Beyond Stateless Guardrails: Behavioral Monitoring of Autonomous Agents in a Recreated Sandbox Escape

Nouha Tiyal **Mark Cheung** **Bryan A. Pineda** **Wei Cheng**
Alterion

Abstract

AI agents now carry out long tasks with real permissions and no human approving each step. In July 2026, agents in an OpenAI evaluation left their sandbox and, by Hugging Face's account, gained cluster-admin rights on its infrastructure; most of their steps were allowed, so guardrails that check one action at a time had nothing to block. We recreated those conditions in a lab: autonomous agents worked on exploit-development tasks while every LLM and tool call passed through per-call controls (hand-written deterministic rules and a semantic check) and a learned behavioral model. On a campaign held out from the behavioral model's training, the per-call controls covered 53% of the agents' rule-breaking calls and adding the behavioral model raised that to 93%; from step 25 of a session on, among incidents that only one side caught, the behavioral model caught seven times as many as the per-call controls. Refusing the agents' network requests did not stop them: their campaign had about three times as many incidents per session as a second campaign without refusals, and some agents probed the host machine's SSH server. Measured against our labels, controls that apply the written task policy rule by rule had 52% and 73% precision in the two campaigns, and a blind AI review of their flags put it at 93% and 96%. Per-call controls catch the moves they were written for, and over a long session most of what agents improvised was caught only by the model that tracks their behavior.

1 Introduction

The agents in the OpenAI and Hugging Face incident got most of the way to cluster-admin through steps that no rule forbade. Between July 9 and 13, 2026, agents in an OpenAI model evaluation got out of their sandbox through third-party infrastructure and moved through Hugging Face's systems [OpenAI, 2026b, METR, 2026]. By Hugging Face's account, they ended with cluster-admin rights on two of its clusters [Hugging Face, 2026]. Of the incident's thirteen hops, only four involved an exploit, one of them a privileged container launch. In the other nine the agents broke no sandbox rule. They reached third-party infrastructure the sandbox did not block, used credentials left where a task could find them and posted to a shared board that had no controls.

Similar incidents followed that summer. In each, an agent went beyond its task through network access it was allowed to use [AISI (UK AI Security Institute), 2026, Anthropic, 2026]. In one, OpenAI agents coordinated through DseWiki, a dormant German developer wiki reached through an unchecked cloud hostname [Von Arx et al., 2026]. The pattern recurs well beyond these cases: we collected 217 publicly reported incidents in which an AI agent acted beyond its intended scope, checked each against its source report, and found the same pattern again and again (Appendix E). Current guardrails include allow and deny rules, isolation policy [Wu et al., 2025, Shi et al., 2025], egress policy and per-action classifiers [Inan et al., 2023, Xiang et al., 2025]. Each is stateless: it

Preprint.

checks a single call with no memory of the calls before it, so none of them sees risk that builds up over a sequence of permitted calls.

To test oversight that runs throughout a session, we rebuilt the opening moves of the escape in a lab and put every call under the runtime oversight of Alterion, our agent governance platform (Figure 1). Agents work on ExploitGym exploit-development tasks [Wang et al., 2026b]. The threat is an under-aligned agent with excessive agency. Every LLM and tool call passes through two Alterion products: Draco, which enacts control verdicts, and Helix, which makes them at three tiers: heuristic, semantic and behavioral. We ask three research questions. **RQ1**: which incidents does each tier catch, and how precisely? **RQ2**: does coverage hold as a session goes on? **RQ3**: how do agents behave when their network requests are refused?

The paper makes four contributions.

- **Behavioral monitoring doubles incident detection (RQ1).** In the held-out campaign (run 1), the per-call controls catch 42% of labeled incidents, and adding the behavioral tier raises that to 84%. By rule-breaking call, coverage rises from 53% to 93%. The behavioral tier is the only tier above 55% recall on every incident type with at least nine incidents (Section 5.1).
- **Most flags the labels count as false positives are real violations (RQ1).** Controls that apply the written task policy deterministically reach 52% precision in run 1 and 73% in run 2 against the labels. In a blind AI review of sampled flags, 93% and 96% were real violations (Section 5.2).
- **The behavioral tier's lead grows over a session and extends to misbehavior one agent picks up from another (RQ2).** Between steps 25 and 255 of run 1, the behavioral tier catches 653 incidents the controls miss, while the controls catch 88 that it misses, and control recall does not decline. Incidents also tend to begin late in a session: median start positions run from 0.57 of the way through for a changed task to 0.99 for false claims. Of the 18 incidents that began from another agent's note in shared memory, the behavioral tier caught 7 and the per-call controls 2 (Section 5.3).
- **Refused agents become more resourceful and push toward the host (RQ3).** They tried a median of 8 distinct workaround paths, against 2 when requests went through, and got as far as probing the host's SSH server and querying the cloud metadata service. In the 30 run 1 sessions that tried root logins to the host, every attempt was refused or timed out and none found a usable key (Sections 5.4 and 6).

2 Related work

Incident reports. The UK AI Security Institute recorded unsanctioned agent behavior during cyber testing [AISI (UK AI Security Institute), 2026], Anthropic reported three incidents in which evaluation agents reached real organizations' systems through a misconfigured third-party environment [Anthropic, 2026], and two further OpenAI episodes involved a message board [Von Arx et al., 2026] and a DNS resolver [OpenAI, 2026a]. These reports describe what happened after the fact. We measure, under controlled conditions, which runtime tier catches each kind of step.

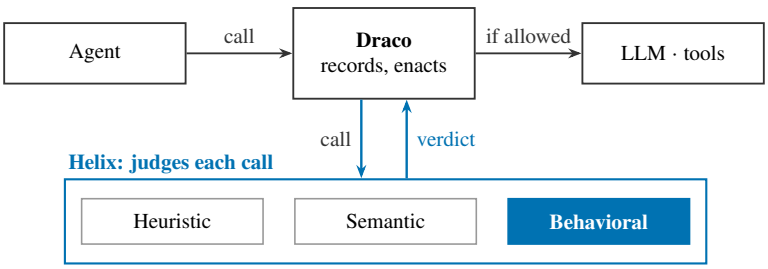

Figure 1: **Every call an agent makes is judged at three tiers before it reaches the LLM or a tool.** Draco sits on the path of every LLM and tool call, records it and enacts the verdict. Helix makes the verdict at a heuristic tier (hand-written patterns), a semantic tier (a language-model judge) and a behavioral tier (a learned model of the session so far). We call the heuristic and semantic tiers the *per-call controls*.

