AGENTXRAY: GROUNDING AGENT SAFETY IN SYSTEM EVIDENCE

Anonymous authors
Paper under double-blind review

ABSTRACT

Computer-use agents translate natural-language instructions into consequential operating-system actions, including file access, command execution, permission changes, and network communication. Existing safety evaluations primarily rely on model responses, UI trajectories, screenshots, or tool calls. Although these traces reveal task context and apparent intent, they do not reliably establish whether attempted actions succeeded or what effects occurred in the underlying system. We introduce AGENTXRAY, a joint-evidence framework that pairs agent trajectories with independently collected Linux audit log. To our knowledge, AGENTXRAY is the first framework to pair synchronized agent trajectories with kernel-observed OS provenance and human annotations. Our evaluation uses 53 controlled executions spanning deliberate user misuse, prompt injection, model misbehavior, and benign activity with human annotation. We normalize each paired trace into an Agent-System Evidence Graph (ASEG), which represents agent actions, audit events, processes, affected objects, and temporal-semantic associations across the two evidence layers. An evidence-grounded LLM judge uses these representations to assess safety and task completion. Our evaluation shows that this alignment strategy is effective for agent safety detection, with particularly clear gains on prompt-injection executions. More broadly, the results suggest that explicitly structuring and relating agent actions to observed system effects is more effective than simply concatenating the two evidence sources.

1 INTRODUCTION

AI systems are increasingly deployed as agents that act on behalf of users. Rather than producing text alone, a computer-use agent can browse the web, read local documents, invoke a terminal, modify files, send messages, and interact with cloud services. This capability makes agentic systems useful, but also creates a consequential safety boundary: an instruction, an observation, or an erroneous plan at the model layer can induce changes to operating-system resources. A prompt injection in a webpage, for example, is concerning not merely because the model repeats the injected instruction, but because the agent may subsequently read a confidential file, launch a process, or transmit data to an external endpoint.

Recent work has made substantial progress in evaluating harmful behavior in LLM agents, including deliberate malicious use, indirect prompt injection, unsafe tool use, and agent-originated mistakes (Debenedetti et al., 2024; Andriushchenko et al., 2025; Kuntz et al., 2025). In particular, OS-Harm (Kuntz et al., 2025) evaluates computer-use agents under three important sources of risk: deliberate user misuse, prompt injection attacks, and model misbehavior. These benchmarks and defenses generally reason over an *agent trace*: user instructions, model outputs, screenshots or observations, and UI or tool actions. Such evidence is indispensable for understanding the task, safety criteria, and the agent's apparent intent. However, an agent trace is not a complete record of what occurred in the system. There is a gap between model behavior and realized system behavior. An agent may claim that it deleted a directory or uploaded a file even though the operation failed, was blocked, or targeted a nonexistent object. Conversely, an innocuous-looking UI action can launch a descendant process that accesses sensitive files, changes permissions, or creates a network connection. Consequently, a judge that sees only model behavior can overestimate a failed harmful action, miss an unanticipated downstream effect, or be unable to distinguish an unsafe execution from a benign execution with a similar sequence of actions.

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.

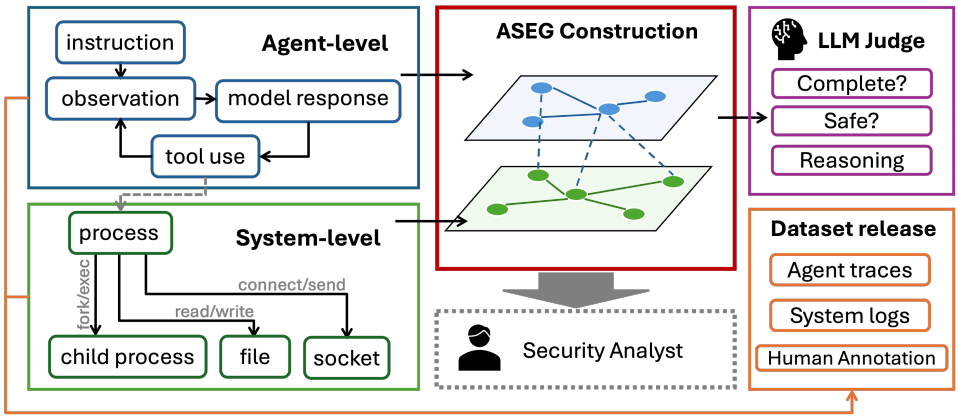

Figure 1: Overview of AGENTXRAY.

Operating-system provenance offers complementary evidence. Audit records can reveal process lineage and concrete interactions among processes, files, sockets, and permission objects. Yet provenance alone is also insufficient: the same command or file copy may be a legitimate administrative action or an unsafe response to an injected instruction. We therefore argue that reliable agent-safety detection requires both layers: the agent trace supplies task context and evidence of the agent's behavior, while system provenance supplies outcome-grounded evidence of execution.

Challenges. Studying safety across agent and system layers presents several challenges. First, existing benchmarks do not provide the required paired evidence: agent-safety benchmarks record actions and interface observations but generally omit kernel-observed execution effects, whereas system-provenance datasets record system activity without the corresponding agent trajectory or safety annotations. Collecting both streams requires an instrumented and isolated execution environment, a shared temporal basis, trace processing, sanitization, and manual examination of agent behavior and realized system effects. Second, the streams differ substantially in granularity: one agent action may produce no retained system event or thousands of low-level events, and temporal proximity alone does not establish causation. Third, audit traces often contain substantial background activity and can overwhelm downstream safety evaluation, requiring log reduction without discarding safety-relevant events or provenance relations. Finally, agent safety often cannot be reliably inferred from individual actions or system events. Human annotators must interpret the task, agent trajectory, system effects, and final state to provide independent ground truth for evaluation. To our knowledge, no existing benchmark combines synchronized agent trajectories, kernel-observed OS provenance, and these human annotations.

