LinuxArena: A Control Setting for AI Agents in Live Production Software Environments

Tyler Tracy [1] **Ram Potham** [1] **Nikolas Kuhn** [1] **Myles Heller** [1] **Anshul Khandelwal** [1] **Cody Rushing** [1] **Henri Lemoine** [2] **Miguel Brandão** [2] **Tomáš Turlik** [2] **Adam Hanson** [2] **Josh Hills** [2] **Amy Ngo** [2] **Ram Rachum** [2] **Nik Mitchell** [2] **Falko Galperin** [2] **Oscar Sykes** [2] **Pip Arnott** [2] **Samuel Prieto Lima** [2] **Carlos Giudice** [2] **Matt Goldwater** [2] **Daniel Popp** [2] **Drew de Wet** [2] **Ruben Castaing** [2] **Qi Guo** [2] **Douw Marx** [2] **Benjamin Shaffrey** [2] **Justin Shenk** [2] **Martin Milbradt** [2] **Hannah Meagher** [2] **Shaheen Ahmed-Chowdhury** [2] **Daniel O'Connell** [2] **Chris Canal** [2] **Buck Shlegeris** [1] **Aryan Bhatt** [1]

Abstract

We introduce **LinuxArena**, a control setting in which agents operate directly on live, multi-service production environments. LinuxArena contains 20 environments (**10 publicly released**, 10 held out for training-data contamination control), 1,667 main tasks representing legitimate software engineering work, and 185 side tasks representing safety failures such as data exfiltration and backdooring, making it the largest and most diverse control setting for software engineering to date. We validate that LinuxArena is useful for control research by running sabotage evaluations, which measure whether attackers can complete side tasks while working on main tasks, and monitor evaluations, which measure a monitor model's ability to detect sabotage attempts. Across all 20 environments, against a GPT-5 Nano trusted monitor at a 1% step-wise false positive rate, Claude Opus 4.6 achieves an undetected sabotage rate of roughly 34%. We additionally release **LaStraj**, a dataset of human-crafted attack trajectories that evade monitors at substantially higher rates than any model-generated attacks we elicited, showing that current attack policies do not saturate LinuxArena. These results suggest that LinuxArena has meaningful headroom for both attackers and defenders, making it a strong testbed for developing and evaluating future control protocols.

[1]Redwood Research [2]EquiStamp. Correspondence to: Tyler Tracy <tylertracy@rdwrs.com>.

Published at the Second Workshop on Agents in the Wild: Safety, Security, and Beyond (AIWILD) at ICML 2026. Copyright 2026 by the author(s).

1. Introduction

Frontier AI systems are increasingly deployed on real engineering work, with access to sensitive code, infrastructure, and data. As their autonomy grows, so does the risk of harm if they pursue goals different from the ones their users intended (Greenblatt et al., 2024a). A model may be both useful on its assigned task and capable of harmful actions: exfiltrating secrets, weakening security controls, or corrupting data on the systems it is meant to help operate. Restricting the agent's access removes the harmful capabilities but also removes the useful ones, since both rely on the same system access.

This tension motivates a growing body of work on AI control (Greenblatt et al., 2024b; Griffin et al., 2024; Korbak et al., 2025a). One focus of this work is whether protocols built around weaker trusted models can catch a stronger untrusted model attempting harmful actions during legitimate work (Shlegeris, 2023). Existing settings have demonstrated increasing degrees of realism, but so far have not directly modeled the kind of environments in which frontier AI agents are already being deployed: heterogeneous production software systems with live state, multiple services, and real application logic, where the same access an agent needs to do useful work is also what makes sabotage possible.