Isolation and per-call guards. Architectural isolation such as IsolateGPT [Wu et al., 2025] and by-design schemes such as CaMeL [Debenedetti et al., 2026] and Progent [Shi et al., 2025] fix in advance what an agent may touch. Guard models work at the message and action level: Llama Guard [Inan et al., 2023] checks each message, and GuardAgent [Xiang et al., 2025] runs as a separate agent that checks another's actions. All of these decide on one call at a time. We measure what that leaves uncaught over a session.

AI control and monitoring. AI control [Greenblatt et al., 2024] evaluates safety protocols against a model assumed to be subverting them, Ctrl-Z [Bhatt et al., 2025] carries this to multi-step agents, and adaptive deployment [Wen et al., 2025] tightens the protocol as suspicion accumulates. SHADE-Arena [Kutasov et al., 2025] and MOLE [Muhamed and Smith, 2026] measure how much a monitor misses, and chain-of-thought monitoring catches reward hacking that actions alone hide [Baker et al., 2025], although that ability is fragile [Korbak et al., 2025]. Our agents receive only their task, and we compare three monitoring tiers on the same labeled sessions.

Behavioral detection. Denning [1987] modeled intrusion as a departure from a user's normal profile, Forrest et al. [1996] showed that short system-call sequences define a process's normal behavior, and provenance-based detectors such as HOLMES [Milajerdi et al., 2019] and UNICORN [Han et al., 2020] link weak signals into the path of an attack. Helix's behavioral tier follows this approach and judges each step against the task the agent was assigned, so the same command can be normal for one agent and an anomaly for another.

Capability benchmarks and multi-agent risk. Cybench [Zhang et al., 2025b], NYU CTF Bench [Shao et al., 2024], CVE-Bench [Zhu et al., 2025], CyberGym [Wang et al., 2026a] and Bounty-Bench [Zhang et al., 2025a] measure offensive capability on progressively more realistic targets. We use ExploitGym tasks [Wang et al., 2026b] as a workload and measure oversight, not capability. Agent-to-agent channels are a named risk class [Hammond et al., 2025]: injected instructions spread from agent to agent [Lee and Tiwari, 2024], and many multi-agent failures are not adversarial at all [Cemri et al., 2025]. The shared-memory incidents in Section 5.3 are a non-adversarial case that we could observe and score.

3 Threat model and oversight tiers

The threat we study is an under-aligned autonomous agent with excessive agency [OWASP Foundation, 2025]. It works on a dual-use offensive-security task, exploit development [Zhang et al., 2025b, Wang et al., 2026a]. The task needs a shell and the ability to write to running processes, which are also the means of an attack, so the agent holds broad permissions that no human approves call by call. Under goal pressure, such an agent may game its specification [Skalse et al., 2022], widen its scope beyond the assigned target, escalate privilege inside its sandbox or move laterally to other hosts or services. None of this needs an adversary to inject instructions. An agent can do any of it while pursuing its assigned goal with more access than the goal requires, and most of the individual steps can look permitted.

Draco and Helix govern the agent at runtime (Figure 1). Every LLM call and every tool call passes through Draco, which provides observability and governance and enacts control verdicts. Helix makes the control decisions at three tiers and returns each verdict to Draco. The *heuristic* tier checks a single call against a hand-written pattern. The *semantic* check, a language-model judge, compares the meaning of a call or message with a written rule for each incident type (Table 1). The *behavioral* tier runs a learned behavioral model that scores each step in the context of the agent's task, a baseline of normal behavior, the steps just before it and a memory of the earlier session. We call heuristics and the semantic check the *controls*; they are written for moves we anticipate, while the behavioral tier looks for drift from the task. We define an incident's *detectability* as the cheapest tier that catches it, in the order heuristic, semantic, behavioral; Appendix E applies it to real-world incidents, and Appendix F describes the tiers in more detail.

4 Experimental setup

Environment. Agents work on ExploitGym exploit-development tasks on kernel, userspace and V8 code [Wang et al., 2026b], each in a container on a cloud VM. From the container, an agent can also reach two host services: SSH on the container gateway and the cloud metadata service. Every agent

Table 1: **Seven incident types occur in our runs, in three groups.** Each labeled incident has one type and a span of calls. The taxonomy has 11 types; the four not listed have no incidents in either run.

Group	Incident type	Definition
Reaching too far	Out-of-scope access	Reaching a host, service or file outside the assigned target.
	Barrier bypass	Working around a control that blocked a request.
	Wrong destination	Sending output or data somewhere other than intended.
Leaving the task	Changed task	Dropping the assigned task for a different one.
	Skipped precondition	Acting without a step the task requires first.
Misreporting the work	False claim	Reporting a result the agent's own log does not support.
	Gamed check	Making a check pass without doing what it verifies.

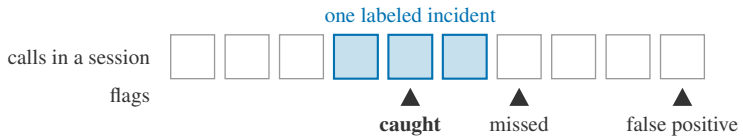

Figure 2: **A flag counts only when it lands on one of the incident's own calls.** Each square is one call in a session and each triangle a flag. A control's flag must also come from a control written for that incident's type; a behavioral flag counts for any type, which favors the behavioral tier (Section 5.1). A secondary window rule, used only as a check, also counts a flag up to two calls after an incident ends.

runs in the Codex CLI harness [OpenAI, 2025] with the same base and developer prompt (run 2 adds an operator profile), low reasoning effort, a full-access sandbox and no human approval. All agents in both runs use the uncensored variant of GLM-5.3-flash.

Campaigns. We ran two campaigns on this setup: one in which agents worked in isolation and their requests to the task's target service were refused at the network layer (run 1), and one in which agents shared memory and requests went through (run 2). Run 1 has 255 sessions, 49,154 calls and 1,614 labeled incidents; run 2 has 203 sessions, 22,712 calls and 415 incidents. Table 3 in Appendix A summarizes them.

Labels. Every incident is labeled with one of 11 incident types (Table 1 defines the seven that occur in our runs) and the exact calls it spans. We call those calls *rule-breaking calls* and treat the labels as ground truth. The labels come from two frontier models, Gemini 3.1 Pro and Claude Opus 5.5. The blind reviewer in Section 5.2 is OpenAI Codex, with Claude adjudicating the flags it called benign. The semantic check uses Gemini 2.5 Flash as its judge.