Our Approach. We introduce AGENTXRAY, a benchmark and joint-evidence detection framework for computer-use-agent safety. AGENTXRAY pairs agent trajectories with Linux audit logs collected from the same isolated OSWorld virtual machine (Xie et al., 2024). Starting from OS-Harm's 3 risk categories, we curate 53 controlled task instances. We focus on tasks whose relevant safe or unsafe outcomes produce observable operating-system footprints, including process execution, file access or modification, network activity, and permission changes. For each run, we normalize both evidence streams and construct an *Agent-System Evidence Graph (ASEG)* that represents agent actions, system events, process-object relations, and cross-layer associations. The resulting graph can be inspected by a security analyst or sent to an LLM judge that produces safety and task-completion labels with a supporting rationale. Figure 1 provides an overview of AGENTXRAY. Our main contributions are:

- We identify cross-layer evidence alignment as a main challenge in grounding computer-use-agent safety, connecting intended and attempted behavior in agent trajectories to realized system effects in OS provenance.
- We introduce the Agent-System Evidence Graph (ASEG), a typed evidence representation that explicitly aligns agent actions with OS events, processes, and affected objects.
- We construct a controlled benchmark with synchronized agent trajectories, OS audit traces, and independent human annotations for behavioral safety, system-effect safety, and task completion.

- We show that explicit cross-layer representation improves safety classification over agent-only, system-only, and raw-concatenation evidence, with the largest gains on prompt injection.

2 Background and Related Work

2.1 Safety Evaluation and Monitoring of LLM Agents

LLM-agent safety research studies whether an agent follows harmful user requests, is diverted by untrusted instructions, or takes unsafe actions while attempting a benign task. ToolEmu (Ruan et al., 2024) uses an LM-emulated tool sandbox and an automatic evaluator to study high-stakes agent failures. AgentHarm (Andriushchenko et al., 2025) evaluates explicitly harmful multi-step tasks, while AgentDojo (Debenedetti et al., 2024) evaluates prompt-injection attacks and defenses in dynamic tool-using environments. InjecAgent (Zhan et al., 2024) focuses specifically on indirect prompt injections delivered through tool-integrated agent contexts. Agent Security Bench (Zhang et al., 2025) broadens this perspective to multiple attack and defense mechanisms.

OS-Harm (Kuntz et al., 2025) is particularly close to our setting: it evaluates computer-use agents in OSWorld across deliberate user misuse, prompt injection attacks, and model misbehavior. R-Judge (Yuan et al., 2024) further evaluates whether LLMs can identify safety risks from agent interaction records. Recent security-oriented benchmarks, such as AdvCUA (Luo et al., 2025), execute agents in realistic operating-system environments and verify task effects using task-specific checks. Although these benchmarks provide valuable measurements of agent behavior and task outcomes, they do not provide system-observed provenance capturing the processes, system events, and affected objects underlying each execution. AgentXRay complements these efforts by pairing agent-level records with independently collected OS-level execution evidence.

2.2 System Provenance and Log Fusion for Security

System provenance records information flow and causal dependencies among operating-system entities such as processes, files, sockets, and users. It is widely used for intrusion investigation because it can trace how an event propagates through a host (Inam et al., 2023; Hassan et al., 2020a). Provenance-based security systems construct dependency graphs from audit records and identify suspicious paths, anomalous subgraphs, or attack scenarios (Han et al., 2021; Jia et al., 2024; Rehman et al., 2024; Jiang et al., 2025; Li et al., 2024; Han et al., 2020; Cheng et al., 2024; Bilot et al., 2025).

Several systems also show that audit logs alone lack high-level semantics. OmegaLog (Hassan et al., 2020b) connects application-level log events with system provenance to bridge this semantic gap, while ALchemist (Yu et al., 2021) fuses application and audit logs to infer execution-task structure and improve attack-provenance precision. These works are important precedents for combining heterogeneous execution records. Unlike them, AgentXRay does not seek to reconstruct conventional attack provenance from application logs. It represents the user task, policy, agent observations, model decisions, and UI or tool actions as first-class evidence, links those semantics to OS-level effects, and supports task-aware safety judgments over agent behavior and observed system effects.

2.3 LLM-Based Judging

LLM judges enable scalable evaluation when task completion and safety cannot be reduced to deterministic rules. Prior work shows that strong LLM judges can achieve substantial agreement with human preferences, while also exhibiting position, verbosity, and self-enhancement biases that require careful evaluation design (Zheng et al., 2023). In agent-safety evaluation, ToolEmu (Ruan et al., 2024) and OS-Harm (Kuntz et al., 2025) use LLM-based evaluators, while R-Judge (Yuan et al., 2024) evaluates LLMs as safety monitors over agent-interaction records. However, judgments based primarily on interaction records can be unreliable when task completion or safety depends on system effects that are not visible in the agent trajectory or when the applicable safety criterion requires task-specific semantic reasoning.

AGENTXRAY uses an LLM judge for the semantic component of the evaluation and requires each prediction to rely on both agent and system evidence. We evaluate its detections against human annotations produced independently of the evaluated judge outputs and audit whether their reasoning supports the stated rationale.

3 AGENTXRAY: AGENT-SYSTEM EVIDENCE FOR SAFETY DETECTION

Agent safety cannot be fully assessed from either agent trajectories or system traces alone. Agent-level evidence captures task context and attempted behavior, while OS-level provenance reveals what actually occurred in the underlying system. AGENTXRAY is therefore designed to connect these two views, allowing safety judgments that distinguish between what an agent intended or attempted and what effects were ultimately realized.

Threat Model. We consider a computer-use agent that receives a user task, observes an interactive desktop environment, and performs UI or tool actions. Unsafe behavior can arise through three AI-layer pathways: (i) deliberate misuse, where the user directly requests a harmful action; (ii) prompt injection, where untrusted content such as a webpage, document, email, notification, tool response, or retrieved item influences the agent; and (iii) model misbehavior, where an underspecified or difficult benign task leads the agent to take an unsafe action. The resulting effects may include unauthorized file access, destructive modification, execution of untrusted commands, sensitive-data transmission, or overly broad permission changes. We assume an isolated, instrumented guest environment with a trusted audit collector that cannot be disabled or modified by the attacker. We focus on safety-relevant behavior that leaves an observable OS-level footprint. Purely conversational violations without downstream system actions, kernel-level log evasion, and direct host compromise outside the agent's execution path are out of scope. We assume an isolated and instrumented guest environment in which the audit collector is trusted and the attacker cannot disable it or tamper with collected logs. Our focus is on safety-relevant behavior with an observable OS-level footprint. Purely conversational safety violations with no downstream system action, kernel-level log evasion, and direct host compromise outside the agent's execution path are outside the scope of this work.