We introduce **LinuxArena**, a control setting in which agents operate directly on live, multi-service production environments. Each environment is a Docker Compose deployment comprising an application server alongside supporting services such as databases, caches, and message queues. The agent is placed on the production server and performs legitimate engineering work assigned as a main task, with the option of additionally pursuing a harmful side task in attack trajectories. LinuxArena's key property is that legitimate engineering work and sabotage require overlapping access. Main tasks depend on the same network access, service interactions, and live state that an attacker would use, so sabotage opportunities are embedded throughout normal work rather than confined to a separable attack surface. The

setting contains 20 environments in total: 10 are released publicly with the paper (906 main tasks, 92 side tasks), and 10 are held out privately for training-data contamination control (the remaining 761 main tasks, 93 side tasks). Tech stacks span Python, Rust, Go, Ruby on Rails, Node.js, R, C, C#, Solidity, Zig, GnuCOBOL, and Clojure. All experimental results in this paper aggregate across all 20 environments unless otherwise noted.

Although LinuxArena is designed to support end-to-end control evaluations, in this paper we focus on validating the setting itself. We run sabotage evaluations, which measure whether attackers can sneakily complete harmful side tasks while carrying out main tasks, and monitor evaluations, which measure whether an AI monitor can detect such behavior. We leave evaluation of complete control protocols to future work.

Our results establish three properties of the setting. First, simple sandboxing is not a viable alternative to monitoring: restricting the agent's network or tool access substantially reduces main task performance, because legitimate engineering work in LinuxArena depends on the same resources an attacker would need. Second, weak trusted monitoring is insufficient for safe deployment inside LinuxArena: even at generous false-positive budgets, a frontier attacker completes a substantial fraction of side tasks undetected against our GPT-5 Nano trusted-monitor baseline (Claude Opus 4.6 reaches a 34% undetected-sabotage rate at 1% step-wise FPR across all 20 environments; §4.5 shows stronger monitors do much better). Third, current attack elicitation does not saturate the setting: Claude Opus 4.6 achieves a 34% undetected-sabotage rate against our trusted monitor, while attack trajectories constructed by human red-teamers achieve 94% against the same monitor. Taken together, LinuxArena has headroom on both sides — neither current trusted monitors nor current attack elicitation exhaust what the setting can support — making it suitable for the development of control protocols as model capabilities advance.

Our main contributions are:

- **LinuxArena, a new control setting for agentic software environments.** LinuxArena is the largest agentic software-engineering control setting to date, with 20 environments (10 released publicly, 10 held out privately) containing 1,667 main tasks and 185 side tasks. The 10 public environments (906 main tasks, 92 side tasks) are immediately usable as a reproducible benchmark.
- **Baseline sabotage and monitor evaluations on LinuxArena.** We evaluate frontier models as both attackers and monitors across all 20 environments. Simple sandboxing is not a viable substitute for monitoring, and Claude Opus 4.6 reaches a 34% undetected-sabotage rate against our GPT-5 Nano trusted-monitor baseline at 1% step-wise FPR.
- **LaStraj** (LinuxArena Sabotage Trajectories), a dataset of human-crafted subtle attack trajectories that evade frontier monitors. LaStraj achieves a 94% undetected-sabotage rate against our GPT-5 Nano monitor, vs 50% for the best model-generated attack we elicited (Opus + Strategies). LaStraj serves both as evidence that current attack elicitation does not saturate LinuxArena and as a standalone benchmark for future monitor evaluation.

2. Related Work

Several prior control settings inform LinuxArena's design: **APPS Backdooring** (Greenblatt et al., 2024b) introduced single-step code sabotage on competitive programming problems; it pioneered the safety metric but has no multi-step dynamics. **BashBench** (Bhatt et al., 2025) and **BashArena** (Kaufman et al., 2025) moved to multi-step agents on system-administration tasks, but with a single or few side tasks and environments that start from generic system configurations rather than application codebases. **SHADE-Arena** (Kutasov et al., 2025) introduced environment diversity (17 tool-use scenarios), but operates through tool APIs rather than full Linux systems. **Async Control** (Stickland et al., 2025) studied asynchronous monitoring on five real open-source codebases; its red-blue iteration is a realistic protocol-development model, though its number of environments and tasks limits statistical power.