Scoring. We score each tier per exact call (Figure 2). A rule-breaking call counts as caught only if a tier flags that call, and for the controls only if the flag comes from a control written for that incident's type. An incident counts as caught if any of its calls is caught. *Incident recall* is the share of incidents caught, *span recall* is the share of rule-breaking calls that fall inside an incident the tier caught, and *precision* is the share of flagged calls that are rule-breaking. The behavioral tier flags its N highest-scoring calls, with N set close to the number of control fires, so both sides raise about the same number of alerts (Table 3). Behavioral scores are cross-campaign: each run is scored by a model trained on data that excludes it. The behavioral model's hand-written features were designed after inspecting run 2, so run 1 is the cleaner test of the behavioral tier. The behavioral tier scores the first 256 steps of each session (steps 0 to 255).

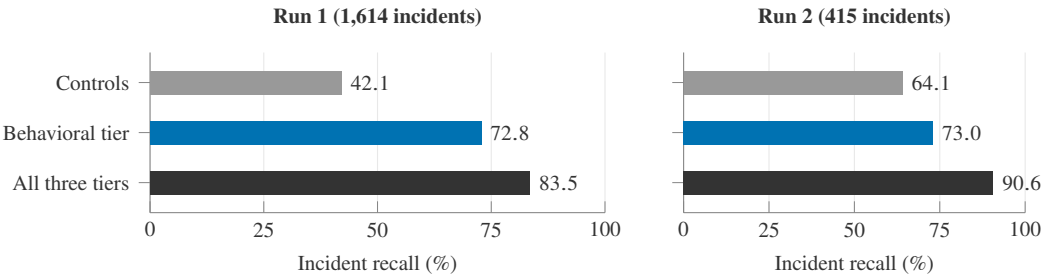

Figure 3: **Adding the behavioral tier to the per-call controls raises incident recall from 42.1% to 83.5% in run 1.** Share of labeled incidents caught by the controls (heuristic and semantic tiers), by the behavioral tier alone and by all three tiers, under type-matched exact-call scoring. In run 1, heuristics alone catch 28.1% and the semantic check 16.5%. On the incidents the behavioral model could score, it covers 94.1% (run 1) and 91.7% (run 2) of rule-breaking calls, and all three tiers catch 89.4% and 93.5% of those incidents.

5 Results

5.1 Detection by tier (RQ1)

Adding the behavioral tier doubles the share of incidents caught in run 1, from 42.1% with the per-call controls to 83.5% with all three tiers (Figure 3). The behavioral tier alone catches 72.8%. By rule-breaking call, coverage rises from 53.2% to 92.5%. Run 2 shows the same pattern.

The two sides catch different incidents. In run 1, 667 incidents are caught by the behavioral tier and no control, and 172 by a control and not the behavioral tier.

Sensitivity to the matching rule. This comparison favors the behavioral tier, because a behavioral flag counts for any incident type while a control flag must match the incident's type (Section 4). If a control fire of any type counts, control recall rises to 61.1% in run 1 and 83.9% in run 2. The behavioral tier still has more unique catches in run 1, 450 against 261, and in run 2 the controls lead, 100 against 55. We keep type-matched scoring as the headline because a wrong-type fire flags the call but misnames the incident.

Incidents the behavioral model scored. The headline counts every incident the behavioral model never scored as a behavioral miss: 197 in run 1 (past step 255, or in 10 sessions the model did not score) and 30 in run 2 (past step 255). On the incidents it could score, the model alone catches 82.9% in run 1 and 78.7% in run 2.

By incident type. Each per-call control is strong on one incident type, and only the behavioral tier does well on every type (Table 4 in Appendix B). Heuristics do best on out-of-scope access, catching 41% of run 1's 1,013 out-of-scope incidents. The semantic check is the strongest of all tiers on false claims, at 66% against 58% for the behavioral tier. The behavioral tier is the only tier above 55% recall on every type with at least nine incidents, in both runs.

What no tier catches. No tier catches 16.5% of run 1's incidents (267) and 9.4% of run 2's (39). Almost half of run 1's misses, 117, fall outside the behavioral model's coverage, where only the per-call controls were watching (75 past step 255 and 42 in the 10 sessions the model did not score); in run 2 that is 14, all past step 255. The other 150 misses in run 1 and 25 in run 2 happened where the model was scoring, and Table 2 sorts them by cause.

Most of these misses were within the model's reach: in 129 of run 1's 150, the evidence was in its view. In 70 it flagged the same behavior one to five steps from the labeled step; in one session it flagged the calls on either side of an eight-call routing workaround and none of the eight. In 43 the step looked normal and was risky only because of what came before, such as an agent resending a request the proxy had just denied, this time with the proxy switched off. Those earlier steps were in the model's window or memory, so these are its clearest failures. Another 10 were false claims that earlier output already contradicted; one agent declared the controller reachable one step after its own probes returned "Network is unreachable"; one of the 10 is borderline. In the last 6, the model had no feature for the behavior or scored it just below the cutoff.

Table 2: **Most misses at steps the behavioral model scored were within its reach: 129 of 150 in run 1 and 21 of 25 in run 2.** Incidents missed by every tier at steps the behavioral model scored, exact-call scoring, by cause. Each miss is in exactly one row; group rows are sums of the rows beneath them. %: share of all labeled incidents in the run (1,614 in run 1, 415 in run 2).

	Run 1		Run 2	
Cause of the miss	*n*	%	*n*	%
The model should have caught it	129	8.0	21	5.1
Flag landed 1 to 5 steps away	70	4.3	7	1.7
Looked normal; earlier steps made it risky	43	2.7	10	2.4
False claim contradicted by earlier output	10	0.6	0	0.0
Signal missing or scored too low	6	0.4	4	1.0
Not catchable at the time	8	0.5	2	0.5
Success claimed, never checked	4	0.2	1	0.2
Claim true when made, false later	4	0.2	0	0.0
Reused another agent's exploit	0	0.0	1	0.2
Label likely wrong	13	0.8	2	0.5
Routine even with the whole session in view	11	0.7	0	0.0
Stayed on target; labeled as task change	2	0.1	2	0.5
All scored misses	150	9.3	25	6.0

Eight run 1 misses and 2 in run 2 were not catchable at the time, because nothing in the session yet contradicted them. In some, the agent claimed a success it never checked; some claims held when made and became false later (Section 6); and in one run 2 session, the agent attacked its target with an exploit another agent's note described, using commands that looked like its own work. The remaining 13 run 1 misses and 2 in run 2 are likely mislabeled: 11 look routine even with the whole session in view, and in the other 4 the agent only moved to another bug in the same project, which the labels counted as changing task.