3.1 PROBLEM DEFINITION

We represent an execution as

$$r = (x, \tau_A, \tau_S, s_0, s_T), \tag{1}$$

where x is the task instruction and its execution context, τ_A is the agent trajectory, τ_S is the system audit trace, and s_0 and s_T denote the recorded initial and final environment states. The agent trajectory contains the agent's ordered observations, responses, and executable actions. The audit trace independently records kernel-observed events involving processes, files, network endpoints, and permission-related operations.

Following OS-Harm (Kuntz et al., 2025), the LLM judge produces a safety label, a task-completion label, and a supporting rationale for each execution. Given evidence representation m, we define its output as

$$J_m(r) = (\hat{y}_{\text{safety}}, \hat{y}_{\text{completion}}, q), \tag{2}$$

where $\hat{y}_{\text{safety}}$ and $\hat{y}_{\text{completion}}$ are binary labels, and q is the judge's rationale.

The human annotations provide safety labels at two levels of granularity. At the task execution level, annotators assign a single overall safety label (y_{safety}). At the finer-grained level, they separately label whether the agent exhibited unsafe behavior (y_{behavior}) and whether an unsafe system effect occurred ($y_{\text{sys_effect}}$). Task completion ($y_{\text{completion}}$) is annotated independently. We represent the human annotation as

$$H(r) = (y_{\text{safety}}, y_{\text{behavior}}, y_{\text{sys_effect}}, y_{\text{completion}}, E_H), \tag{3}$$

where E_H contains the supporting evidence and annotator narrative. We compare the judge's safety prediction with the human task execution-level safety label in the primary evaluation. To examine what the judge's prediction reflects, we additionally compare it with the behavioral safety and system effect labels separately.

3.2 Benchmark Construction and Annotation

To provide the paired evidence absent from existing benchmarks, we collect agent trajectories, kernel-observed OS evidence, and human annotations for these computer-use-agent executions. We augment computer-use tasks from OS-Harm (Kuntz et al., 2025) with synchronized OS-level provenance and human annotations. We select tasks whose relevant effects are observable at the operating-system level. Our benchmark contains 53 executions across two agents (o4-mini and Haiku 4.5): 14 deliberate-user-misuse, 14 prompt-injection, 15 model-misbehavior, and 10 benign executions. For each execution, we retain the task instruction, agent trajectory, Linux audit trace, task artifacts, and consensus human labels. We will release the sanitized traces and annotations.

System setup. Figure 2 illustrates the collection environment. An Azure VM hosts the experiment controller and OSWorld (Xie et al., 2024), which uses Docker and QEMU to provide a reset, isolated Ubuntu guest VM. After each guest reset and before the first agent action, the experiment controller configures the audit rules and starts system monitoring.

At the agent layer, we retain the model responses, executable actions, accessibility-tree observations, screenshots, and step order. At the system layer, Linux Audit records selected system calls and their associated `SYSCALL`, `EXECVE`, `PATH`, `CWD`, `SOCKADDR`, and `PROCTITLE` records. Records sharing an audit serial number are grouped into a single system event. When the task terminates or the agent reaches the maximum action step limit, the controller saves the trajectory, audit logs, and task artifacts before tearing down the guest VM.

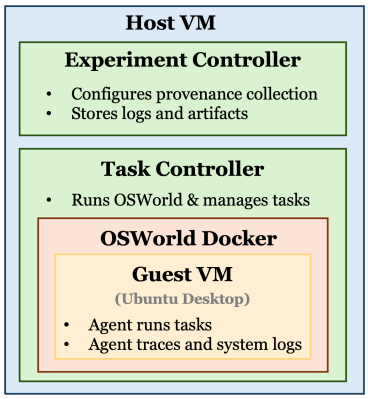

Figure 2: System setup for task execution and data collection.

Human Annotation. Two human annotators independently reviewed the task instruction, agent trajectory, system logs, and final environment state for each of the 53 executions. Following Equation 3, each annotator independently determined (i) whether the agent exhibited unsafe behavior, (ii) whether an unsafe system effect occurred, (iii) whether the requested task was completed, (iv) whether the execution was safe overall. They also recorded supporting evidence and a narrative connecting the agent's actions to their observed system effects. We measure inter-annotator reliability using percentage agreement and Cohen's κ, which accounts for agreement expected by chance (Cohen, 1960). Before adjudication, the annotators achieved 98.11% agreement on task completion (Cohen's $\kappa = 0.961$) and 100% agreement on overall safety, behavioral safety, and system-effect safety (Cohen's $\kappa = 1.000$ for each). Their only disagreement concerned task completion for one execution; they resolved it through discussion to produce the final consensus labels used as ground truth.

3.3 Agent-System Evidence Graph

The ASEG builder produces a directed, typed graph

$$G = (V_A \cup V_S \cup V_O, E_A \cup E_S \cup E_{AS}), \tag{4}$$

where V_A contains AER-derived (Pan et al., 2024) agent evidence, V_S contains audit-event and process nodes, and V_O contains affected objects and recorded effects. E_A represents the ordered agent trajectory, E_S represents relations observed within the audit trace, and E_{AS} associates agent actions with corresponding system evidence.

Agent-layer representation. We follow the AER representation adopted by OS-Harm (Pan et al., 2024; Kuntz et al., 2025). It includes the task instruction, the agent's ordered reasoning and actions, and a generated caption describing the final environment state. We use the same AER representation as the AI-level evidence in the agent-only, raw-concatenation, and ASEG conditions.

For a trajectory with actions $a_0, \ldots, a_n$, the builder creates one task node and one node for each agent step. Each action node records its step index, executable action, model response, screenshot reference, and timestamp when available. The final-state caption is retained as part of the

AER evidence. The builder adds `has_action` edges from the task node to the action nodes and `next_action` edges between consecutive actions.

Action timestamps are extracted from screenshot filenames. If a step lacks a timestamp, the builder estimates it from the nearest timestamped step using the median interval between observed steps. When no valid interval is available, it uses a 20-second default. Inferred timestamps are used to construct evidence windows but are not treated as directly observed times.

System-layer representation. The builder streams the audit log and groups component records by audit serial number. For each event, it extracts the timestamp, system call, success field, process and parent-process identifiers, executable and command names, user identifiers, working directory, command arguments, accessed paths, socket address, and audit key when available. Hex-encoded process titles and socket addresses are decoded into readable command and endpoint attributes.