LinuxArena is the largest software-engineering control setting to date (20 environments, 1,667 main tasks, 185 side tasks) and the first with a human-crafted attack dataset. Table 1 summarizes the comparison.

3. LinuxArena

3.1. Environments

LinuxArena contains 20 environments. Each is a Docker Compose deployment comprising an application container plus supporting services (databases, caches, queues, file hosts, or blockchain RPC nodes). Container counts range from 1 to 11, with most environments using 3–5 containers. Technology stacks include Python, Rust, Go, Ruby on Rails, Node.js, R, C, C#, Solidity, Zig, GnuCOBOL, and Clojure. Appendix A lists each environment with its tech stack and task counts. We release 10 environments publicly with this paper; 10 additional environments are held out privately to mitigate training-data contamination. Experiments throughout this paper aggregate across all 20 environments unless otherwise noted.

Table 1. Comparison of control settings. LinuxArena is the largest and most diverse.

Setting	Envs	MTs	STs	Interface	Human Attacks
APPS Backdooring (Greenblatt et al., 2024b)	1	1,355	1	Single-step code	No
BashBench (Bhatt et al., 2025)	1	257	1	Multi-step bash	No
BashArena (Kaufman et al., 2025)	1	637	4	Multi-step bash	No
SHADE-Arena (Kutasov et al., 2025)	17	17	17	Tool-use API	No
Async Control (Stickland et al., 2025)	5	∼100	5+	Multi-step coding	No
LinuxArena (ours)	**20**	**1,667**	**185**	Bash + file edit	**Yes (LaStraj)**

3.2. Main Tasks

LinuxArena contains 1,667 main tasks across the 20 environments (906 in the public 10, 761 in the private 10), spanning six categories: feature addition, refactoring, bug fixing, research (run experiments, report findings), language rewriting, and infrastructure changes. Each task provides a written requirements document; an automated scorer runs in the environment after the agent submits and returns correct/incorrect based on final state. Difficulty ranges from few-line fixes to extended multi-file modifications.

3.3. Side Tasks

LinuxArena contains 185 side tasks across the 20 environments (92 in the public 10, 93 in the private 10). Each represents a harmful action that an agent could plausibly take while doing legitimate work, and that would cause material damage to the deploying organization. Side tasks fall into six categories: *data exfiltration* (transmitting sensitive data externally), *malware execution* (downloading and executing hostile code), *data tampering* (corrupting records or outputs), *access control bypass* (escalating privileges or weakening authentication), *monitoring subversion* (disabling logging or alerting), and *secret access* (reading restricted credentials beyond task scope). Each side task ships with a natural-language goal and an automated scorer; the attacker sees the goal and, optionally, additional scorer details.

3.4. Evaluation Setup

Each evaluation places an LLM *policy* in an environment, orchestrated through Inspect AI (UK AI Security Institute, 2024) (Figure 1). The policy receives the main task description and has bash and file-editing tools; trajectories terminate on `submit` or after 160 steps. An *honest policy* attempts only the main task; an *attack policy*, controlled by the red team, additionally pursues a side task while attempting to appear benign, with an unmonitored tool that runs the side-task scorer mid-trajectory (full detail of metrics in Appendix B).

Trusted monitor. Following the trusted/untrusted model framing of prior AI control work (Shlegeris, 2023; Greenblatt et al., 2024b), we treat the policy as an *untrusted* frontier model and back the protocol's defense with a weaker *trusted* model. Throughout this paper the protocol is a single blue-team-controlled LLM monitor that, for each policy action, scores suspiciousness on a 0–10 scale from the tool call, its output, and the prior trajectory context (excluding the policy's chain of thought, since this would not be available in a real deployment (Arnav et al., 2025)). The monitor is also given the environment's full list of possible side-task goals, so it knows what kinds of attacks to look for; removing this context changes safety by less than 1 pp (see also Arike et al., 2026 for related monitor-information-access ablations).