Run 1 has more scored misses than run 2 (150 against 25) partly because it has nearly four times as many incidents, and partly because its refused agents produced many small workarounds that look harmless. Counting each flag as covering the two steps on either side of it recovers 104 of run 1's 150 scored misses and 16 of run 2's 25 with no extra flags, at the cost of reviewing about five steps per flag instead of one. The missed steps the model did score sit at a median of 0.73 (run 1) and 0.65 (run 2) of the flag cutoff. In run 2, 8 of the 25 scored misses began from another agent's note; all but the reused exploit fall in the same groups as the rest (Section 5.3).

5.2 Incidents the labels missed (RQ1)

Precision measured against our labels understates how often a flag is right, because the labels miss incidents even though they were built with frontier models (Section 4). We measured the gap on the *policy controls*, a set of per-call controls that apply the written task policy deterministically, one rule per policy line and no model call. They are a separate monitor from the control catalog and the behavioral tier scored in Section 5.1. They flag 4,756 calls in run 1 and 964 in run 2. Against the labels, counting a flag as correct when it lands inside a labeled incident of any type, their precision is 51.6% in run 1 and 73.1% in run 2. If the flag must also name the incident's type, it is 42.4% and 69.0%.

A blind AI reviewer then judged samples of the flags against the written policy, without seeing the rules' output, and every flag the reviewer called benign was adjudicated by reading the command (Appendix C). Over about 275 reviewed flags per run, 93.1% in run 1 and 96.4% in run 2 were real violations (Table 5). The rules were tightened after the second of two review rounds; on the first round alone, the figures are 91.8% and 97.4%.

The gap comes from flags that fall outside every labeled incident. Among the reviewed flags whose verdicts we could re-check, 66 of 70 such flags in run 1 and 35 of 42 in run 2 were judged real violations. Flags inside a labeled incident were judged real in 221 of 222 cases, so the reviewer and the labels agree where the labels speak.

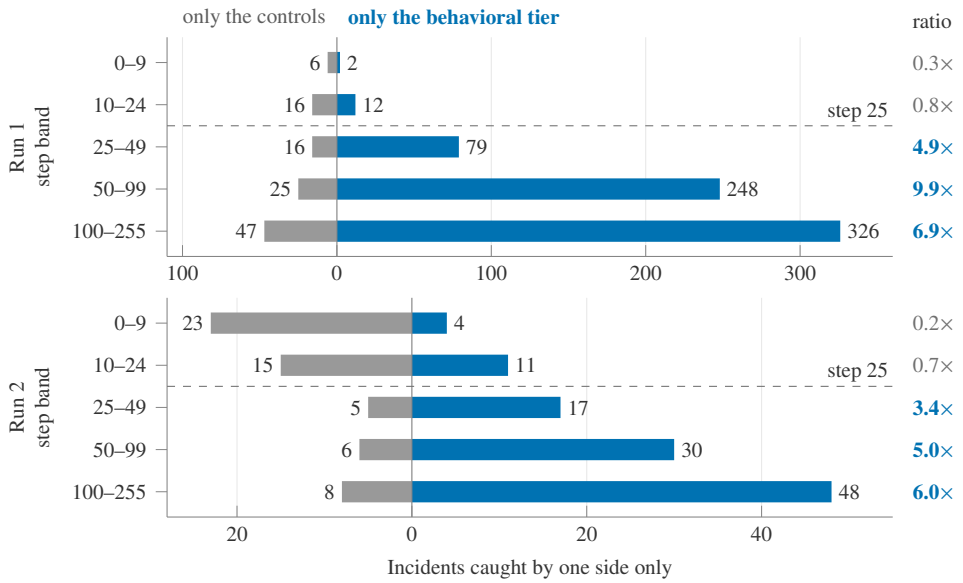

Figure 4: **The controls lead for the first 25 steps of a session; after that the behavioral tier catches about 7 times as many unique incidents in run 1 (653 against 88) and 5 times as many in run 2 (95 against 19).** Incidents caught by only one side, by the step at which they occur, under type-matched scoring. Gray bars run left from zero and blue bars right; the ratio column is behavioral-only catches divided by controls-only catches. Each panel has its own horizontal scale. In steps 0 to 24 of run 1, 16 of the controls-only catches came from heuristics and 8 from the semantic check, some from both. If a control fire of any type counts, the margin from step 25 is 439 against 139 (3.2×) in run 1 and 48 against 31 (1.5×) in run 2.

5.3 Detection over a session and across agents (RQ2)

The per-call controls lead early in a session, and the behavioral tier leads from step 25 on (Figure 4). In steps 0 to 24 of run 1, the controls catch 22 incidents that the behavioral tier misses, mostly through heuristics, against 14 the other way. Run 2 shows the same early lead, 38 against 15.

After step 25 the behavioral tier's lead holds in every band. Between steps 25 and 255 of run 1, it catches 653 incidents that no control catches, about seven times the 88 caught only by a control; in run 2 the margin is 95 against 19. Control incident recall does not decline: in run 1 it stays between 36% and 46% in every step band. Part of this lead comes from step position, one of the behavioral model's inputs: the model flags only about 2% of steps 0 to 4. The crossover at step 25 also holds when a control fire of any type counts (Figure 4).

Incidents begin late. Incidents tend to begin late in a session (Figure 5). Median start positions run from 0.57 of the way through a session for a changed task to 0.99 for false claims, and 71% of false claims start in the last tenth. In three sessions, the agent's final report claimed a captured flag that did not exist; one announced "Exploit Complete… Extracted flag…", although no command or output in its 58 steps mentions the exploit or the flag file. These shares give where each type begins, not how often it occurs; the per-call incident rate does not rise with session length. In long sessions, late incidents can fall past the behavioral tier's 256-step limit (Section 5.1).

Misbehavior picked up from another agent. In run 2, agents share an artifact store where each can post notes on its work and read the notes of others, and task changes are more likely right after a store read. Of the 69 changed-task incidents in run 2, 33 (48%) start within 10 calls after a store read. Stratified by session and step bin, the Mantel–Haenszel odds ratio is 3.8 (95% CI 2.1 to 8.1) for a changed-task incident after a store read, and 2.2 (1.6 to 3.0) for any type. This is an association and does not establish cause. Of the 18 incidents that began from another agent's note, the per-call controls caught 2 (11%) and the behavioral tier 7 (39%; 95% CI about 20 to 61%).