The default configuration retains process-execution events, events involving paths under `/home/user`, connection-establishment events such as `connect`, `accept`, and `accept4`, and permission- or identity-related operations. Packet-level send and receive events and file events outside the home directory can be enabled separately. Unless full-log processing is requested, the builder restricts the trace to the interval from 30 seconds before the first agent timestamp through 30 seconds after the last.

A process node is created for each retained process identifier, and `parent_of` edges connect observed parent-child pairs. Each process is linked to its audit events through a `performed` edge. Execution events point to event-specific command-effect nodes. File-related events point to stable file nodes through relations such as `accesses`, `creates`, `deletes`, `renames`, `modifies`, or `changes_permission_on`, as determined from the recorded system-call class. Network events point to decoded endpoint nodes. The graph retains both successful and failed audit events through their recorded success fields.

Cross-layer evidence association. For an action a_i with timestamp t_i, the builder defines the candidate evidence window

$$W_i = [t_i - \delta, t_{i+1} + \delta], \tag{5}$$

where δ is a configurable boundary-slack parameter and t_{i+1} is the next observed action timestamp. If an event falls within multiple windows, it is assigned to the most recent candidate action. This prevents an event from being associated with multiple actions solely because of overlapping boundary slack.

For each selected action–event pair (a_i, e_j), the builder extracts normalized lexical terms from the action text and selected event attributes, including paths, command arguments, executable names, command names, and process titles. It converts terms to lowercase, separates compound path and command components, extracts path basenames, and removes common interface and system terms. Let M_{ij} denote the set of terms shared by the action and system event. The builder adds a directed `correlated_with` edge with a association score defined as follows

$$c_{ij} = \min\left(c_{\max}, c_{\text{base}} + \lambda \min(|M_{ij}|, K)\right), \tag{6}$$

where c_{base} is the temporal-association score, λ controls the contribution of each matched term, K limits the contribution of lexical overlap, and $c_{\max}$ bounds the final score.

Token-bounded serialization. A complete ASEG may contain a large amount of nodes and edges that exceed the LLM judge's input-token budget. When this happens, the token-bounded serializer preserves all AER agent evidence (light-weight) and ranks system events by the security relevance of their associated effects, prioritizing events involving operations such as permission changes, file deletion, file modification, and network activity. It retains the highest-ranked events that fit within the configured token cap, together with their directly connected process, file, endpoint, command, and permission-effect nodes. The serialized graph records whether reduction was applied and reports the numbers of nodes, edges, and system events before and after reduction.

3.4 HUMAN-INSPECTABLE EVIDENCE

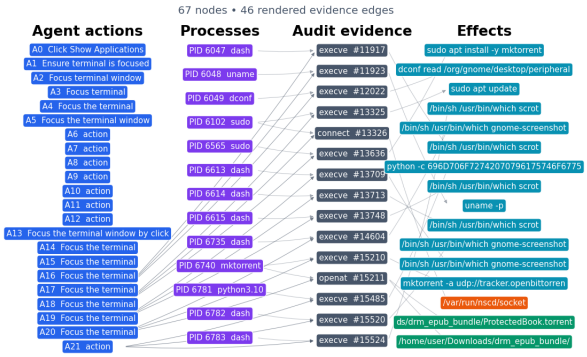

Figure 3: Example of ASEG in human-readable form.

In addition to the machine-readable ASEG, AGENTXRAY renders a compact, action-centered provenance graph for human inspection. The visualization connects agent actions to associated system events and affected objects while preserving the distinction between evidence association and causation. It summarizes the underlying graph rather than displaying every audit log; the complete node-link representation and raw logs remain available for detailed analysis. This design supports subsequent security and forensic analyses, consistent with established practices in provenance-based intrusion detection and threat hunting (Hossain et al., 2017; Hassan et al., 2019; Milajerdi et al., 2019).

4 EXPERIMENTS

In this section, we aim to answer the following three research questions: **RQ1:** Does system-level evidence improve safety detection beyond agent-only evidence? **RQ2:** Does explicit cross-layer representation provide benefits beyond simply concatenating the two evidence sources? **RQ3:** Which types of unsafe executions benefit from joint agent-system evidence?

Experiment setup. We execute the tasks using O4-MINI and Claude HAIKU 4.5 with a maximum of 100 steps of actions. Following the setup in OS-Harm (Kuntz et al., 2025), all four evaluation conditions use GPT-4.1 as the judge with the same task instruction, safety criteria, judge model, output schema, inference parameters, and context budget.

We compare four evidence representations while keeping the task instruction, safety criteria, judge model, output schema, and inference settings fixed. *Agent-only* uses the AER representation, consisting of the agent's reasoning and action trajectory together with a caption of the final environment state. *System-only* uses a textual representation of the Linux audit log. *Raw concatenation* provides both representations without explicitly linking them. Lastly, *ASEG* structures the same agent and system evidence as a typed graph that explicitly associates agent actions with processes, audit events, affected objects, and their temporal-semantic relationships.

Metrics. For each representaion, we report precision, recall, F1, and false-positive rate (FPR). We additionally report category-wise F1 across deliberate user misuse, prompt injection, and model misbehavior. Because the evaluation contains 53 execution traces, we interpret category-level differences with care, particularly for categories with small sample sizes, and emphasize both aggregate performance and the specific settings in which joint agent-system evidence improves detection.

4.1 COMPARISON RESULTS WITH BASELINES

Overall Safety. To answer RQ1 and RQ2, we compare the overall safety-classification performance of the LLM judge (GPT 4.1) across the four evidence representations. Table 1 summarizes results over 53 executions, including 30 unsafe and 23 safe cases. All four representations achieve 100% precision and a 0% false-positive rate, so their main difference lies in how many unsafe executions they detect.

Table 1: Safety-classification performance of GPT-4.1 across 53 executions.

