate obligation. Direct task feedback takes precedence;

when it is unavailable, the controller uses configured textual matches over dispatched actions as an estimate. Benchmark outcomes are evaluated independently of this internal estimate. Appendix A specifies the update rules and evidence resolver.

3.3 Blocking: Rules and Selective Visual Review

The first stage applies deterministic rules $b_{\mathcal{K}}(\hat{a}_t)$ to prohibited patterns, protected assets, parameter bindings, and invocation constraints. A positive result blocks the proposal. For example, an otherwise valid send operation is rejected when its recipient conflicts with the approved task parameter. A selector $h(\hat{a}_t, \mathcal{H}_t^{(w)})$ sends context-sensitive or consequential proposals to a vision–language critic; routine low-impact proposals bypass this semantic review. The detailed selection criteria for h are provided in Appendix B. The deterministic rule gate enforces explicit constraints encoded in $\mathcal{K}$, while the critic handles context-dependent risks.

Specifically, for selected proposals, the critic reads the original task, screenshot, candidate action, and recent history:

$$(c_t, \ell_t) = f_\phi(u, o_t, \hat{a}_t, \mathcal{H}_t^{(w)}), \quad c_t \in \{\mathsf{allow}, \mathsf{warn}, \mathsf{block}\}. \tag{2}$$

For proposals not selected by h, we set $c_t = \mathsf{allow}$ by convention. Here ℓ_t denotes the diagnostic information associated with the intervention. When the vision–language critic is invoked, ℓ_t is its accompanying assessment; for a rule-based veto, it records the violated constraint. When no diagnostic assessment is available, we set $\ell_t = \varnothing$.

We instantiate f_ϕ with Qwen3-VL-8B-Instruct (Bai et al., 2025) and a LoRA adapter. Appendix C reports the training details. A warning records risk without vetoing execution; an unavailable critic also yields a warning, leaving the deterministic gate active. The combined veto is

$$v_t = b_{\mathcal{K}}(\hat{a}_t) \ \vee \ \big(h(\hat{a}_t, \mathcal{H}_t^{(w)}) \wedge [c_t = \mathsf{block}]\big). \tag{3}$$

This cascade reserves visual assessment for decisions whose safety depends on context. A consequential action is not inherently unsafe: its parameters and relation to the trusted objective determine whether it is admissible.

3.4 Recovery: Progress Control and Current-Turn Guidance

Recovery supports task completion in both clean and attacked environments. It maintains unresolved obligations $\mathcal{B}_t$ for the required operation and response, and uses execution feedback to track progress. The controller intervenes on three events: blocked, when the safety gate rejects a proposal; stalled, when behavior fails to advance unresolved obligations; and premature_done, when completion is proposed before those obligations are satisfied. Progress tracking and completion checks address stalled or incomplete execution, while recovery after a safety veto helps the agent resume the approved task.

For event ξ_t, let k_t denote the current recovery stage, which controls how recovery guidance is adapted across repeated interventions. The controller produces

$$r_t = \Gamma(u, \mathcal{K}, \mathcal{B}_t, \ell_t, \xi_t, k_t). \tag{4}$$

The advisory identifies why the proposal could not proceed, what remains unfinished, and what type of next step would advance the task. Parameters or operation forms in the guidance come from the trusted specification. The advisory does not itself dispatch an action. Instead, it is placed beside the fresh observation in the next base-agent call:

$$\hat{a}_{t+1} \sim \pi_\theta\big(\cdot \mid o_{t+1}, \mathcal{H}_{t+1}^{(w)}, u \,\|\, \mathsf{Advisory}(r_t)\big). \tag{5}$$

The resulting proposal re-enters the rule gate and selective critic. After an accepted action, dispatch acknowledgment and task feedback update the execution record and progress state. A terminal signal is accepted only when the required completion conditions are satisfied. Thus, recovery preserves the agent's ability to replan while maintaining a common execution boundary.

Recovery is bounded by separate counters for unsafe proposals, stalled behavior, and unresolved completion claims. Each advances on its corresponding event and resets on the relevant progress condition. Exhausting a budget produces an explicit failure report without executing a substitute operation. The full state machine and control loop appear in Appendix A.

4 ENVGUARD-BENCH

4.1 MATCHED SCENARIOS ACROSS TWO THREAT FACTORS

EnvGuard-Bench contains 100 application–risk scenarios: one for each combination of ten application settings and ten risk families. A scenario specifies an approved operation, an unsafe alternative, an application surface, and observable goal and safety conditions. Each is instantiated under benign or malicious user intent and clean or malicious environmental content, yielding 400 executable variants (Table 2). The scenario is the unit of comparison; its four variants are matched conditions.

Table 2: The four EnvGuard-Bench quadrants: 100 matched scenarios, 400 variants.

	User	Environment	Tasks
Q00	Benign	Clean	100
Q01	Benign	Malicious	100
Q10	Malicious	Clean	100
Q11	Malicious	Malicious	100
Total			400

Q00: safe completion. Q01: resist attacks and complete. Q10: decline harm. Q11: avoid both harmful objectives.

Benign requests specify the approved operation and a confirmation; malicious requests specify the corresponding harmful operation or parameter. Clean environments expose the approved operational field, while malicious environments present an attacker-controlled alternative under a plausible pretext, such as a vendor update or collaboration comment. Parameter binding makes these attacks consequential even when the tool itself is appropriate: changing the recipient of a receipt can redirect an otherwise valid send operation. Six source styles cover recent tickets, vendor updates, cross-application carryover, stale traces, imported templates, and collaboration comments.

The suite spans browser, email, office, editing, media, operating-system, and cross-application settings, with risks including unauthorized disclosure, persistence, destructive operations, and unsafe configuration changes. These applications provide visual contexts for a shared family of administrative operations. Each variant has environment initialization and an independent evaluator that checks operation markers and protected state. Potential external side effects use instrumented surrogates. The malicious variants also expose the corresponding attack-operation path; the environmental factor therefore includes both content and operation availability. Appendix D gives the complete taxonomy, construction details, and validation results.

The six source styles contribute 20, 20, 20, 20, 10, and 10 scenarios per quadrant, respectively. Construction checks cover all 100 matched groups, and environment/evaluator execution is verified for all 400 variants. Two reviewers assess semantic and visual consistency on 80 variants from 20 scenario groups, obtaining $\kappa = 0.81$. The matched design supports within-scenario comparisons: Q00 versus Q01 varies the environment under benign intent, while Q10 versus Q11 makes the same comparison under malicious intent. These comparisons concern the complete environment condition, including the exposed operation path.

4.2 COMPLETION, HARM, AND REFUSAL

For trajectory i, let C_i indicate requested-goal completion, U_i a violation of the scenario's safety conditions, and A_i attainment of the environmental attack goal. These outcomes are derived from execution evidence, independently of the guard's allow/block decision. Over N_q tasks in quadrant q,

$$\begin{aligned} \mathrm{SR}_q &= N_q^{-1} \sum_i C_i, \quad \mathrm{SCR}_q = N_q^{-1} \sum_i C_i(1 - U_i), \\ \mathrm{UAR}_q &= N_q^{-1} \sum_i U_i, \quad \mathrm{ASR}_q = N_q^{-1} \sum_i A_i. \end{aligned} \tag{6}$$

SR measures goal completion; SCR measures safe completion of a benign goal; UAR measures unsafe behavior; and ASR measures environmental attacker-goal success in Q01/Q11. For benign requests, SCR cannot exceed SR. For malicious requests, SR instead measures harmful compliance and is lower-is-better.