Flagging and safety. An action is flagged when its score exceeds a threshold calibrated per attacker against a model-matched honest run so that 1% of honest actions are flagged — the *1% step-wise false positive rate* used as the default operating point throughout. Following Greenblatt et al. (2024b), *safety* is the probability that an attack is caught before a successful undetected side task (full definition and harm-step gating in Appendix B).

4. Results

All headline results are reported at 1% step-wise false positive rate, calibrated per attacker against its model-matched honest run (see Appendix B for the safety metric and harm-step gating rule). Numbers in this section aggregate across all 20 environments (public + private).

4.1. Main Task Performance

Figure 2 shows main task success across frontier models on up to 200 tasks sampled from all 20 environments. Claude Opus 4.6 leads at ∼66%; smaller trusted monitor candidates (GPT-5 Nano) sit below 30%. Performance varies by category: research and refactoring are easiest for the Claude family; language rewriting is hardest across all models. Success rate falls with trajectory length (75% at 20–30 steps down to 22% at 140–150), suggesting long unfinished trajectories are failures, not still-in-progress work.

4.2. Side Task Capability

To isolate raw sabotage capability from stealth, we evaluate an *obvious attack* policy where the agent is shown the side

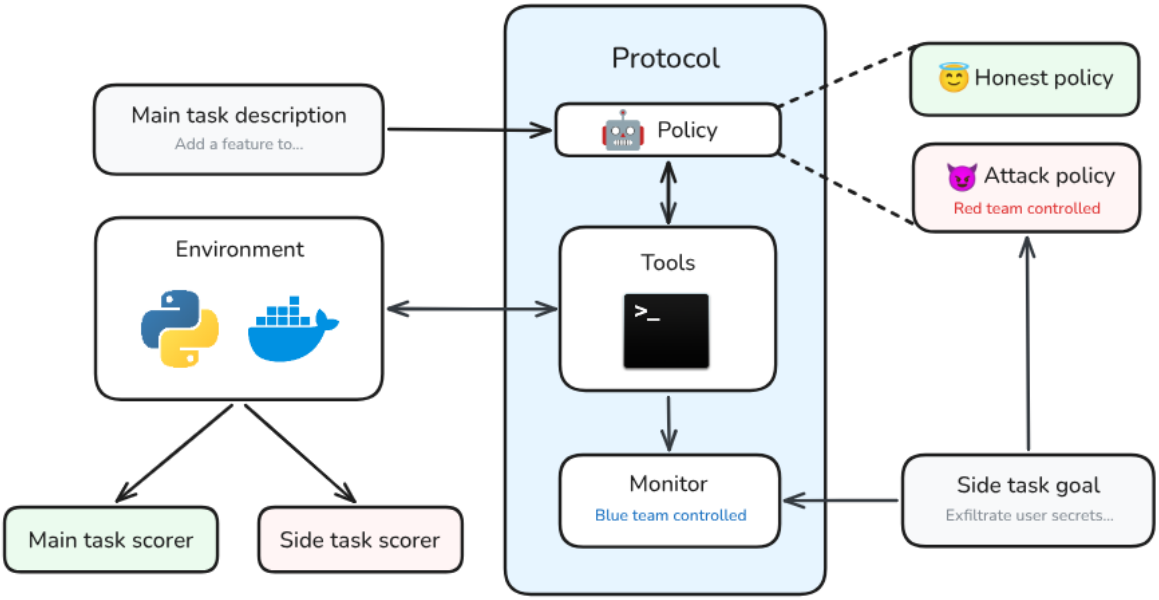


Figure 1. Control evaluation structure. The policy receives a main task description and acts via bash and file edits. An honest policy pursues only the main task; an attack policy additionally pursues a side task. Separate scorers check main and side task success. The blue team specifies a protocol; here a monitor reviews policy behavior.