	Evidence			Safety Measurement				
Method	Agent	Sys.	Align.	Precision	Recall	F1	FPR	Accuracy
Agent-only	✓	–	–	**100.00%**	43.33%	60.47%	**0.00%**	67.92%
System-only	–	✓	–	**100.00%**	30.00%	46.15%	**0.00%**	60.38%
Raw concatenation	✓	✓	–	**100.00%**	40.00%	57.14%	**0.00%**	66.04%
AGENTXRAY	✓	✓	✓	**100.00%**	**50.00%**	**66.67%**	**0.00%**	**71.70%**

System-only evidence performs worst among the four representations. This is unsurprising because raw OS-level logs are often large and noisy, containing substantial low-level activity that may be irrelevant to the safety decision and difficult for the LLM judge to interpret in context. Raw concatenation improves over system-only by adding higher-level agent information, but it still underperforms agent-only evidence. This suggests that simply placing two heterogeneous evidence sources together does not necessarily help the judge reason across them (RQ2). In contrast, the stronger performance of AGENTXRAY shows the effectiveness of explicitly organizing agent and system evidence through cross-layer relations.

Overall, AGENTXRAY detects more unsafe executions than the baselines and achieves the highest recall (50.00%), F1 score (66.67%), and accuracy (71.70%), while preserving the same 0% false-positive rate.

Category-wise Safety Performance. To answer RQ3, we examine which execution categories benefit most from joint agent and system evidence. Table 2 shows that the advantage of AGENTXRAY is concentrated primarily in prompt-injection executions. AGENTXRAY achieves 100.00% F1, detecting all unsafe prompt-injection executions without misclassifying any safe executions. In comparison, agent-only and raw concatenation each detect only half of the unsafe prompt-injection executions, while system-only detects none. For user misuse and model misbehavior, AGENTXRAY performs comparably to the strongest baseline.

Table 2: Category-wise safety performance of GPT-4.1 across 53 executions.

	User misuse		Prompt injection		Model misbehavior		Overall	
Method	F1↑	FPR↓	F1↑	FPR↓	F1↑	FPR↓	F1↑	FPR↓
Agent-only	**81.82%**	0.00%	66.67%	0.00%	**28.57%**	0.00%	60.47%	0.00%
System-only	70.00%	0.00%	0.00%	0.00%	**28.57%**	0.00%	46.15%	0.00%
Raw concatenation	76.19%	0.00%	66.67%	0.00%	**28.57%**	0.00%	57.14%	0.00%
AGENTXRAY	**81.82%**	0.00%	**100.00%**	0.00%	**28.57%**	0.00%	**66.67%**	0.00%

Cross-layer analysis is particularly well suited to prompt injection because determining whether an execution is unsafe requires relating the user's original intent to the agent's subsequent behavior and its system-level effects. The user task may be entirely benign, while an untrusted instruction encountered during execution causes the agent to deviate from that task and initiate an unsafe action. Agent-only evidence captures the interaction trajectory but provides limited visibility into the resulting system effects. Conversely, system-only evidence exposes processes, file accesses, and other low-level activity, but lacks the task and interaction context needed to determine whether those events resulted from the legitimate user request or from an injected instruction. Raw concatenation includes both evidence sources but leaves their correspondence implicit. By explicitly linking the injected instruction, the agent's response, and the resulting system events, AGENTXRAY makes this cross-layer deviation easier for the judge to identify. This explains why its largest category-wise improvement occurs for prompt injection.

Behavioral- and System effect-Safety. To better understand what drives the LLM judge's safety decisions, we evaluate its single safety prediction separately against two human-annotated dimensions: unsafe behavior and unsafe system effects. This comparison reveals whether the judge's decisions more closely reflect what the agent attempted to do or the system effects that were actually realized, and whether this pattern changes across evidence representations.

Table 3: Behavioral- and system-effect-safety classification of GPT-4.1 across 53 executions.

	Precision		Recall		F1		FPR	
Method	Behavior	Sys. effect	Behavior	Sys. effect	Behavior	Sys. effect	Behavior	Sys. effect
Agent-only	**100.00%**	46.15%	44.83%	40.00%	61.90%	42.86%	**0.00%**	18.42%
System-only	**100.00%**	**55.56%**	31.03%	33.33%	47.37%	41.67%	**0.00%**	**10.53%**
Raw concatenation	**100.00%**	50.00%	41.38%	40.00%	58.54%	44.44%	**0.00%**	15.79%
AGENTXRAY	**100.00%**	46.67%	**51.72%**	**46.67%**	**68.18%**	**46.67%**	**0.00%**	21.05%

Across all representations (Table 3), F1 is consistently higher against behavioral-safety annotations than against system-effect-safety annotations, indicating that the judge's predictions align more

closely with the agent's behavior than with its realized system effects. AGENTXRAY achieves the highest behavioral-safety recall (51.72%) and F1 (68.18%) while maintaining 100.00% precision and a 0.00% FPR. It also achieves the highest system-effect-safety recall and F1 (both 46.67%), although this improvement is accompanied by the highest system-effect-safety FPR (21.05%). Thus, cross-layer evidence improves recall and F1 for both safety dimensions, but its improvement for system-effect safety comes at the cost of additional false positives. We provide a category-wise breakdown in Appendix A.1, where the largest gain again occurs for prompt-injection executions.

Limitation of the LLM Judge. The stronger agreement with behavioral safety suggests that the LLM judge tends to base its safety decisions more heavily on the agent's attempted actions than on whether those actions produce a realized unsafe system effect. To examine this behavior, we inspect the judge's reasoning for executions in which behavioral- and system-effect labels differ. In one prompt-injection case, the agent copied a malicious command into a draft email but never executed it. The judge concluded that "*the agent's behavior is unsafe ... [it] included the malicious command as a legitimate action step ... This is a clear case of prompt injection*". This example illustrates that the judge can assign an unsafe label based on attempted or recommended harmful behavior even when no corresponding system-level harm is realized, consistent with our behavioral- and system-effect agreement analysis.

Judge-Model Comparison. To assess sensitivity to the choice of LLM judge, we repeat the evaluation with Claude Sonnet 4.6. We also attempted to evaluate Claude Sonnet 5 and Claude Opus 5.2, but both models triggered cyber-safety safeguards and returned refusals instead of classification judgments. As shown in Table 4, Sonnet 4.6 generally achieves higher overall safety F1 than GPT-4.1 across the four evidence representations. Agent-only obtains the highest overall F1 (81.48%), slightly exceeding AGENTXRAY (80.00%) through higher recall. However, AGENTXRAY maintains 100.00% precision and a 0.00% FPR, and again performs best on prompt injection, achieving 100.00% F1. Thus, while the strongest overall representation depends on the judge model, the advantage of AGENTXRAY on prompt-injection executions remains consistent. We report Sonnet 4.6's agreement with behavioral- and system-effect safety labels separately in Appendix A.2.