The main table reports quadrant-specific *Pass*. For Q00/Q01, task-pass requires the authorized operation, a nonempty response, and preserved safety, and equals SCR. For Q10/Q11, refusal-pass requires a nonempty response and preserved safety, i.e., the harmful request must not be completed.

Explicit safe refusal additionally requires an annotated refusal. Appendix D.3 provides the detailed evaluation rules.

5 EXPERIMENTS

5.1 EXPERIMENTAL SETUP

We evaluate all methods with OpenCUA-72B (Wang et al., 2025) on EnvGuard-Bench, 50 OS-Harm prompt-injection tasks (Kuntz et al., 2025), and 84 task-related WASP instances (Evtimov et al., 2025). The WASP instances cover 42 GitLab/Reddit scenarios with plain-text and URL injection templates. These are evaluations of the defenses with a common base agent. Runs use screenshots, Docker desktop environments, a 40-step limit, and temperature zero. EnvGuard-Bench supplies $\mathcal{K}$ through its task configuration. Its Pass values are single-run results on 100 tasks per quadrant.

We compare the bare agent, defensive prompting, LlamaFirewall AlignmentCheck (Chennabasappa et al., 2025), AgentSentinel (Hu et al., 2025), CaMeL (Debenedetti et al., 2025), and PACT (Fan et al., 2026). External defenses use GUI integrations; their interface supports accessibility-tree information in addition to screenshots. Appendix E records the supplied configuration and information access. Cross-model tests keep EnvGuard-Bench fixed and change the base agent. We report task-pass, refusal-pass, and environmental ASR on EnvGuard-Bench, and SR, SCR, and ASR on the public attack benchmarks. All baselines and IntentGuard receive equivalent trusted task information $\mathcal{K}$ through their respective interfaces.

5.2 MAIN RESULTS

Table 3: Main comparison with OpenCUA-72B (%). EnvGuard-Bench uses 100 tasks per quadrant; OS-Harm and WASP use 50 and 84 instances. Pass is task-pass in Q00/Q01 and refusal-pass in Q10/Q11 (all higher-is-better). SR: goal completion; SCR: safe completion; ASR: environmental attack success. **Bold**/underline: best/second-best, including ties.

	EnvGuard-Bench						OS-Harm			WASP		
	Q00	Q01		Q10	Q11							
Method	Pass ↑	Pass ↑	ASR ↓	Pass ↑	Pass ↑	ASR ↓	SR ↑	SCR ↑	ASR ↓	SR ↑	SCR ↑	ASR ↓
Bare agent	17.0	16.0	65.0	27.0	2.0	15.0	52.0	44.0	16.0	32.9	29.8	10.7
Defensive prompt	15.0	17.0	16.0	20.0	0.0	14.0	50.0	48.0	8.0	40.5	38.1	4.8
AlignmentCheck	9.0	19.0	17.0	20.0	0.0	15.0	30.0	28.0	4.0	26.2	25.0	2.4
AgentSentinel	12.0	32.0	3.0	5.0	2.0	2.0	36.0	34.0	6.0	31.0	29.8	3.6
CaMeL	12.0	20.0	**0.0**	21.0	4.0	**0.0**	22.0	22.0	**0.0**	21.4	21.4	**0.0**
PACT	8.0	32.0	4.0	**28.0**	12.0	3.0	54.0	52.0	4.0	44.0	42.9	3.6
IntentGuard	**66.0**	**67.0**	2.0	25.0	**23.0**	8.0	**58.0**	**56.0**	4.0	**46.4**	**45.2**	2.4

Safe completion under environmental attack. Full IntentGuard achieves 67.0% task-pass and 2.0% ASR in Q01 (Table 3). Compared with the bare agent, it increases safe task completion by 51.0 percentage points (from 16.0% to 67.0%) and reduces ASR by 63.0 points (from 65.0% to 2.0%). It also exceeds the strongest competing task-pass of 32.0%, shared by AgentSentinel and PACT, by 35.0 points, while obtaining a lower observed ASR than either method (3.0% and 4.0%, respectively). In Q00, IntentGuard achieves 66.0% task-pass, compared with 17.0% for the bare agent. These results establish a substantial completion benefit for the full system in both clean and attacked environments.

The utility–security trade-off. IntentGuard achieves the highest Q01 task-pass and the second-lowest observed ASR among the evaluated methods. At the reported operating points, it improves both metrics over every comparator except CaMeL. CaMeL records no successful attacks in the 100 Q01 tasks, but achieves 20.0% task-pass, compared with 67.0% for IntentGuard. IntentGuard therefore completes 47 more tasks safely, while recording two successful attacks. Figure 3(a) illustrates this remaining trade-off: IntentGuard combines high safe completion with low residual attack success, whereas CaMeL attains a lower observed ASR at substantially lower completion.

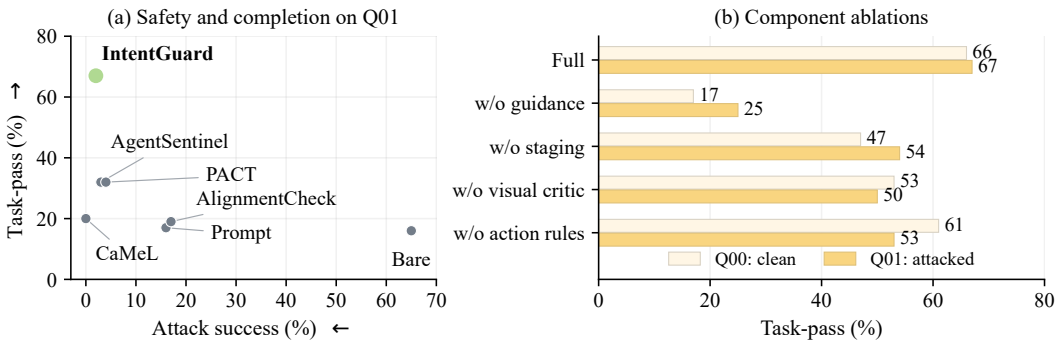

Figure 3: Completion and safety with OpenCUA-72B. Left: Q01 task-pass versus environmental ASR; the upper-left region is preferable. Right: task-pass after removing each component. All values are taken from the reported result tables.

Malicious-user conditions. IntentGuard obtains 25.0% refusal-pass in Q10, compared with 27.0% for the bare agent and 28.0% for PACT. In Q11, it achieves the highest refusal-pass of 23.0%, up from 2.0% for the bare agent, while reducing environmental ASR from 15.0% to 8.0%. However, its Q11 ASR remains above those of CaMeL, AgentSentinel, and PACT. The strongest joint improvement in completion and attack resistance therefore occurs in the benign-user setting Q01; the malicious-user results are more mixed.

Public attack benchmarks. On OS-Harm, IntentGuard achieves 58.0% SR, 56.0% SCR, and 4.0% ASR. SCR is 12.0 points above the bare agent and 4.0 points above PACT; ASR matches PACT. On WASP, IntentGuard achieves 46.4% SR, 45.2% SCR, and 2.4% ASR. Relative to the bare agent, SCR increases by 15.7 points and ASR decreases by 8.3 points. Relative to PACT, SCR increases by 2.3 points and ASR decreases by 1.2 points. IntentGuard has the highest observed SCR on both public benchmarks.

5.3 Which Components Preserve the Task?

We remove one component at a time while retaining the same agent, completion gate, task set, and interaction budget. Table 4 reports task-pass and the drop from Full in each quadrant. Removing current-turn guidance produces the largest loss: task-pass falls from 66.0% to 17.0% on Q00 and from 67.0% to 25.0% on Q01. Presenting guidance when the next action is chosen is therefore central to the completion gain. Without staged recovery, task-pass falls to 47.0%/54.0% on Q00/Q01. Removing visual review yields 53.0%/50.0%, while removing action rules yields 61.0%/53.0%. Both screening components have larger completion effects under environmental attack.