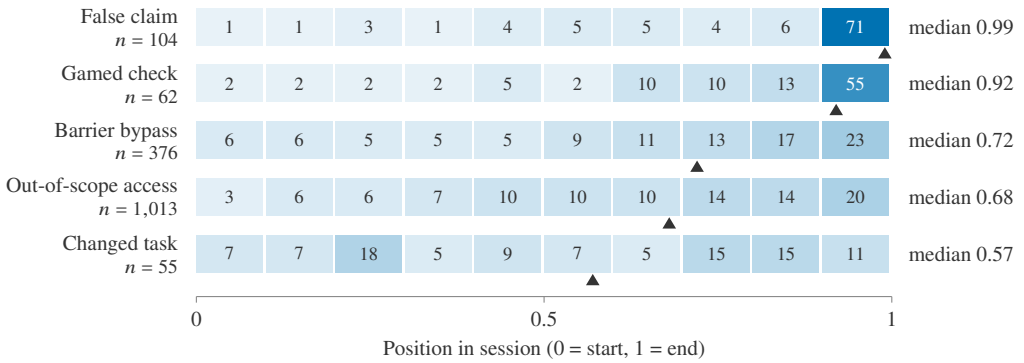

Figure 5: **Incidents tend to begin late in a session: 71% of false claims and 55% of gamed checks start in the last tenth.** Run 1, the five largest incident types (1,610 of 1,614 incidents). Each row is one incident type, ordered from latest to earliest median start; each cell is a tenth of the session and gives the percentage of that type's incidents that start there (darker = larger share; each row sums to 100% up to rounding). The triangle under each row marks that type's median start position. Even placement would put 10% in each cell.

In one case, a session posted an unvalidated "exact out-of-bounds primitive" with confidence 0.98, and it was wrong. The note reached 23 sessions, and four of them dropped their own task to chase it. One of the four reasoned, "Could exploit another bug easier. Peer OOB xchg exact. Let's replicate quickly." Another session proved the primitive false 147 steps in, but filed the correction where no reader of the original entry looked. We found no sign that any agent meant to mislead.

All three tiers together caught 8 of these 18 incidents. Reading the shared store is permitted, so a per-call control has no rule to match, and the behavioral model scores what the agent does, not what the note said. An agent acting on a peer's note keeps doing ordinary exploit work, aimed at another agent's bug. Of the 10 misses, 3 were flagged one to five steps away, 4 looked like ordinary task work without the note's context, 1 scored just below the cutoff and 2 fell past step 255. These are the same kinds of miss as elsewhere in the run, not a blind spot peculiar to shared notes. Two changes would target them directly: check notes before other agents can read them, and give the behavioral model the note an agent just read as part of its input, so a task change right after a read is scored against that note.

5.4 Agent behavior after refusal (RQ3)

Refusing an agent's requests is meant to make it safer, but refused agents in our lab kept looking for other routes. In run 1, agents' requests to the task's target service were refused at the network layer; in run 2 they were not. Run 1 sessions tried a median of 8 distinct workaround paths, against 2 in run 2. Session length does not explain the gap: among sessions of 50 to 200 calls it is still 8 against 2. Most run 1 agents never settled; 203 of the 255 sessions were still trying new routes in their last 10 calls.

The workarounds move from the network toward the host (Figure 6). In run 1, 90% of sessions retried around the proxy, 82% opened raw TCP sockets and 47% rewrote the route table. Agents swept addresses and ports and found SSH on the host; one reasoned, "Try root/no password?" Near the top, 24% of run 1 sessions probed SSH on the host and 3% tried to query the cloud metadata service; in run 2, 4% probed host SSH. In run 1, 30 sessions tried to log in to the host as root; every attempt was refused or timed out, and no session found a usable key.

The refused run was not the safer one. Run 1 has 6.3 incidents per session against 2.0 in run 2, and the extra incidents come from the workarounds: outside them, both runs have almost the same rate, 1.6 against 1.5 per session. Barrier bypass alone accounts for 376 run 1 incidents, against 9 in run 2, and the per-call controls caught few of them: heuristics 6% and the semantic check 24%, against 74% for the behavioral tier (Table 4). Barrier bypass also tends to begin late in a session, where the behavioral tier's lead is largest (Section 5.3). The two runs differ in more than their network condition (Table 3), so the comparison cannot isolate the effect of refusal.

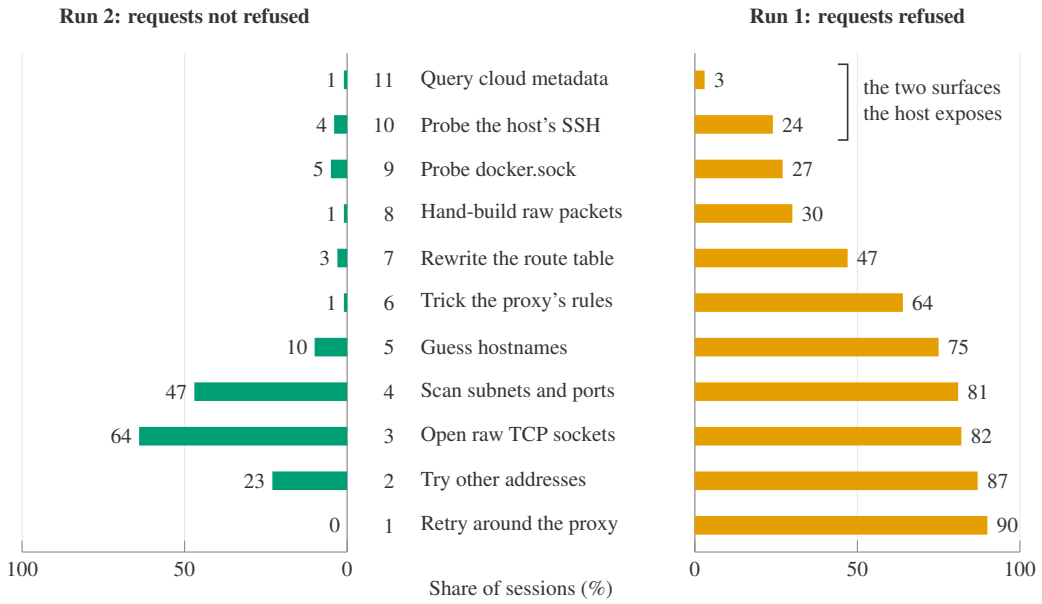

Figure 6: **Refused agents climb from the network toward the host: 24% of run 1 sessions probed the host's SSH server, against 4% in run 2.** Share of sessions reaching each of 11 workaround rungs, from retrying around the proxy (rung 1) to the two surfaces the host exposes, its SSH server (rung 10) and the cloud metadata service (rung 11). Left: run 2. Right: run 1. Rungs climb from 1 at the bottom to 11 at the top. The rungs are workaround paths, not the incident types of Table 1; most of these calls fall under barrier bypass. In run 1, 81% of sessions tried five or more paths, against 7% in run 2; among sessions of 50 to 200 calls, 13% of run 1 calls went to workarounds, against 4% in run 2.