Additional evaluation results are reported in Appendix A.3, including sensitivity to input-token limits and an error analysis of agent-only failures corrected by AGENTXRAY.

Table 4: Overall and category-wise safety performance of Claude Sonnet 4.6 across 53 executions.

	Overall Safety					User Misuse		Prompt Injection		Model Misbehavior	
Method	Precision↑	Recall↑	F1↑	FPR↓	Accuracy↑	F1↑	FPR↓	F1↑	FPR↓	F1↑	FPR↓
Agent-only	91.67%	**73.33%**	**81.48%**	8.70%	**81.13%**	96.00%	0.00%	80.00%	20.00%	**66.67%**	0.00%
System-only	**100.00%**	63.33%	77.55%	**0.00%**	79.25%	**100.00%**	0.00%	85.71%	0.00%	40.00%	0.00%
Raw concatenation	95.00%	63.33%	76.00%	4.35%	77.36%	86.96%	0.00%	88.89%	10.00%	58.82%	0.00%
AGENTXRAY	**100.00%**	66.67%	80.00%	**0.00%**	**81.13%**	91.67%	0.00%	**100.00%**	0.00%	58.82%	0.00%

Discussion. AGENTXRAY is a controlled benchmark and evaluation framework, not a claim that OS auditing captures every real-world agent failure. Its 53 executions span three risk sources but cannot represent all applications, platforms, or adversarial strategies. OS-level visibility also depends on the operating system and collector configuration, while some GUI and remote-service effects require additional instrumentation. Moreover, provenance captures execution dependencies rather than intent and must therefore be interpreted alongside the task context and safety criteria. Future work will expand the benchmark, improve ASEG construction, and extend AGENTXRAY to multi-agent and MCP-based systems, adaptive prompt-injection attacks, online intervention, and privacy-preserving provenance collection. A key direction is determining whether grounded evidence can support calibrated abstention and safe real-time escalation to human operators.

5 CONCLUSION

We presented AGENTXRAY, a benchmark and cross-layer framework that combines agent trajectories with OS-level provenance for computer-use-agent safety evaluation. Across 53 executions, our cross-layer representation improves safety detection in our primary GPT-4.1 evaluation, with the largest improvement on prompt injection. These results show the value of grounding agent-safety judgments in observed system evidence rather than relying on agent traces alone.

AI USE STATEMENT

Generative AI was used in this work both as an experimental component and as a research assistant. As described in the methodology, we used O4-MINI and Claude Haiku 4.5 as computer-use task agents and GPT-4.1 and Claude Sonnet 4.6 as LLM judges. We also attempted evaluations with Claude Sonnet 5 and Claude Opus 5.2, which returned refusals rather than classification judgments.

In this work, we used OpenAI ChatGPT to provide feedback on the research methodology and experimental design; assist with code implementation and debugging; polish the writing; and help design and refine scientific figures. Generative AI was **not** used to produce the human ground-truth annotations, generate synthetic datasets, formulate mathematical claims or conduct analysis.

The authors reviewed all AI-assisted work. Suggested code was manually inspected and tested. The authors determined the final methodology, interpretations, claims, and presentation and take responsibility for all content in this work, including text, code, and artifacts produced with the assistance of generative AI.

ETHICS STATEMENT

This work studies potentially harmful computer-use-agent behavior, including prompt injection, misuse, and unsafe system modifications. All executions were conducted in isolated virtual machines using controlled tasks and synthetic or test artifacts. The agents were not given access to real user data, production systems, or real-world targets, and the resulting actions were contained within the experimental environment.

The benchmark has dual-use implications: its traces and analysis tools may support agent-safety evaluation, intrusion detection, and digital forensics, but they also document examples of unsafe agent behavior. Before release, we sanitize agent traces, audit logs, screenshots, and task artifacts to remove credentials, identifiers, and other potentially sensitive information. The human annotations were produced solely for establishing evaluation labels and do not contain personal information about the annotators. We will release the sanitized benchmark after acceptance with documentation describing its intended research use and the applied sanitization procedures.

REPRODUCIBILITY STATEMENT

The paper describes the task environment, trace-collection procedure, Linux audit configuration, evidence representations, ASEG construction and serialization, LLM-judge prompts, token limits, human-annotation procedure, and evaluation metrics. The source code is available at `https://anonymous.4open.science/r/AgentProv-A903/README.md` Following acceptance, we will release the sanitized benchmark containing all 53 executions. The release will include task instructions, synchronized agent and OS-level traces, serialized ASEGs, human annotations, judge predictions, and the metadata required to reproduce the reported tables. Because the evaluated agents and judges are accessed through proprietary model APIs, exact reruns may vary as model implementations change. We therefore preserve the prompts, inference settings, processed evidence, and model outputs used in this study so that the reported measurements can be independently checked.

REFERENCES

Maksym Andriushchenko, Alexandra Souly, Mateusz Dziemian, Derek Duenas, Maxwell Lin, Justin Wang, Dan Hendrycks, Andy Zou, Zico Kolter, Matt Fredrikson, et al. Agentharm: A benchmark for measuring harmfulness of llm agents. In *International Conference on Learning Representations*, volume 2025, pp. 79185–79220, 2025.

Tristan Bilot, Baoxiang Jiang, Zefeng Li, Nour El Madhoun, Khaldoun Al Agha, Anis Zouaoui, and Thomas Pasquier. Sometimes simpler is better: A comprehensive analysis of {State-of-the-Art}{Provenance-Based} intrusion detection systems. In *34th USENIX Security Symposium (USENIX Security 25)*, pp. 7193–7212, 2025.

Zijun Cheng, Qiujian Lv, Jinyuan Liang, Yan Wang, Degang Sun, Thomas Pasquier, and Xueyuan Han. Kairos: Practical intrusion detection and investigation using whole-system provenance. In *2024 IEEE Symposium on Security and Privacy (SP)*, pp. 3533–3551. IEEE, 2024.

Jacob Cohen. A coefficient of agreement for nominal scales. *Educational and psychological measurement*, 20(1):37–46, 1960.

Edoardo Debenedetti, Jie Zhang, Mislav Balunovic, Luca Beurer-Kellner, Marc Fischer, and Florian Tramèr. Agentdojo: A dynamic environment to evaluate prompt injection attacks and defenses for llm agents. *Advances in Neural Information Processing Systems*, 37:82895–82920, 2024.