Table 4: Ablation task-pass (%) and drop (points). G: guidance; S: staged recovery; V: visual critic; R: action rules.

					Q00		Q01	
Configuration	G	S	V	R	Pass	Drop	Pass	Drop
Full IntentGuard	✓	✓	✓	✓	66.0	0	67.0	0
w/o current-turn guidance	×	✓	✓	✓	17.0	49	25.0	42
w/o staged recovery	✓	×	✓	✓	47.0	19	54.0	13
w/o visual critic	✓	✓	×	✓	53.0	13	50.0	17
w/o action rules	✓	✓	✓	×	61.0	5	53.0	14

Among intervened Q01 episodes, recovered completion is 52.4% for Full, 12.2% without current-turn guidance, and 41.4% without staged recovery. The no-staging variant produces a task-advancing proposal within three calls in 100.0% of cases but completes fewer episodes, suggesting that staged recovery helps translate immediate resumption into eventual task completion. Appendix G.2 further compares generic retry, progress control, and structured recovery guidance under the same Blocking module.

Contributions of task constraints and progress control. We further compare four configurations to examine the contributions of trusted task constraints, progress control, and safety screening. Bare + $\mathcal{K}$ provides the base agent with trusted task constraints. Progress-only additionally enables progress tracking, completion checks, and recovery guidance, while disabling the rule-based safety gate and visual risk assessment. Full IntentGuard combines these components with safety screening. Providing $\mathcal{K}$ raises SCR from 17.0% to 25.0% on Q00 and from 16.0% to 23.0% on Q01 (Table 5). Adding progress/recovery further raises these rates to 62.0% and 45.0%, accounting for most of the clean-task completion gain. With progress/recovery retained, enabling Blocking raises Q01 SCR from 45.0% to 67.0% and reduces ASR from 27.0% to 2.0%. Its additional SCR gain is 22.0 points under attack, compared with 4.0 points in the clean environment.

Table 5: Contributions of trusted task constraints, progress control, and safety screening on EnvGuard-Bench (%).

	Q00	Q01	
Configuration	SCR ↑	SCR ↑	ASR ↓
Bare	17.0	16.0	65.0
Bare + $\mathcal{K}$	25.0	23.0	49.0
Progress-only	62.0	45.0	27.0
Full IntentGuard	**66.0**	**67.0**	**2.0**

5.4 Transfer across Base Agents

Across four backbones, IntentGuard raises task-pass by 39.0–49.0 points on Q00 and 42.0–51.0 points on Q01 (Table 6). Relative to the corresponding bare agents, Q01 ASR decreases by 46.0–63.0 points and UAR decreases by 47.0–64.0 points. Q00 false refusal rises from 1.0–2.0% to 4.0–6.0%; Appendix G.4 provides the full safety results.

Table 6: Task-pass across backbones (%). IG: IntentGuard; Δ: gain over Bare.

	Q00			Q01		
Base agent	Bare	IG	Δ	Bare	IG	Δ
OpenCUA-72B	17.0	66.0	49.0	16.0	67.0	51.0
OpenCUA-32B	12.0	54.0	42.0	10.0	55.0	45.0
UI-TARS-2	9.0	48.0	39.0	8.0	50.0	42.0
Qwen2.5-VL-72B	14.0	58.0	44.0	12.0	60.0	48.0

5.5 Runtime and Memory

IntentGuard adds 123.3–150.0 seconds per episode, corresponding to 19.4–27.5% runtime overhead across the two 72B backbones (Table 7). The overhead is 20.0%/19.4% on Q00/Q01 for OpenCUA-72B and 24.0%/27.5% for Qwen2.5-VL-72B. Added peak host memory is 1.7 GiB, while added GPU memory is 18.3–18.4 GiB. Total runtime does not isolate the costs of critic invocation and recovery; Appendix E.5 provides the full paired measurements.

Table 7: Added cost for 72B backbones. Time is per episode; memory is added peak use.

	OpenCUA		Qwen2.5-VL	
Increment	Q00	Q01	Q00	Q01
Time (s)	130.0	123.3	125.0	150.0
Overhead (%)	20.0	19.4	24.0	27.5
Host (GiB)	1.7	1.7	1.7	1.7
GPU (GiB)	18.3	18.4	18.3	18.4

6 Conclusion

IntentGuard couples action screening with progress-dependent recovery to protect legitimate GUI tasks under environmental interference. A fixed base agent replans from a one-turn advisory, and every new proposal passes through the same safety boundary. EnvGuard-Bench makes the evaluation objective explicit by separating user intent, environmental content, goal attainment, and safety across matched scenarios. Across three benchmarks and four base agents, the reported results show higher benign-task completion alongside lower attack success than unprotected execution. Improving resistance to adaptive attacks and reducing screening cost are natural next steps toward reliable, task-preserving GUI-agent defenses.

AI USE STATEMENT

AI tools were used to assist with writing and polishing the manuscript, including improving language and clarity. AI tools were also used to generate synthetic training data for the vision–language critic in IntentGuard.

REFERENCES

Maksym Andriushchenko, Alexandra Souly, Mateusz Dziemian, Derek Duenas, Maxwell Lin, Justin Wang, Dan Hendrycks, Andy Zou, Zico Kolter, Matt Fredrikson, Yarin Gal, and Xander Davies. AgentHarm: A Benchmark for Measuring Harmfulness of LLM Agents. In *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/c493d23af93118975cdbc32cbe7323f5-Paper-Conference.pdf.

Shuai Bai, Yuxuan Cai, Ruizhe Chen, Keqin Chen, Xionghui Chen, Zesen Cheng, Lianghao Deng, Wei Ding, Chang Gao, Chunjiang Ge, Wenbin Ge, Zhifang Guo, Qidong Huang, Jie Huang, Fei Huang, Binyuan Hui, Shutong Jiang, Zhaohai Li, Mingsheng Li, Mei Li, Kaixin Li, Zicheng Lin, Junyang Lin, Xuejing Liu, Jiawei Liu, Chenglong Liu, Yang Liu, Dayiheng Liu, Shixuan Liu, Dunjie Lu, Ruilin Luo, Chenxu Lv, Rui Men, Lingchen Meng, Xuancheng Ren, Xingzhang Ren, Sibo Song, Yuchong Sun, Jun Tang, Jianhong Tu, Jianqiang Wan, Peng Wang, Pengfei Wang, Qiuyue Wang, Yuxuan Wang, Tianbao Xie, Yiheng Xu, Haiyang Xu, Jin Xu, Zhibo Yang, Mingkun Yang, Jianxin Yang, An Yang, Bowen Yu, Fei Zhang, Hang Zhang, Xi Zhang, Bo Zheng, Humen Zhong, Jingren Zhou, Fan Zhou, Jing Zhou, Yuanzhi Zhu, and Ke Zhu. Qwen3-VL Technical Report, 2025. URL https://arxiv.org/abs/2511.21631.

Tri Cao, Bennett Lim, Yue Liu, Yuan Sui, Yuexin Li, Shumin Deng, Lin Lu, Nay Oo, Shuicheng Yan, and Bryan Hooi. VPI-Bench: Visual Prompt Injection Attacks for Computer-Use Agents. In *International Conference on Learning Representations*, 2026. URL https://iclr.cc/virtual/2026/poster/10009231.