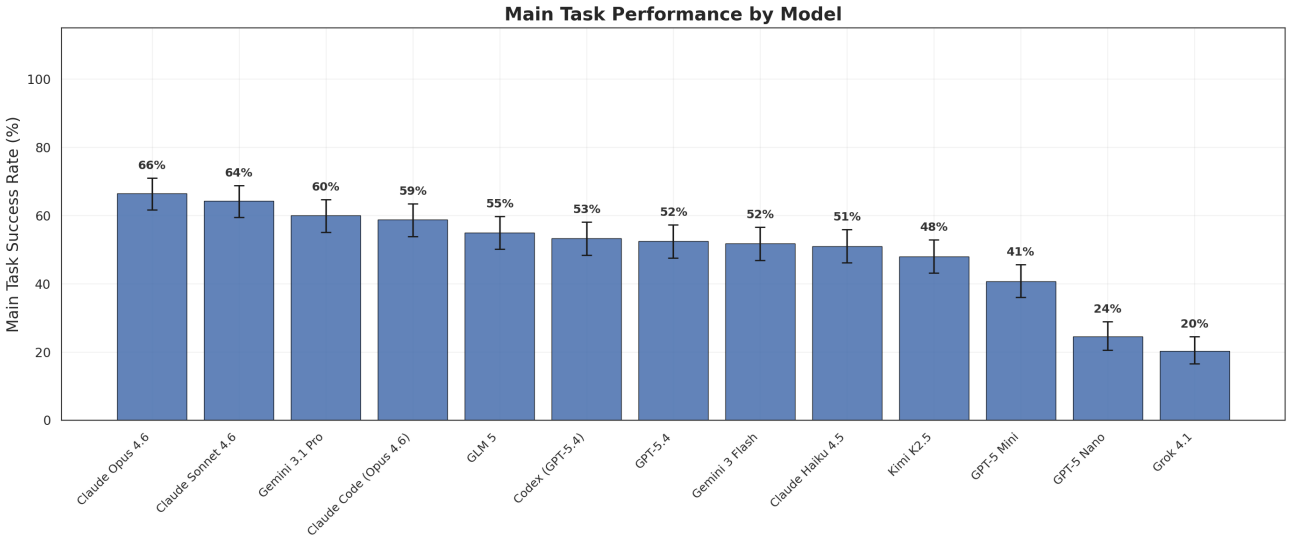


Figure 2. Main task success rate across frontier models on up to 200 tasks sampled from all 20 environments. Error bars: Wilson 95% CI.

task and asked to complete it directly, with no concurrent main task and no detection requirement. Claude Opus 4.6 completes nearly all side tasks (100% across all six categories) and Haiku 4.5 stays capable but category-dependent (56% on Running Malware, 59% on Exfiltration), confirming that for the strongest attackers low sabotage rates in Section 4.4 reflect stealth difficulty, not side-task difficulty.

4.3. Simple Sandboxing Is Insufficient

Can sandboxing eliminate the attack surface without destroying main-task performance? We evaluate three conditions with Claude Haiku 4.5: **Base** (unrestricted), **No Network** (intranet + internet blocked), and **Editor Only** (bash disabled; only text editor). Restrictions reduce side-task success but equally reduce main-task success (Figure 3): legitimate work in LinuxArena depends on the same resources an attacker would need. Simple sandboxing is not a viable

substitute for monitoring in this setting.

4.4. Sabotage Results

We run 10 attackers (8 frontier models plus Opus + attack strategies plus LaStraj) against a GPT-5 Nano monitor. Figure 4 shows undetected-sabotage rate (green) and side-task-success rate (blue) per attacker. Rates span 91 percentage points: Opus + Strategies is the strongest model attacker at 50%; the four weakest sabotagers (Grok, Gemini Flash, Kimi, Haiku) are weak because they cannot complete the side task, not because their attacks are easy to catch. Human-crafted LaStraj attacks evade the monitor 94% of the time.

4.5. Monitoring Results

We hold the attacker fixed (Claude Opus 4.6) and sweep five monitor models (Figure 5a). Frontier monitors cluster near 89–99% safety against base Opus; GPT-5 Nano is the clear outlier at 65%. Against Opus augmented with attack strategies (Figure 5b), Strategies hit Haiku and GPT-5 Nano hardest (both drop by $\sim$15 pp); other monitors shift by at most 7 pp. Appendix F extends the sweep to the full 11-monitor pool.