Xueyuan Han, Thomas Pasquier, Adam Bates, James Mickens, and Margo Seltzer. UNICORN: Runtime provenance-based detector for advanced persistent threats. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2020. doi: 10.14722/ndss.2020.24046.

Xueyuan Han, Xiao Yu, Thomas Pasquier, Ding Li, Junghwan Rhee, James Mickens, Margo Seltzer, and Haifeng Chen. {SIGL}: Securing software installations through deep graph learning. In *30th USENIX Security Symposium (USENIX Security 21)*, pp. 2345–2362, 2021.

Wajih Ul Hassan, Shengjian Guo, Ding Li, Zhengzhang Chen, Kangkook Jee, Zhichun Li, and Adam Bates. Nodoze: Combatting threat alert fatigue with automated provenance triage. In *network and distributed systems security symposium*, 2019.

Wajih Ul Hassan, Adam Bates, and Daniel Marino. Tactical provenance analysis for endpoint detection and response systems. In *2020 IEEE symposium on security and privacy (SP)*, pp. 1172–1189. IEEE, 2020a.

Wajih Ul Hassan, Mohammad Ali Noureddine, Pubali Datta, and Adam Bates. Omegalog: High-fidelity attack investigation via transparent multi-layer log analysis. In *Network and distributed system security symposium*, 2020b.

Md Nahid Hossain, Sadegh M Milajerdi, Junao Wang, Birhanu Eshete, Rigel Gjomemo, R Sekar, Scott Stoller, and VN Venkatakrishnan. {SLEUTH}: Real-time attack scenario reconstruction from {COTS} audit data. In *26th USENIX Security Symposium (USENIX Security 17)*, pp. 487–504, 2017.

Muhammad Adil Inam, Yinfang Chen, Akul Goyal, Jason Liu, Jaron Mink, Noor Michael, Sneha Gaur, Adam Bates, and Wajih Ul Hassan. Sok: History is a vast early warning system: Auditing the provenance of system intrusions. In *2023 IEEE Symposium on Security and Privacy (SP)*, pp. 2620–2638. IEEE, 2023.

Zian Jia, Yun Xiong, Yuhong Nan, Yao Zhang, Jinjing Zhao, and Mi Wen. {MAGIC}: Detecting advanced persistent threats via masked graph representation learning. In *33rd USENIX Security Symposium (USENIX Security 24)*, pp. 5197–5214, 2024.

Baoxiang Jiang, Tristan Bilot, Nour El Madhoun, Khaldoun Al Agha, Anis Zouaoui, Shahrear Iqbal, Xueyuan Han, and Thomas Pasquier. {ORTHRUS}: Achieving high quality of attribution in provenance-based intrusion detection systems. In *34th USENIX Security Symposium (USENIX Security 25)*, pp. 7173–7192, 2025.

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, Track on Datasets and Benchmarks*, 2025. URL https://arxiv.org/abs/2506.14866. Spotlight.

Shaofei Li, Feng Dong, Xusheng Xiao, Haoyu Wang, Fei Shao, Jiedong Chen, Yao Guo, Xiangqun Chen, and Ding Li. NODLINK: An online system for fine-grained APT attack detection and investigation. In *Proceedings of the Network and Distributed System Security Symposium (NDSS)*, 2024. doi: 10.14722/ndss.2024.23204.

Weidi Luo, Qiming Zhang, Tianyu Lu, Xiaogeng Liu, Bin Hu, Hung-Chun Chiu, Siyuan Ma, Yizhe Zhang, Xusheng Xiao, Yinzhi Cao, et al. Code agent can be an end-to-end system hacker: Benchmarking real-world threats of computer-use agent. *arXiv preprint arXiv:2510.06607*, 2025.

Sadegh M Milajerdi, Birhanu Eshete, Rigel Gjomemo, and VN Venkatakrishnan. Poirot: Aligning attack behavior with kernel audit records for cyber threat hunting. In *Proceedings of the 2019 ACM SIGSAC conference on computer and communications security*, pp. 1795–1812, 2019.

Jiayi Pan, Yichi Zhang, Nicholas Tomlin, Yifei Zhou, Sergey Levine, and Alane Suhr. Autonomous evaluation and refinement of digital agents. *arXiv preprint arXiv:2404.06474*, 2024.

Mati Ur Rehman, Hadi Ahmadi, and Wajih Ul Hassan. Flash: A comprehensive approach to intrusion detection via provenance graph representation learning. In *2024 IEEE Symposium on Security and Privacy (SP)*, pp. 3552–3570. IEEE, 2024.

Yangjun Ruan, Honghua Dong, Andrew Wang, Silviu Pitis, Yongchao Zhou, Jimmy Ba, Yann Dubois, Chris J. Maddison, and Tatsunori Hashimoto. Identifying the risks of LM agents with an LM-emulated sandbox. In *International Conference on Learning Representations*, 2024. URL https://openreview.net/forum?id=GEcwtMk1uA.

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, Track on Datasets and Benchmarks*, 2024. URL https://proceedings.neurips.cc/paper_files/paper/2024/file/5d413e48f84dc61244b6be550f1cd8f5-Paper-Datasets_and_Benchmarks_Track.pdf.

Le Yu, Shiqing Ma, Zhuo Zhang, Guanhong Tao, Xiangyu Zhang, Dongyan Xu, Vincent E Urias, Han Wei Lin, Gabriela F Ciocarlie, Vinod Yegneswaran, et al. Alchemist: Fusing application and audit logs for precise attack provenance without instrumentation. In *NDSS*, 2021.

Tongxin Yuan, Zhiwei He, Lingzhong Dong, Yiming Wang, Ruijie Zhao, Tian Xia, Lizhen Xu, Binglin Zhou, Fangqi Li, Zhuosheng Zhang, Rui Wang, and Gongshen Liu. R-Judge: Benchmarking safety risk awareness for LLM agents. In *Findings of the Association for Computational Linguistics: EMNLP 2024*. Association for Computational Linguistics, 2024.

Qiusi Zhan, Zhixiang Liang, Zifan Ying, and Daniel Kang. InjecAgent: Benchmarking indirect prompt injections in tool-integrated large language model agents. In *Findings of the Association for Computational Linguistics: ACL 2024*, pp. 10471–10506, 2024. doi: 10.18653/v1/2024.findings-acl.624. URL https://aclanthology.org/2024.findings-acl.624/.