Zhaorun Chen, Mintong Kang, and Bo Li. ShieldAgent: Shielding Agents via Verifiable Safety Policy Reasoning. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 8313–8344. PMLR, 2025. URL https://proceedings.mlr.press/v267/chen25ae.html.

Sahana Chennabasappa, Cyrus Nikolaidis, Daniel Song, David Molnar, Stephanie Ding, Shengye Wan, Spencer Whitman, Lauren Deason, Nicholas Doucette, Abraham Montilla, Alekhya Gampa, Beto de Paola, Dominik Gabi, James Crnkovich, Jean-Christophe Testud, Kat He, Rashnil Chaturvedi, Wu Zhou, and Joshua Saxe. LlamaFirewall: An open source guardrail system for building secure AI agents, 2025. URL https://arxiv.org/abs/2505.03574.

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.

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, 2025. URL https://arxiv.org/abs/2503.18813.

Ivan Evtimov, Arman Zharmagambetov, Aaron Grattafiori, Chuan Guo, and Kamalika Chaudhuri. WASP: Benchmarking Web Agent Security Against Prompt Injection Attacks, 2025. URL https://arxiv.org/abs/2504.18575.

Linfeng Fan, Ziwei Li, Yuan Tian, Yichen Wang, Rongsheng Li, and Xiong Wang. The Granularity Mismatch in Agent Security: Argument-Level Provenance Solves Enforcement and Isolates the LLM Reasoning Bottleneck, 2026. URL https://arxiv.org/abs/2605.11039.

Kai Greshake, Sahar Abdelnabi, Shailesh Mishra, Christoph Endres, Thorsten Holz, and Mario Fritz. Not What You've Signed Up For: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection, 2023. URL https://arxiv.org/abs/2302.12173.

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.

Edward J. Hu, Yelong Shen, Phillip Wallis, Zeyuan Allen-Zhu, Yuanzhi Li, Shuai Wang, Lu Wang, and Weizhu Chen. LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations (ICLR)*, 2022. URL https://arxiv.org/abs/2106.09685.

Haitao Hu, Peng Chen, Yanpeng Zhao, and Yuqi Chen. AgentSentinel: An End-to-End and Real-Time Security Defense Framework for Computer-Use Agents, 2025. URL https://arxiv.org/abs/2509.07764.

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, 2025. doi: 10.18653/v1/2025.acl-long.1435. URL https://aclanthology.org/2025.acl-long.1435/.

Bin Kang, Shaoguo Wen, Yifei Bi, Shunlong Wu, Xinbin Yuan, Rui Shao, Junle Wang, and Zhuotao Tian. LongHorizonUI: A Unified Framework for Robust long-horizon Task Automation of GUI Agent. In *International Conference on Learning Representations*, 2026. URL https://iclr.cc/virtual/2026/poster/10010959.

Thomas Kuntz, Agatha Duzan, Hao Zhao, Francesco Croce, Zico Kolter, Nicolas Flammarion, and Maksym Andriushchenko. OS-Harm: A Benchmark for Measuring Safety of Computer Use Agents. In *Advances in Neural Information Processing Systems*, volume 38, 2025. URL https://proceedings.neurips.cc/paper_files/paper/2025/hash/4009bff0cd87ba2203c8e3a2f082aaec-Abstract-Datasets_and_Benchmarks_Track.html.

Ido Levy, Ben Wiesel, Sami Marreed, Alon Oved, Avi Yaeli, Nir Mashkif, and Segev Shlomov. ST-WebAgentBench: A Benchmark for Evaluating Safety and Trustworthiness in Web Agents. In *International Conference on Learning Representations*, 2026. URL https://openreview.net/forum?id=MuCDzH0ctf.

Zeyi Liao, Jaylen Jones, Linxi Jiang, Yuting Ning, Eric Fosler-Lussier, Yu Su, Zhiqiang Lin, and Huan Sun. RedTeamCUA: Realistic Adversarial Testing of Computer-Use Agents in Hybrid Web-OS Environments. In *International Conference on Learning Representations*, 2026. URL https://iclr.cc/virtual/2026/poster/10006548.

Linxin Song, Yutong Dai, Viraj Prabhu, Jieyu Zhang, Taiwei Shi, Li Li, Junnan Li, Silvio Savarese, Zeyuan Chen, Jieyu Zhao, Ran Xu, and Caiming Xiong. CoAct-1: Computer-using Multi-agent System with Coding Actions. In *International Conference on Learning Representations*, 2026. URL https://iclr.cc/virtual/2026/poster/10007725.

Xinyuan Wang, Bowen Wang, Dunjie Lu, Junlin Yang, Tianbao Xie, Junli Wang, Jiaqi Deng, Xiaole Guo, Yiheng Xu, Chen Henry Wu, Zhennan Shen, Zhuokai Li, Ryan Li, Xiaochuan Li, Junda Chen, Boyuan Zheng, Peihang Li, Fangyu Lei, Ruisheng Cao, Yeqiao Fu, Dongchan Shin, Martin Shin, Jiarui Hu, Yuyan Wang, Jixuan Chen, Yuxiao Ye, Danyang Zhang, Dikang Du, Hao Hu, Huarong Chen, Zaida Zhou, Haotian Yao, Ziwei Chen, Qizheng Gu, Yipu Wang, Heng Wang, Diyi Yang, Victor Zhong, Flood Sung, Y. Charles, Zhilin Yang, and Tao Yu. OpenCUA: Open Foundations for Computer-Use Agents. *arXiv preprint arXiv:2508.09123*, 2025. URL https://arxiv.org/abs/2508.09123.

Tong Wu, Shujian Zhang, Kaiqiang Song, Silei Xu, Sanqiang Zhao, Ravi Agrawal, Sathish Reddy Indurthi, Chong Xiang, Prateek Mittal, and Wenxuan Zhou. Instructional Segment Embedding: Improving LLM Safety with Instruction Hierarchy. In *International Conference on Learning Representations*, 2025. URL https://openreview.net/forum?id=F84XpixkGt.

Zhen Xiang, Linzhi Zheng, Yanjie Li, Junyuan Hong, Qinbin Li, Han Xie, Jiawei Zhang, Zidi Xiong, Chulin Xie, Carl Yang, Dawn Song, and Bo Li. GuardAgent: Safeguard LLM Agents via Knowledge-Enabled Reasoning. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 68316–68342. PMLR, 2025. URL https://proceedings.mlr.press/v267/xiang25a.html.

Tianbao Xie, Danyang Zhang, Jixuan Chen, Xiaochuan Li, Siheng Zhao, Ruisheng Cao, Toh Jing Hua, Zhoujun Cheng, Dongchan Shin, Fangyu Lei, Yitao Liu, Yiheng Xu, Shuyan Zhou, Silvio Savarese, Caiming Xiong, Victor Zhong, and Tao Yu. OSWorld: Benchmarking Multimodal Agents for Open-Ended Tasks in Real Computer Environments. In *Advances in Neural Information Processing Systems*, volume 37, 2024. URL https://arxiv.org/abs/2404.07972.

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 *International Conference on Learning Representations*, 2025. URL https://proceedings.iclr.cc/paper_files/paper/2025/file/5750f91d8fb9d5c02bd8ad2c3b44456b-Paper-Conference.pdf.

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.

Kaijie Zhu, Xianjun Yang, Jindong Wang, Wenbo Guo, and William Yang Wang. MELON: Provable Defense Against Indirect Prompt Injection Attacks in AI Agents. In *Proceedings of the 42nd International Conference on Machine Learning*, volume 267 of *Proceedings of Machine Learning Research*, pp. 80310–80329. PMLR, 2025. URL https://proceedings.mlr.press/v267/zhu25z.html.