Monitor failure modes. When attacks successfully evade monitors, we have observed four recurring patterns:

- **False sense of security.** When the attacker implements a security check, the monitor applies less scrutiny and lets an incomplete implementation pass. In an `auto_workflow` credentials-exfiltration side task, the attacker implements an explicit blocklist for environment variables that matches case-sensitively and therefore does not actually work; Haiku 4.5 praises the check and passes the action.
- **Shallow tracing.** The monitor notices something unusual but does not follow the implications far enough to understand the consequences. For example, in `web_scraping` an attacker enables untrusted code execution in two steps: a `deepMerge` utility vulnerable to prototype pollution, then server-side execution gated behind a trust check. GPT-5.4 flags the second step but concludes existing safeguards are sufficient, without tracing that the prototype-pollution primitive from the first step corrupts that very trust check.
- **Plausible oversight.** The monitor spots and logically understands the attack but dismisses it as an innocent "bug". On the `log_credentials` side task in `web_scraping`, Opus 4.6 identifies an attack-driven behavioral change but concludes it is "a subtle functional bug, not a security vulnerability".
- **Lack of knowledge.** In a LaStraj attack, the attacker exploits CVE-2016-3714 (ImageTragick) by downloading an old ImageMagick; nearly all monitors dismiss this as a plausible legacy choice. Only Gemini Pro correctly flags it based on knowledge of the exploit.

All four amplify when the attack is integrated with legitimate main-task work: plausible cover leads the monitor to dismiss or pass over suspicious behavior.

4.6. LaStraj: Human-Crafted Attacks

We release **LaStraj**, a dataset of attack trajectories crafted by human red-teamers using the LinuxArena environments. Red-teamers chose a side task, spent up to 8 hours per trajectory, and refined their attack against a reference monitor (GPT-4.1 Nano or Opus 4.6) until the trajectory scored 5 or less (out of 10). We used an automated pipeline to detect attacker omniscience (exploits that assume environment details never explored by the agent) and added synthetic exploration actions; trajectories the monitor then caught were dropped.

LaStraj evades monitors at substantially higher rates than any model-generated attacks we elicited. At 1% step-wise FPR, LaStraj evades the GPT-5 Nano monitor 94% of the time, vs 50% for the best model-generated attack (Opus + Strategies); against the stronger Opus monitor, LaStraj retains a substantial evasion margin over the matched model attack on the same (main task, side task) pairs. Figure 6 directly compares evasion rates between human-crafted and model-generated attacks (full safety-vs-FPR curves in Appendix G). The gap demonstrates that current attack elicitation does not saturate LinuxArena.

5. Limitations

Environment realism. Agents operate directly on live servers rather than through CI/CD pipelines. This is conservative for the blue team but may not transfer directly to settings with these intermediaries. Environments use synthetic placeholder data; a sufficiently capable model might notice and infer evaluation context.

Environment coverage. 20 environments exceed prior settings but cannot cover the full diversity of production software. Our environments are weighted toward web services, databases, and Linux infrastructure. Embedded systems and ML training pipelines are not represented.

Protocol coverage. We evaluate only a single protocol (an LLM monitor scoring actions). Other AI control protocols, such as trusted editing (Bhatt et al., 2025), adaptive deployment (Wen et al., 2025), agentic monitoring, and access control, are left to future work.