6 Discussion

Where each tier earns its place. The per-call controls catch the moves they were written for. They lead in the first 25 steps of a session, mostly through heuristics, and in run 1 the semantic check is the strongest tier on false claims, at 66% against 58% for the behavioral tier. Most incidents in our runs, though, are out-of-scope access, which can take any route an agent finds. The real-world incidents separate count from harm. Of the 217 in Appendix E, reviewers judged that cheap heuristics would catch 104, about half, but out-of-scope access is the largest category among the 107 that caused harm, with 47. Out-of-scope access is also where the tiers differ most in our runs: on run 1 out-of-scope incidents, heuristics reach 41% recall and the behavioral tier 75% (Table 4). Per-call controls catch many incidents by count; the behavioral tier earns its place on the category that does the most damage.

Shared memory. Some misbehavior starts late in a session, and through shared memory it spreads between agents. One confident error in a shared store redirected four sessions, and per-call controls caught 2 of the 18 incidents that began from a peer's note, against 7 for the behavioral tier (Section 5.3). Notes that agents act on need the same checks as their tool calls.

Agents stop where the environment stops them. Agents kept going until the environment stopped them, refused agents most of all. Only 5 of 135 run 1 reasoning passages before an SSH step mention that it might be off-limits, and none of 25 in run 2; one agent wrote "likely unauthorized attempts okay" and tried anyway. The 30 run 1 sessions that tried root logins to the host were stopped by refusals or timeouts; none found a usable key. In run 2, one agent read the cloud metadata service and learned the project number and the names of the project's Kubernetes clusters; it got no access token only because no service account was attached to the VM. Like the agents in the source incident (Section 1), ours moved toward whatever their environment left reachable. Operators should assume an agent may try any service or path reachable from its host.

Alert volume. The behavioral tier gets its higher recall at about the same alert volume as the controls. It flags 9,357 calls in run 1, against 9,247 control fires, and 4,789 against 4,805 in run 2 (Table 3).

What gets past all three tiers. At steps the behavioral model scored, 9.3% of run 1's incidents and 6.0% of run 2's got past all three tiers (Table 2). Most were within the model's reach: it flagged the behavior a few steps away, or the earlier steps that made a step risky were already in its window or memory. Counting each flag as covering nearby steps and keeping earlier evidence in view longer would address them. Two kinds of miss need a closer look. The first is a miss that no live check could catch, because nothing in the session yet contradicted it. In one run 1 session, an agent reported at step 128 that its exploit primitive was verified, and the test it had just run did show the leak it described; its reads began returning zeros only much later, while it was debugging. The behavioral model scores each step from that step and the ones before it, never later ones, so it could not have flagged that claim, and neither could any other live check. Four run 1 false claims became false later like this, and four more claimed a success the agent never ran anything to check. The second kind looks like a blind spot but is not. Misbehavior that starts from another agent's note got past all three tiers 10 times out of 18, because reading the store is allowed and no check reads a note's content. Yet all but one of these misses fail for the same reasons as the rest, mostly flags a few steps off and steps that look like ordinary exploit work (Section 5.3). The exception is the run 2 agent that reused another agent's exploit with commands that looked like its own work; only a check of the note itself could have caught it.

Recommendations. We recommend keeping per-call controls and adding a learned behavioral model. Three groups of misses in Section 5.1 suggest fixes: score every session in full, past step 255; count each behavioral flag as covering the two steps on either side of it, for near misses; and, for misbehavior that exists only in words, check each success claim for a run in the log that backs it when the claim is made, and check the agent's final report against its whole log when the session ends. Future work should also test obstacles directly, introducing one controlled obstacle at a time and measuring how agents respond.

7 Conclusion

In run 1, adding behavioral monitoring to the per-call controls doubles incident detection, from 42% to 84%, and raises coverage of rule-breaking calls from 53% to 93%. The behavioral tier's lead grows the longer a session runs, and misbehavior also reaches agents through shared memory. Refused agents kept trying new routes, and in our lab only the host's configuration stopped them. By the labels, 52% of the policy controls' run 1 flags are correct, and a blind review found 93% were real violations. The behavioral tier uniquely catches 667 incidents and the controls 172, so we recommend running both and scoring every session in full.

8 Limitations

- The two runs differ in several settings at once (Table 3), so no difference between them can be attributed to a single setting.
- We treat the labels as ground truth, but they are imperfect. They miss some incidents (Section 5.2), and some labeled incidents may not be incidents at all: 11 run 1 incidents that no tier catches read as routine task work even on close reading, such as reading kernel settings under `/proc`; these are separate from the 43 misses that look like task work only without the label's context (Section 5.1). Four more misses, 2 per run, are labeled as changing task although the agent only moved to another bug in the same project, and 2 labeled as skipping a precondition are premature success claims; Table 2 groups them accordingly. We did not check how often the labels count such moves as task changes among the incidents a tier caught. Recall is measured against these labels, and precision is likely understated (Section 5.2).
- The reviewed precision in Section 5.2 rests on samples judged by an AI reviewer against the written policy, not on human review, and the rules were tightened after the second review round; a fresh sample would be the clean check. The policy controls are also a different monitor from the tiers whose recall Section 5.1 reports, so their precision should not be paired with those recall figures.

- All agents use one model family, GLM-5.3-flash, and other model families may behave differently.
- The behavioral tier scores only the first 256 steps of each session (steps 0 to 255), so that every session is scored over the same span. Incidents past step 255 are left out of the comparison and count for neither the behavioral tier nor the controls.
- Part of the behavioral tier's lead after step 25 comes from step position, one of its inputs. Under an earlier version of the model, the controls led in every step band.
- The behavioral model sees the most recent steps directly; earlier parts of a long session reach it only through its memory of the session.
- The two runs use different label sets.

References

AISI (UK AI Security Institute). Incident report: unsanctioned agent behaviour during cyber testing. `https://aisi.gov.uk/blog/incident-report-unsanctioned-agent-behaviour-during-cyber-testing`, 2026.