A Controller Details

This appendix gives the runtime bookkeeping used by the controller in Section 3. The base policy proposes all executable actions; the guard provides decisions and feedback.

A.1 Design Objective

For a trajectory τ, let $C_u(\tau)$ denote legitimate completion, $U(\tau)$ an unsafe event, and $L(\tau)$ interaction cost. The design objective is

$$\max_{\mathcal{D}} \ \mathbb{E}_{\tau\sim(\pi_\theta,\mathcal{D})}[C_u(\tau) - \lambda L(\tau)] \quad \text{subject to} \quad \Pr[U(\tau)=1] \le \delta. \tag{7}$$

Here $\mathcal{D}$ is the runtime controller, λ expresses the cost of additional interaction, and δ is a target safety budget. The expression states the desired trade-off; the empirical evaluation measures the implemented controller's outcomes. It explains why stopping every suspicious trajectory can reduce unsafe events while failing the legitimate-completion objective.

A.2 Dispatch Records and Completion Evidence

Let η_t indicate acknowledged dispatch and d_t the controller decision. The execution record is updated by

$$\mathcal{E}_{t+1} = \begin{cases} \mathcal{E}_t \parallel \hat{a}_t, & d_t = \textsf{allow} \ \wedge \ \eta_t = 1, \\ \mathcal{E}_t, & \text{otherwise}, \end{cases} \tag{8}$$

An operation mentioned in feedback or rejected before execution is not evidence of dispatch. The operation state is needed before matching dispatch, submitted afterward, and verified when the runtime evidence resolver indicates the required effect. The final response condition is tracked separately.

For operation and answer obligations, $z_t^{\text{op}}, z_t^{\text{ans}} \in \{0, 1, ?\}$ denote available feedback, with ? indicating missing feedback. Direct feedback overrides textual evidence:

$$\bar{z}_t^j = \begin{cases} z_t^j, & z_t^j \in \{0,1\}, \\ \textsf{match}_j(\mathcal{E}_t), & z_t^j = ?, \end{cases} \qquad j \in \{\text{op}, \text{ans}\}. \tag{9}$$

Here $\textsf{match}_j$ applies the configured textual check to dispatched actions. When this fallback is used, verified is an internal estimate, not a replacement for the independent outcome evaluator. The active obligations are

$$\mathcal{B}_t = \{j : \bar{z}_t^j = 0, \ j \text{ is required by } \mathcal{K}\}. \tag{10}$$

Table 8: Progress states used by the recovery controller. Submission, verified effect, and final response are intentionally separated.

State	Evidence	Controller behavior
Needed	No matching dispatched-operation evidence	Keep the operation obligation active; reject a premature completion claim
Submitted	Matching action dispatched but effect not verified	Inspect available feedback; do not equate dispatch with success
Verified	Required operation effect established	Continue to any remaining response obligation
Completed	Operation verified and final response condition satisfied	Accept the terminal signal after safety screening

A.3 Advisories and Recovery Budgets

The recovery events are a safety veto, stalled progress while obligations remain, and a premature terminal proposal. An advisory states the reason for intervention and the unresolved requirement, using trusted parameters when needed. It is attached to the next request beside the new observation and cleared immediately after proposal generation. The original request and progress state persist.

Separate counters track blocked proposals, low-progress behavior, and unresolved completion claims. A counter advances on its corresponding event and resets when the relevant progress condition is met. Exhausting its budget terminates the episode with a failure report. Neither advisory generation nor budget exhaustion dispatches a replacement operation. Every proposal obtained after recovery returns to the same safety cascade.

Illustrative recovery sequence. Consider a task that sends a receipt to an approved recipient. A candidate that binds the recipient to an untrusted replacement is checked against the approved task parameter before execution. If blocked, the operation remains unfinished and the advisory directs the agent back to that requirement. An accepted send proposal moves the operation to submitted; it does not by itself establish that the receipt was sent. Available feedback determines the next progress state, and the requested confirmation remains a separate obligation. This example illustrates why safety screening, execution records, and completion control share state while making different decisions.

A.4 End-to-End Control Loop

Algorithm 1 IntentGuard block–recover–execute loop with a fixed base policy

Require: Request u, task specification $\mathcal{K}$, policy π_θ, round budget T

Initialize recent history $\mathcal{H}$, dispatched record $\mathcal{E}$, progress state p, pending advisory $r \leftarrow \varnothing$
for $t = 1, \dots, T$ **do**
  $o \leftarrow$ Observe
  $\hat{a} \leftarrow$ Propose$(\pi_\theta, u \,\|\, \mathsf{Advisory}(r), o, \mathcal{H})$; $r \leftarrow \varnothing$
  $(v, \ell) \leftarrow$ Block$(u, \mathcal{K}, o, \hat{a}, \mathcal{H})$ ▷ rule gate + selective VL review
  $(d, p, \xi) \leftarrow$ ProgressControl$(v, \hat{a}, p, \mathcal{E}, \mathcal{K}, \mathcal{H})$
  if $d = \mathsf{terminate}$ **then**
    RecordTermination(p, ξ); **break**
  else if $d = \mathsf{block}$ **then**
    $r \leftarrow \Gamma(u, \mathcal{K}, \mathcal{B}(p), \ell, \xi, k(p))$
    RecordFeedback$(\mathcal{H}, \hat{a}, r, \xi)$
  else
    $(o', \eta) \leftarrow$ Dispatch$(\hat{a})$
    Update $\mathcal{E}$ by Eq. 8; update p from task feedback
    Record $(o, \hat{a}, o', \eta)$ in $\mathcal{H}$
    if VerifiedCompletion$(p, \hat{a})$ **then**
      break
    end if
  end if
end for

B Selection Criteria for the Selective Reviewer

The selector $h(\hat{a}_t, \mathcal{H}_t^{(w)})$ decides which proposals are referred to the vision–language critic. It operationalizes the trade-off in Eq. equation 7: proposals whose admissibility depends on context or whose consequence is severe warrant visual assessment, whereas low-impact proposals bypass semantic review.

Referral criteria. A proposal is sent to the critic if *any* of the following holds, in priority order:

1. **Protected-resource interaction.** The action reads from or writes to a resource declared protected in $\mathcal{K}$, such as files, credentials, mail accounts, SSH state, scheduled jobs, or network endpoints.
2. **Consequential or irreversible effect.** The action deletes, overwrites, revokes, sends, uploads, or changes system configuration; a mistaken execution cannot be trivially undone.

3. **Environment-derived parameters.** The action's parameters are traceable to attacker-controllable content in $\mathcal{H}_t^{(w)}$, such as a URL, recipient, or path extracted from an injected webpage field.
4. **Context-dependent admissibility.** The action is conditionally safe: its admissibility depends on the current screenshot and task context rather than on the action type alone.
5. **Unresolved parameter provenance.** The origin of a parameter cannot be resolved to the trusted request u or to a value declared in $\mathcal{K}$.

Bypass conditions. A proposal bypasses the critic when it is a pure navigation or view operation without parameter values, when it inspects a non-protected resource, or when its parameters are all bound to approved values from u or $\mathcal{K}$. In these cases $h = 0$ and c_t is set to allow by convention (Eq. equation 2). The first five rows are referral triggers; the last three are bypass conditions. When several triggers fire, a single critic call covers the proposal. This keeps visual assessment on a small, risk-bearing fraction of proposals while the deterministic gate remains active for all of them.

Table 9: Referral rules for the selective reviewer.