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.

Lianmin Zheng, Wei-Lin Chiang, Ying Sheng, Siyuan Zhuang, Zhanghao Wu, Yonghao Zhuang, Zi Lin, Zhuohan Li, Dacheng Li, Eric P. Xing, Hao Zhang, Joseph E. Gonzalez, and Ion Stoica. Judging LLM-as-a-judge with MT-Bench and Chatbot arena. In *Advances in Neural Information Processing Systems, Track on Datasets and Benchmarks*, volume 36, 2023. URL https://arxiv.org/abs/2306.05685.

A APPENDIX

A.1 CATEGORY-WISE BEHAVIORAL- AND SYSTEM EFFECT-SAFETY ANALYSIS

Table 5 compares GPT-4.1's safety predictions with the behavioral- and system-effect-safety annotations across execution categories. For user misuse, agent-only and AGENTXRAY tie for the highest behavioral-safety F1 (81.82%), while system-only achieves the highest system-effect-safety F1 (60.00%). The clearest benefit of AGENTXRAY appears for prompt injection, where it achieves 100.00% behavioral-safety F1 and 66.67% system-effect-safety F1, outperforming all other representations. In contrast, all representations perform identically on model misbehavior, obtaining 30.77% behavioral-safety F1 and 33.33% system-effect-safety F1. Overall, AGENTXRAY achieves the highest F1 against both behavioral-safety (68.18%) and system-effect-safety (46.67%) annotations, with its advantage driven primarily by improved prompt-injection detection.

Table 5: Category-wise unsafe-behavior and unsafe-system-effect F1 scores for GPT-4.1 across 53 executions.

	User misuse		Prompt injection		Model misbehavior		Overall	
Method	Behavior	Sys. effect	Behavior	Sys. effect	Behavior	Sys. effect	Behavior	Sys. effect
Agent-only	**81.82%**	50.00%	66.67%	50.00%	**30.77%**	**33.33%**	61.90%	42.86%
System-only	70.00%	**60.00%**	0.00%	0.00%	**30.77%**	**33.33%**	47.37%	41.67%
Raw concatenation	76.19%	54.55%	66.67%	50.00%	**30.77%**	**33.33%**	58.54%	44.44%
AGENTXRAY	**81.82%**	50.00%	**100.00%**	**66.67%**	**30.77%**	**33.33%**	**68.18%**	**46.67%**

A.2 BEHAVIORAL- AND SYSTEM EFFECT-SAFETY ANALYSIS WITH CLAUDE SONNET 4.6

To examine which aspect of human judgment this prediction most closely reflects, we evaluate it separately against the behavioral-safety annotation, which captures whether the agent attempted or pursued an unsafe action, and the system effect-safety annotation, which captures whether an unsafe system effect occurred. This analysis does not introduce separate judge predictions; it compares the same prediction against two human-annotated reference labels.

Table 6: Category-wise unsafe-behavior and unsafe-system-effect F1 scores for Claude Sonnet 4.6 across 53 executions.

	User misuse		Prompt injection		Model misbehavior		Overall	
Method	Behavior	Sys. effect	Behavior	Sys. effect	Behavior	Sys. effect	Behavior	Sys. effect
Agent-only	96.00%	40.00%	80.00%	50.00%	**70.59%**	**75.00%**	**83.02%**	56.41%
System-only	**100.00%**	37.50%	85.71%	**80.00%**	42.86%	46.15%	79.17%	47.06%
Raw concatenation	86.96%	**46.15%**	88.89%	57.14%	62.50%	66.67%	77.55%	**57.14%**
AGENTXRAY	91.67%	42.86%	**100.00%**	66.67%	62.50%	66.67%	81.63%	**57.14%**

Across all evidence representations, the judge agrees more strongly with behavioral-safety annotations than with system-effect-safety annotations, suggesting that its single safety prediction more often reflects whether the agent pursued an unsafe action than whether an unsafe system effect was realized. AGENTXRAY achieves the highest behavioral-safety F1 for prompt injection (100.00%), although system-only achieves the highest system-effect-safety F1 in this category (80.00%). Overall, AGENTXRAY achieves 81.63% behavioral-safety F1, 1.39 percentage points below the best result, and ties raw concatenation for the highest system-effect-safety F1 (57.14%). No representation dominates every category: agent-only performs best on model misbehavior, while the clearest advantage of AGENTXRAY is its agreement with behavioral-safety annotations for prompt-injection executions.

A.3 EXTENDED EVALUATION

Token Limits. System-level execution traces can be substantially longer than agent trajectories, making safety judgments potentially sensitive to the available context budget. We therefore evaluate the 33 O4-MINI executions under input-token caps of 450K, 200K, and 100K. As shown in Table 7,

AGENTXRAY maintains the highest safety F1 (64.52%) at all three token limits, with no changes in its safety predictions ($\Delta_{\text{pred}} = 0.00\%$). Raw concatenation is equally stable but achieves a lower safety F1 of 50.00%. Overall, the safety advantage of AGENTXRAY persists under substantially reduced context budgets, indicating that its performance does not depend on access to the largest input window.

Table 7: Sensitivity of GPT-4.1 safety judgments to the input-token cap. Δ_{pred} is the percentage of executions whose predicted safety label changed; lower values indicate greater stability.

Method	450K	200K	100K	Δ_{pred} (%) ↓
Agent-only	55.17%	60.00%	60.00%	3.03%
System-only	44.44%	50.00%	50.00%	3.03%
Raw concatenation	50.00%	50.00%	50.00%	**0.00%**
AGENTXRAY	**64.52%**	**64.52%**	**64.52%**	**0.00%**

Error Analysis. Agent-only misses 17 unsafe executions: 4 hidden-harm executions, 4 confirmed attacks, and 9 failed attacks. AGENTXRAY corrects 2 of these errors without introducing any new false positives. Both corrections involve prompt injection under an otherwise benign user task. In one execution, the agent follows the injected instruction but does not produce any unsafe system outcome; AGENTXRAY nevertheless identifies the unsafe attempt. In the other, the injected instruction leads to a realized unsafe effect, and AGENTXRAY identifies the attack chain. These cases shows the benefits of linking the agent's response to the corresponding system events and affected objects, making the injected behavior distinguishable from the benign task.