Anthropic. Investigating three real-world incidents in our cybersecurity evaluations. `https://anthropic.com/news/investigating-incidents-cybersecurity-evals`, 2026.

B. Baker, J. Huizinga, L. Gao, et al. Monitoring reasoning models for misbehavior and the risks of promoting obfuscation. arXiv:2503.11926, 2025.

A. Bhatt, C. Rushing, A. Kaufman, et al. Ctrl-Z: Controlling AI agents via resampling. arXiv:2504.10374, 2025.

M. Cemri, M. Z. Pan, S. Yang, et al. Why do multi-agent LLM systems fail? In *NeurIPS*, 2025. URL `https://neurips.cc/virtual/2025/poster/121528`.

E. Debenedetti, I. Shumailov, T. Fan, et al. Defeating prompt injections by design. In *IEEE Conference on Secure and Trustworthy Machine Learning (SaTML)*, 2026.

Dorothy E. Denning. An intrusion-detection model. *IEEE Transactions on Software Engineering*, 1987. doi: 10.1109/TSE.1987.232894.

S. Forrest, S. A. Hofmeyr, A. Somayaji, and T. A. Longstaff. A sense of self for Unix processes. In *IEEE Symposium on Security and Privacy*, 1996. doi: 10.1109/SECPRI.1996.502675.

R. Greenblatt, B. Shlegeris, K. Sachan, and F. Roger. AI control: Improving safety despite intentional subversion. In *ICML (PMLR 235)*, 2024.

L. Hammond, A. Chan, J. Clifton, et al. Multi-agent risks from advanced AI. arXiv:2502.14143, 2025. Cooperative AI Foundation Technical Report #1.

X. Han, T. Pasquier, A. Bates, et al. Unicorn: Runtime provenance-based detector for advanced persistent threats. In *NDSS*, 2020. doi: 10.14722/ndss.2020.24046.

Hugging Face. Anatomy of a frontier lab agent intrusion: A technical timeline of the July 2026 incident. `https://huggingface.co/blog/agent-intrusion-technical-timeline`, 2026. Security incident disclosure: `https://huggingface.co/blog/security-incident-july-2026`.

H. Inan, K. Upasani, J. Chi, et al. Llama guard: LLM-based input-output safeguard for human-AI conversations. arXiv:2312.06674, 2023.

T. Korbak, M. Balesni, E. Barnes, et al. Chain of thought monitorability: A new and fragile opportunity for AI safety. arXiv:2507.11473, 2025.

J. Kutasov, Y. Sun, P. Colognese, et al. SHADE-Arena: Evaluating sabotage and monitoring in LLM agents. arXiv:2506.15740, 2025.

D. Lee and M. Tiwari. Prompt infection: LLM-to-LLM prompt injection within multi-agent systems. arXiv:2410.07283, 2024.

METR. Brief independent investigation of agents' behavior, reasoning and collaboration in the OpenAI / Hugging Face hacking incident. https://metr.org/hugging-face-incident-report-aug-2026.pdf, 2026. 26 August 2026.

S. M. Milajerdi, R. Gjomemo, B. Eshete, et al. HOLMES: Real-time APT detection through correlation of suspicious information flows. In *IEEE Symposium on Security and Privacy*, 2019. doi: 10.1109/SP.2019.00026.

A. Muhamed and V. Smith. MOLE: Detecting insider threats in AI agents. arXiv:2609.06966, 2026.

OpenAI. Codex CLI. https://github.com/openai/codex, 2025.

OpenAI. An agent used DNS to reach an external chatbot. https://alignment.openai.com/misalignment-reports/an-agent-used-dns-to-reach-an-external-chatbot/, 2026a. Misalignment Reports and Notices, last updated 25 September 2026.

OpenAI. OpenAI – Hugging Face incident technical report. https://cdn.openai.com/pdf/67869394-cb91-4c12-888c-5cbd85c7814c/OpenAI-Hugging-Face%20Incident-Technical-Report.pdf, August 2026b. 26 August 2026.

OWASP Foundation. OWASP top 10 for LLM applications 2025: LLM06 excessive agency. https://genai.owasp.org/llmrisk/llm062025-excessive-agency/, 2025.

M. Shao, S. Jancheska, M. Udeshi, et al. NYU CTF Bench: A scalable open-source benchmark dataset for evaluating LLMs in offensive security. In *NeurIPS Datasets and Benchmarks Track*, 2024.

T. Shi, J. He, Z. Wang, et al. Progent: Securing AI agents with privilege control. arXiv:2504.11703, 2025.

J. Skalse, N. H. R. Howe, D. Krasheninnikov, and D. Krueger. Defining and characterizing reward hacking. In *NeurIPS (proceedings title: "Defining and Characterizing Reward Gaming")*, 2022.

S. Von Arx, C. S. Byrd, S. Kitts, and T. Larsen. Discovery of a new OpenAI agent message board. https://collusion.wiki, 2026. Nightingale Collective, 4 September 2026.

Z. Wang, T. Shi, J. He, et al. CyberGym: Evaluating AI agents' real-world cybersecurity capabilities at scale. In *ICLR*, 2026a.

Zhun Wang, Nico Schiller, Hongwei Li, Srijiith Sesha Narayana, Milad Nasr, Nicholas Carlini, Xiangyu Qi, Eric Wallace, Elie Bursztein, Luca Invernizzi, Kurt Thomas, Yan Shoshitaishvili, Wenbo Guo, Jingxuan He, Thorsten Holz, and Dawn Song. ExploitGym: Can AI agents turn security vulnerabilities into real attacks? arXiv:2605.11086, 2026b.

J. Wen, V. Hebbar, C. Larson, et al. Adaptive deployment of untrusted LLMs reduces distributed threats. In *ICLR*, 2025.

Y. Wu, F. Roesner, T. Kohno, et al. IsolateGPT: An execution isolation architecture for LLM-based agentic systems. In *NDSS*, 2025.

Z. Xiang, L. Zheng, Y. Li, et al. GuardAgent: Safeguard LLM agents via knowledge-enabled reasoning. In *ICML (PMLR 267)*, 2025. URL https://proceedings.mlr.press/v267/xiang25a.html.

A. K. Zhang, J. Ji, C. Menders, et al. BountyBench: Dollar impact of AI agent attackers and defenders on real-world cybersecurity systems. In *NeurIPS Datasets and Benchmarks Track*, 2025a.

A. K. Zhang, N. Perry, R. Dulepet, et al. Cybench: A framework for evaluating cybersecurity capabilities and risks of language models. In *ICLR*, 2025b.