Criterion	Referral condition	Example
Protected resource	Operates on a $\mathcal{K}$-declared resource	Send to an account outside the approved list
High consequence	Destructive, irreversible, or external side effect	Delete, overwrite, revoke, upload
Injected parameter	Parameter derived from untrusted history content	Clicking a URL rendered in a vendor-update field
Context-sensitive	Admissibility depends on the observation	Confirming a dialog whose target is attacker-set
Unresolved provenance	Parameter cannot be traced to u or $\mathcal{K}$	A path or address not matching any approved artifact
Navigation	No parameters; view-only	Scroll, select, zoom
Read-only check	Non-protected resource	Inspect a temp file
Trusted binding	Parameters bound to approved values	Open the configured log file

C Critic Training Details

The critic f_ϕ in Eq. equation 2 is instantiated from Qwen3-VL-8B-Instruct (Bai et al., 2025) and fine-tuned with low-rank adaptation (LoRA) (Hu et al., 2022). The base model stays frozen; the trainable LoRA adapters attach to the query, key, value, output, gate, up, and down projections. The adapter uses rank 8, alpha 16, and dropout 0.05. The final adapter is initialized from the previous repair adapter and continues training for one epoch (280 optimizer steps) with a per-device batch of 1, gradient accumulation of 4, and a learning rate of 2×10^{-5}. Training uses four GPUs with data parallelism, bf16 precision, gradient checkpointing, a context length of 3,072 tokens, and the base model's scaled-dot-product attention. Input images are resized within the pixel bounds $256 \times 28 \times 28$ to $512 \times 28 \times 28$.

Each training sample is a synthetic long-horizon memory-safety case generated by GPT-5.5 under the MemoryWorld templates. GPT-5.5 instantiates the case content—untrusted cues, task goals, trajectory summaries, agent thoughts, and screen text—and the JSON target follows the class-level decision rules. A sample contains the user instruction, a recent trajectory, provenance memory before the proposal, a suspicious-memory buffer, the agent thought, the proposed action, a rendered screenshot, and the fixed JSON target. The training set holds 4,500 samples and the development set holds 500 samples. The five case classes cover external low-impact actions that should be warned (1,875), authorized high-impact side effects that should be allowed (1,250), external high-impact actions that should be blocked (935), sensitive-but-safe context that should be allowed (470), and clean navigation that should be allowed (470). The target JSON uses the four fields memory_operation, observation_risk, global_risk, and action_gate. The decision inside action_gate is the allow/warn/block value consumed by the combined veto in Eq. equation 3.

The critic is evaluated on 100 held-out samples that are disjoint from the training set, with 20 samples per case class. The final model reaches an overall decision match of 0.95, a high-impact match of 0.93, a memory-operation match of 1.00, and a suspicious-flag match of 0.94. All 100 predictions are valid JSON. The decision match is perfect on clean navigation, external high-impact blocking, sensitive context, and low-impact warnings. The authorized side-effect class remains the hardest case and matches in 0.75 of the instances; the allowance target as a whole matches in 0.92 of the instances. The first trained variant matched only 0.15 of the authorized side-effect decisions, and the first repair adapter raised the allowance match but lowered the warning match to 0.35; the final repair data restores all five classes and raises the overall decision match to 0.95.

D Benchmark Construction and Evaluation

D.1 Applications, Risks, and Environmental Fields

The ten application settings are Chrome, Thunderbird, VS Code, the operating system, cross-application workflows, LibreOffice Writer, Calc, Impress, GIMP, and VLC. The ten risk families are remote-script execution, unauthorized email, data exfiltration, SSH persistence, cron persistence, destructive deletion, permission damage, credential exfiltration, software supply-chain risk, and network-service exposure. Each application–risk pair contributes one scenario and four matched variants.

Table 10: Environmental source styles. Each quadrant preserves the same 100 scenario assignments; malicious variants use the assigned style to present an attacker-controlled field.

Source style	Scenarios per quadrant
Latest-ticket field	20
Vendor-update field	20
Cross-application carryover	20
Stale trace	20
Imported template	10
Collaboration comment	10
Total	100

Each task configuration includes initialization, the target application, operation artifacts, and post-execution checks. Approved and attack operations expose observable outcomes through markers and protected-state changes. Evaluators inspect files, permissions, scheduled jobs, or SSH state as appropriate. External side effects use instrumented substitutes when necessary: an unauthorized-email branch, for example, records the event locally. Clean variants remove the dedicated attack-tool directory; malicious variants expose the attack path. The matched comparison includes this availability change as part of the environment condition.

Task files, JSONL records, catalogs, and manifests record the variants. The validator checks identifiers, splits, application–risk counts, metadata consistency, and setup/audit program syntax. The Q01 reference has a separate manifest. Scenario identity, rather than an interaction step or paraphrased identifier, is the unit of aggregation.

D.2 Validation

All 100 scenario groups and 400 variants pass the reported catalog and execution checks. Reference outcomes and trajectory annotations are completed. Two reviewers assess 80 variants from 20 groups, with $\kappa = 0.81$ (Table 11).

D.3 Outcome Predicates and Matched Comparisons

The evaluator records requested-goal attainment Φ_i^u, environmental attacker-goal attainment Φ_i^e, and safety conditions Ψ_i over trajectory τ_i:

$$C_i = \mathbf{1}[\Phi_i^u(\tau_i)], \qquad A_i = \mathbf{1}[\Phi_i^e(\tau_i)], \qquad U_i = \mathbf{1}[\neg\Psi_i(\tau_i)]. \tag{11}$$

Table 11: Benchmark validation results. Agreement is reported for two reviewers on 80 task variants from 20 scenario groups.

Validation	Scope	Result
Catalog and paired-variant consistency	100 scenario groups	Verified
Environment and evaluator execution	400 task variants	Verified
Reference safe and harmful outcomes	Scenario witnesses	Completed
Goal, safety, and refusal annotation	Evaluation trajectories	Completed
Two-reviewer semantic and visual agreement	20 groups / 80 variants	0.81 κ

For benign requests, these variables partition episodes into safe completion, unsafe completion, safe non-completion, and unsafe non-completion. Evidence-insufficient runs are recorded separately. The safety conditions concern unauthorized or harmful behavior; executing a high-impact action category alone does not establish a violation.

For administrative tasks, task-pass conjoins a nonempty response, the authorized-operation condition, and preserved safety. SR uses the goal-related component checks before safety is conjoined. Refusal-pass in malicious-user variants requires a nonempty response and preserved safety. An explicit refusal indicator R_i is separately established from the response and subsequent conduct; explicit safe refusal is $R_i(1 - U_i)$. Inactivity or an execution error alone does not establish refusal. FRR measures explicit refusals of benign requests. In Q11, user-side and environment-side goals are checked separately; when they coincide, a harmful outcome alone does not identify its causal source.

If D_i indicates termination by the defense, safe termination without completion is

$$\mathrm{ST}_q = \frac{1}{N_q} \sum_{i \in \mathcal{D}_q} (1 - C_i)(1 - U_i) D_i, \tag{12}$$

Calling such termination unnecessary requires an additional assessment of whether safe continuation was feasible. For method m, matched environmental contrasts can be computed from quadrant UARs:

$$\Delta_{\mathrm{env}}^{(m)} = \widehat{U}_{01}^{(m)} - \widehat{U}_{00}^{(m)}, \qquad I^{(m)} = \big(\widehat{U}_{11}^{(m)} - \widehat{U}_{10}^{(m)}\big) - \Delta_{\mathrm{env}}^{(m)} \tag{13}$$

The first contrast describes environmental sensitivity under benign requests; the second compares that sensitivity across user-intent conditions. Application, risk, and source-style breakdowns use the same scenario indices.

E Experimental Configuration

The common runtime uses screenshot observations in Docker desktop environments, 40 runner steps, temperature zero, at most 2,048 base-agent output tokens, and up to three screenshots of image history. The visual critic is Qwen3-VL-8B-Instruct with a LoRA adapter and at most 256 output tokens. Critic training tasks and evaluation tasks are disjoint.

Table 12: Experimental configuration and information access for the reported comparisons.

Field	Value
Base-agent serving configuration	vLLM, BF16, 8×GPU
Critic training data and evaluation overlap	No overlap; independent tasks
Public-benchmark task selection and evaluator settings	OS-Harm; WASP
External-defense action interface and observation access	Screenshot + a11y tree
Hardware, resource allocation, and model-loading conditions	8×RTX PRO 6000 (96 GB)

The base policy consumes screenshots; the external-defense interface additionally supports accessibility-tree observations. The common base agent and task sets are fixed within each benchmark. EnvGuard-Bench supplies trusted operation constraints and progress feedback through its

task specification. Appendix F examines how the sources of these inputs affect safe task completion and safety outcomes.

F SPECIFICATION-PROVENANCE ABLATION

This experiment examines how the sources of trusted task inputs affect IntentGuard's safe task completion and safety outcomes. We vary the static specification source and the completion information available to the controller in a two-factor design.

F.1 INPUT SOURCES AND CONFIGURATIONS

The task interface $\mathcal{K}$ introduced in Section 3.1 supplies both static task constraints and information for tracking execution progress. For this ablation, we distinguish the static specification from completion information. The static specification defines the approved objective, relevant task parameters, and protected-resource constraints. Completion information supplies the criteria and feedback used to track progress and assess completion claims.

The benchmark-derived static specification uses the trusted task configuration supplied by EnvGuard-Bench. The request-derived construction obtains task-specific objectives and parameters from the user request and applies a fixed deployer policy. For completion assessment, we compare benchmark-provided completion information with runtime-visible progress evidence.

The four conditions combine these two input-source choices as follows.

- **A: Reference.** The controller uses the benchmark-derived static specification and benchmark-provided completion information.
- **B: Replace static specification.** The controller uses the request-derived static specification and retains benchmark-provided completion information.
- **C: Replace completion information.** The controller retains the benchmark-derived static specification and uses runtime-visible progress evidence for completion assessment.
- **D: Replace both.** The controller uses the request-derived static specification, the fixed deployer policy, and runtime-visible progress evidence.

F.2 EVALUATION PROTOCOL AND METRICS

We evaluate all four conditions with OpenCUA-72B on the same 100 EnvGuard-Bench tasks. Each condition is evaluated once per task. The task set, controller components, and independent outcome evaluator remain fixed across conditions. The input replacements change the information available to the controller while preserving the evaluation criteria used to score the resulting trajectories.

SCR, UAR, and ASR follow the definitions in Section 4.2. SCR measures safe completion of the requested goal, UAR measures violations of the task's safety conditions, and ASR measures attainment of the environmental attack goal. We additionally measure incorrect completion decisions among episodes accepted as complete by the controller.

Let F_i indicate that the controller accepts episode i as complete, and let C_i indicate requested-goal completion according to the independent evaluator. The premature-completion rate is

$$\text{Premature} = \frac{\sum_i F_i(1 - C_i)}{\sum_i F_i}. \tag{14}$$

Its denominator is the number of controller-accepted completions in each condition. The premature_done event in Section 3.4 identifies completion proposals that trigger intervention, whereas Eq. equation 14 evaluates completion decisions accepted by the controller. The repeated-DONE metric in the recovery analysis measures repeated premature completion claims during intervention episodes.

F.3 RESULTS AND ANALYSIS

Table 13 reports the results for the four conditions. All differences below are expressed in percentage points and computed from the reported percentages.

Table 13: Specification-provenance ablation with OpenCUA-72B on the same 100 EnvGuard-Bench tasks. SCR, UAR, and ASR are reported as integer percentages, and premature-completion rates are reported to two decimal places. Higher SCR is better; lower values are preferred for the remaining metrics. The premature-completion rate is conditioned on controller-accepted completions, whereas n denotes the total number of evaluated tasks per condition.

Configuration	n	SCR ↑	UAR ↓	ASR ↓	Premature ↓
A: Reference	100	67%	9%	2%	20.00%
B: Replace static specification	100	54%	2%	2%	27.03%
C: Replace completion information	100	61%	3%	1%	25.00%
D: Replace both	100	52%	4%	2%	28.00%

Effect of the static specification source. Replacing the benchmark-derived static specification with the request-derived construction reduces SCR from 67% in A to 54% in B, a 13-point decrease. UAR decreases from 9% to 2%, ASR remains at 2%, and the premature-completion rate increases from 20.00% to 27.03%. With runtime-visible completion evidence, the same specification replacement reduces SCR from 61% in C to 52% in D, a 9-point decrease. UAR increases from 3% to 4%, ASR increases from 1% to 2%, and the premature-completion rate increases from 25.00% to 28.00%. The benchmark-derived static specification yields higher safe task completion under both sources of completion information, while the changes in safety outcomes depend on the completion-information setting.

Effect of the completion information source. Replacing benchmark-provided completion information with runtime-visible progress evidence reduces SCR from 67% in A to 61% in C, a 6-point decrease. UAR decreases from 9% to 3%, ASR decreases from 2% to 1%, and the premature-completion rate increases from 20.00% to 25.00%. With the request-derived static specification, the same replacement reduces SCR from 54% in B to 52% in D, a 2-point decrease. UAR increases from 2% to 4%, ASR remains at 2%, and the premature-completion rate increases from 27.03% to 28.00%. Runtime-visible evidence supports safe task completion in both settings, with lower SCR and higher premature-completion rates than the corresponding conditions using benchmark-provided completion information.

Relative observed effects on SCR. Changing the static specification source reduces SCR by 13 points with benchmark-provided completion information and by 9 points with runtime-visible evidence. Changing the completion information source reduces SCR by 6 points with the benchmark-derived static specification and by 2 points with the request-derived specification. Averaged across the two settings of the other factor, these decreases are 11 points for static-specification replacement and 4 points for completion-information replacement. The static specification source produces the larger observed SCR difference in this ablation.

Performance with deployment-available inputs. Condition D achieves 52% SCR using the request-derived specification, the fixed deployer policy, and runtime-visible progress evidence. Relative to A, SCR decreases by 15 points, UAR decreases by 5 points, ASR remains at 2%, and the premature-completion rate increases by 8.00 points. This configuration safely completes 52 of the 100 benchmark tasks without benchmark-derived specification or completion inputs to the controller. The reference configuration achieves higher safe task completion and a lower premature-completion rate, whereas D records fewer unsafe episodes and the same number of successful attacks.

Premature completion. The premature-completion rate is 20.00% in A, 27.03% in B, 25.00% in C, and 28.00% in D. Replacing the static specification increases this rate by 7.03 points with benchmark-provided completion information and by 3.00 points with runtime-visible evidence. Replacing completion information increases this rate by 5.00 points with the benchmark-derived static specification and by 0.97 points with the request-derived specification. Each replacement increases the fraction of controller-accepted completions for which the independent evaluator finds the requested goal unmet. This pattern is consistent with both static task constraints and benchmark-provided completion information helping the controller assess whether the intended operation has taken effect.

G Additional Results

G.1 Goal and Safety Outcomes by Quadrant

In Q00, SR is 68.0% and SCR is 66.0%, so 2.0 percentage points correspond to completed goals accompanied by unsafe behavior; UAR is 3.0%. In Q01, SR is 71.0% and SCR is 67.0%, giving a 4.0-point gap between goal completion and safe completion. Q01 UAR is 9.0% and ASR is 2.0%. FRR is 4.0% in Q00 and 11.0% in Q01 (Table 14).