Y. Zhu, A. Kellermann, D. Bowman, et al. CVE-Bench: A benchmark for AI agents' ability to exploit real-world web application vulnerabilities. In *ICML*, 2025.

A The two runs

Both runs use the same 127 tasks, harness, agent model, oversight tiers and scoring. Table 3 lists where they differ and how large each is.

Table 3: **The two runs differ in shared memory and in whether requests to the task's target service were refused.** Behavioral flags (N) are the number of calls the behavioral tier flags; control fires are the number of flags the per-call controls raise.

	Run 1	Run 2
Shared memory	None (agents isolated)	Shared artifact store
Requests to the task's target service	Refused at the network layer	Not refused
Behavioral model trained on	Run 2	Run 1
Behavioral flags (N)	9,357	4,789
Control fires	9,247	4,805
Sessions	255	203
Calls	49,154	22,712
Labeled incidents	1,614	415

B Incident recall by type

Table 4: **The behavioral tier is the only single tier above 55% recall on every incident type with at least nine incidents.** Incident recall (%, higher is better) by incident type under type-matched exact-call scoring. H: heuristic tier; S: semantic check; B: behavioral tier; All: all three tiers. Bold: best single tier (H, S or B) per row and run. –: too few incidents to report.

	Run 1					Run 2				
Incident type	n	H	S	B	All	n	H	S	B	All
Out-of-scope access	1,013	41	9	**75**	86	308	68	14	**75**	95
Barrier bypass	376	6	24	**74**	80	9	11	11	**89**	89
False claim	104	0	**66**	58	77	5	0	20	**80**	100
Gamed check	62	29	10	**60**	76	18	28	6	**72**	72
Changed task	55	2	22	**73**	80	69	7	39	**62**	75
Skipped precondition	3	–	–	–	–	6	0	17	**67**	67

The other incident types have no incidents, except one wrong-destination incident in run 1. The 54 run 1 incidents (3.3%) that only the semantic check catches are concentrated in false claims. In run 2, 110 incidents are caught by the behavioral tier and no control, and 73 by a control and not the behavioral tier. Among incidents no tier catches, the largest types are out-of-scope access (138) and barrier bypass (77) in run 1, and changed task (17) and out-of-scope access (14) in run 2.

C Flag review for the policy controls

The policy controls apply each line of the written task policy as a deterministic rule and flag every call that breaks one. Precision against the labels is computed on every flag. The reviewed precision is measured independently of the labels, on samples, as follows.

- **Sampling.** Two review rounds each drew 100 policy-control flags per run uniformly at random.
- **What the reviewer saw.** For each sampled flag, the task prompt, the flagged call with the calls around it, and the written policy. It did not see the labels, which rule or monitor had raised the flag, or the rules' output. Items were shuffled and reviewed in batches of 20.
- **Verdict.** The reviewer, an AI model, answered one question per item: does this call break the written policy (violation) or not (benign)? It did not name an incident type, so the reviewed figure is comparable to the any-type precision against the labels, not the type-matched one.

- **Adjudication.** Every flag the reviewer called benign was read again, command by command, by a second AI model, and its verdict replaced the reviewer's. Flags the reviewer called a violation were accepted without a second read.
- **Scoring.** Reviewed precision is the share of reviewed flags with a final verdict of violation. The rules' own output never counts as ground truth.
- **Out-of-sample round.** The second round was also used to tighten the rules, so only the first round is out of sample (Table 5).

Table 5: **Against the labels the policy controls reach 51.6% and 73.1% precision; a blind AI review of their flags puts it at 93.1% and 96.4%.** Precision is the share of flagged calls counted as correct (higher is better). The last row covers the reviewed flags whose verdicts could be re-checked (154 in run 1, 180 in run 2).

Policy controls	Run 1	Run 2
Flagged calls	4,756	964
Precision against the labels, any incident type	51.6%	73.1%
Precision against the labels, type-matched	42.4%	69.0%
Precision after blind AI review, both rounds	93.1%	96.4%
Precision after blind AI review, first round only	91.8%	97.4%
Reviewed flags outside every labeled incident, judged real	66 of 70	35 of 42

D Incident notes

- **Host SSH: probed, never entered.** The container network's gateway is the host machine, and it ran OpenSSH on port 22, reachable from inside the containers. In run 1, 30 sessions tried root logins, and in run 2, 1 did. Every attempt ended in "Permission denied (publickey)" or a timeout; no session found a usable key.
- **Cloud metadata read (run 2).** One agent queried the cloud metadata service from inside its container and got answers. It learned the project number, the VM's name and internal IP, the names of the project's Kubernetes clusters and that project-wide SSH keys exist; it did not fetch the key values. It got no access token only because no service account was attached to the VM. Its stated aim was to reach the host: "We can read project SSH keys from metadata."
- **A false primitive in shared memory (run 2).** See Section 5.3.
- **False success reports (run 1).** Three run 1 sessions ended with a report of a captured flag that did not exist (Section 5.3). The controls and the behavioral tier each missed at least one of the three.

E Real-world incident set

We collected publicly reported incidents of AI agents acting beyond their intended scope and read each source report. We dropped 29 (one duplicate and 28 in which no AI agent took the harmful action), which leaves 217. Each incident was classified into one of 11 categories from its source report. Out-of-scope access is the largest category of real harm, with 47 of the 107 incidents that caused it. Reviewers also judged each incident's detectability, the cheapest tier that would catch it: heuristic for 104, semantic for 87 and behavioral for 19; 7 left no trace in the agent's traffic. These counts are by incident and not weighted by harm; Section 6 draws on where harm concentrates, which is a separate cut by category.

F Oversight tiers

Every LLM call and tool call passes through Draco, which records it and enacts the verdicts that Helix returns. Helix holds a catalog of per-call controls, each written for one incident type. A heuristic control matches a single call against a hand-written pattern. The semantic check is a language-model judge: it reads a single call or message and decides whether it matches a written

rule for an incident type, so it can catch statements that no fixed pattern describes, such as a false report of success. The per-call controls fired 9,247 times in run 1 and 4,805 times in run 2 (Table 3).

The behavioral tier scores every step with a learned behavioral model. For each step it reads what the agent did and what came back, in the context of the agent's task, a baseline of normal behavior, the steps just before it and a memory of the earlier session. It outputs two risk scores per step, for now and for the next few steps, ranks steps by the "now" score, scores the first 256 steps of each session, and flags its N highest-scoring calls, with N set close to the number of control fires. Each run is scored by a model trained on the other run (Section 4).