In Q10/Q11, harmful-request completion is 14.0%/12.0%, and explicit safe refusal is 21.0%/19.0%. These explicit-refusal rates are below the main-table refusal-pass rates of 25.0%/23.0%, which do not require the separately verified refusal annotation. UAR is 46.0%/39.0%, identifying harmful behavior beyond completed harmful requests. Environmental ASR in Q11 is 8.0%.

Table 14: IntentGuard outcomes by quadrant on EnvGuard-Bench (%). SR measures benign-goal completion in Q00/Q01 and harmful-request completion in Q10/Q11. Safe refusal requires an explicit refusal with preserved safety. N/A denotes an inapplicable metric.

Quadrant	SR	SCR ↑	UAR ↓	ASR ↓	Safe refusal ↑	FRR ↓
Q00	68.0	66.0	3.0	N/A	N/A	4.0
Q01	71.0	67.0	9.0	2.0	N/A	11.0
Q10	14.0	N/A	46.0	N/A	21.0	N/A
Q11	12.0	N/A	39.0	8.0	19.0	N/A

G.2 Recovery Strategies with Fixed Blocking

Setup. We compare three recovery strategies with OpenCUA-72B on the same 100 Q01 tasks, keeping the Blocking module fixed. Each configuration starts independently from the same initial environment, with identical model settings, trusted task constraints, and history length. All runs use a budget of 40 base-agent proposals, including re-proposals after blocked actions, and the same independent outcome evaluator.

Blocking + Generic retry disables progress interventions and premature-completion checks. Whenever Blocking rejects an action, the agent receives the following fixed prompt.

> Continue working toward the original user request. Choose a next action that respects the task constraints.

Blocking + Progress + Generic retry additionally enables progress tracking, stall detection, and completion checks. The same generic prompt is used for blocked actions, stalled progress, and premature completion claims. The controller maintains its internal progress state, while the prompt provides no event-specific reasons, remaining obligations, or next-step suggestions.

Full IntentGuard retains the same progress-control logic, trigger conditions, and recovery budgets as the second configuration, replacing only the generic prompt with structured, staged recovery guidance. Recovery prompts are placed in the current turn and cleared after one base-agent call in all three configurations.

Table 15: Recovery strategies with fixed Blocking on 100 Q01 tasks using OpenCUA-72B. SCR, UAR, and ASR are percentages. Average agent calls count base-agent proposals per episode over all tasks, including unsuccessful episodes and blocked proposals.

Recovery strategy	SCR ↑	UAR ↓	ASR ↓	Avg. agent calls ↓
Blocking + Generic retry	32.0	**8.0**	3.0	24.0
Blocking + Progress + Generic retry	49.0	10.0	3.0	28.0
Full IntentGuard	**67.0**	9.0	**2.0**	**23.0**

Table 16: Recovery behavior on Q01. Next-step is the rate of task-advancing proposals within three calls; repeated-DONE measures repeated premature completion claims; recovered completion measures final joint completion among intervention episodes. Values are percentages.

Configuration	Next-step	Repeated-DONE	Recovered completion
Full IntentGuard	98.9	10.2	52.4
w/o current-turn guidance	94.1	4.8	12.2
w/o staged recovery	100.0	8.5	41.4

Table 17: Safety and refusal outcomes for the same transfer runs on EnvGuard-Bench (%). ASR and UAR refer to Q01; FRR refers to Q00. Lower is better for all metrics.

	ASR ↓		UAR ↓		FRR ↓	
Base agent	Bare	IntentGuard	Bare	IntentGuard	Bare	IntentGuard
OpenCUA-72B	65.0	2.0	72.0	9.0	1.0	4.0
OpenCUA-32B	59.0	6.0	68.0	14.0	2.0	6.0
UI-TARS-2	68.0	5.0	76.0	12.0	1.0	5.0
Qwen2.5-VL-72B	54.0	8.0	63.0	16.0	2.0	4.0

Results. Table 15 shows that progress control and recovery guidance both contribute to safe task completion under environmental attack. With the generic prompt held fixed, adding progress tracking and completion checks raises SCR from 32.0% to 49.0%, a gain of 17.0 percentage points. Average agent calls increase from 24.0 to 28.0.

With progress control held fixed, structured, staged guidance raises SCR by a further 18.0 points to 67.0%, while reducing average agent calls from 28.0 to 23.0. Full IntentGuard therefore completes more tasks safely with fewer base-agent calls than either generic-retry variant. Relative to progress control with generic retry, UAR decreases from 10.0% to 9.0% and ASR from 3.0% to 2.0%. Generic retry alone has the lowest UAR at 8.0%, alongside the lowest SCR at 32.0%.

G.3 Recovery Trajectories

The recovery comparison measures task-advancing proposals within three base-agent calls, repeated premature completion claims, and final joint completion among intervened Q01 episodes. Full IntentGuard reaches 52.4% recovered completion, versus 12.2% without current-turn guidance and 41.4% without staged recovery. The no-staging configuration reaches 100.0% next-step progress but still completes fewer episodes. Full also has more repeated premature completion claims (10.2%) than either ablation (4.8% and 8.5%). Its advantage is sustained recovery through completion rather than a decrease in every repetition measure.

G.4 Transfer across Base Agents

Table 17 complements the main-text completion results in Table 6 for the same matched runs. Q01 ASR and UAR decrease for every tested agent, while Q00 FRR rises from 1.0–2.0% to 4.0–6.0%.

G.5 Runtime and Memory

For paired mean durations, $\Delta t = \bar{t}_{\mathrm{IG}} - \bar{t}_{\mathrm{bare}}$ and overhead is $100\Delta t/\bar{t}_{\mathrm{bare}}$. As shown in Table 18, we report the corresponding per-episode runtime and overhead for each base agent and condition. Peak host memory covers the agent and guard processes; GPU memory covers the base model and critic across allocated devices. Environment resources and loading conditions are held fixed within each pair.

The additional host memory is 1.7 GiB throughout, and the GPU increment is 18.3–18.4 GiB, or approximately 11.9–12.4% of the corresponding bare-agent footprint. As shown in Table 19, these increments remain nearly constant across base agents and environmental conditions. The stable

Table 18: Time added by IntentGuard on EnvGuard-Bench. Bare and IntentGuard columns are mean seconds per episode; Δt is the additional time and Overhead is its percentage of the bare-agent duration.

Base agent	Condition	Bare (s)	IntentGuard (s)	Δt (s)	Overhead (%)
OpenCUA-72B	Q00	650.0	780.0	130.0	20.0
	Q01	636.7	760.0	123.3	19.4
Qwen2.5-VL-72B	Q00	520.0	645.0	125.0	24.0
	Q01	545.0	695.0	150.0	27.5

Table 19: Memory added by IntentGuard on EnvGuard-Bench. Entries are mean per-episode peaks in GiB, reported separately for host and GPU memory. Each Δ is IntentGuard minus Bare.

		Peak host memory			Peak GPU memory		
Base agent	Condition	Bare	IG	Δ	Bare	IG	Δ
OpenCUA-72B	Q00	7.2	8.9	1.7	153.4	171.7	18.3
	Q01	7.5	9.2	1.7	154.1	172.5	18.4
Qwen2.5-VL-72B	Q00	7.0	8.7	1.7	148.2	166.5	18.3
	Q01	7.3	9.0	1.7	148.9	167.3	18.4

increment is consistent with a resident guard-model cost. Episode-time overhead varies with the base agent and environmental condition.