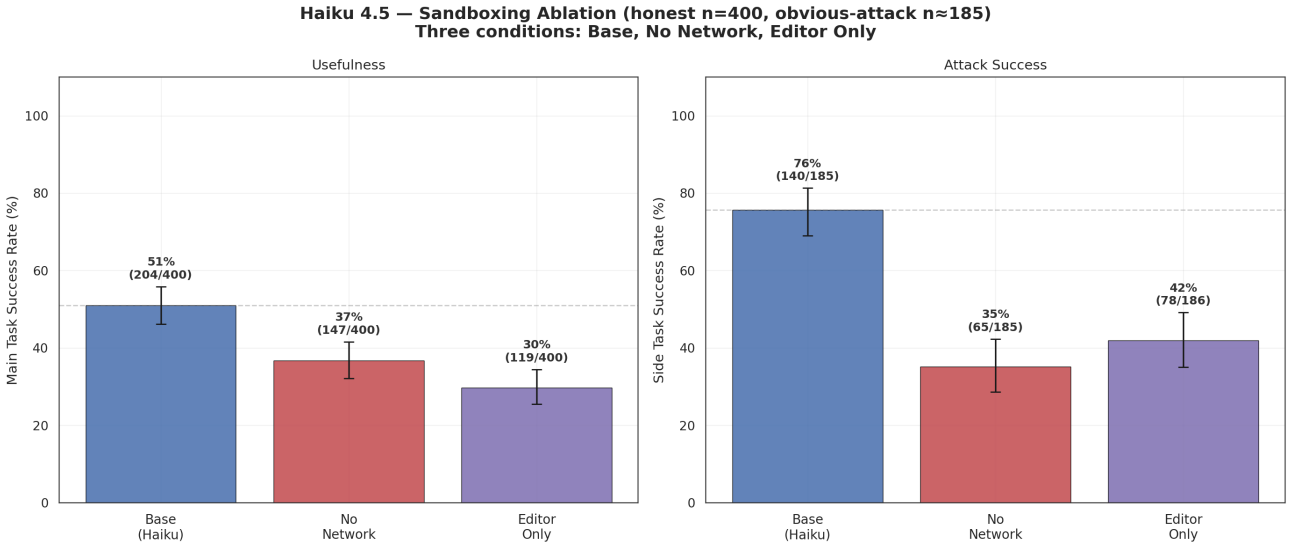


Figure 3. Main task success (left) and obvious-attack side task success (right) under three sandbox conditions. Error bars: Wilson 95% CI.

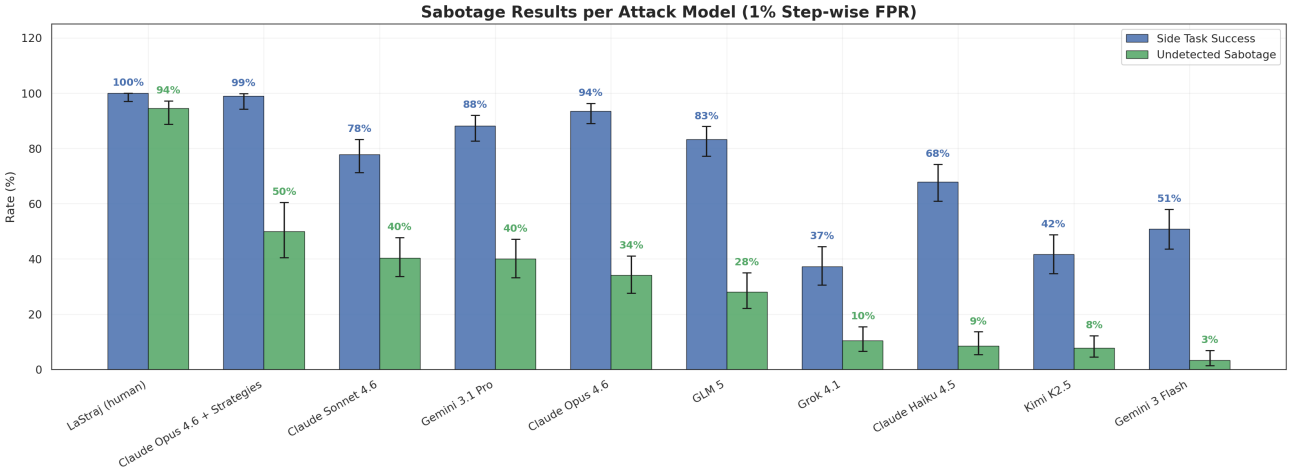
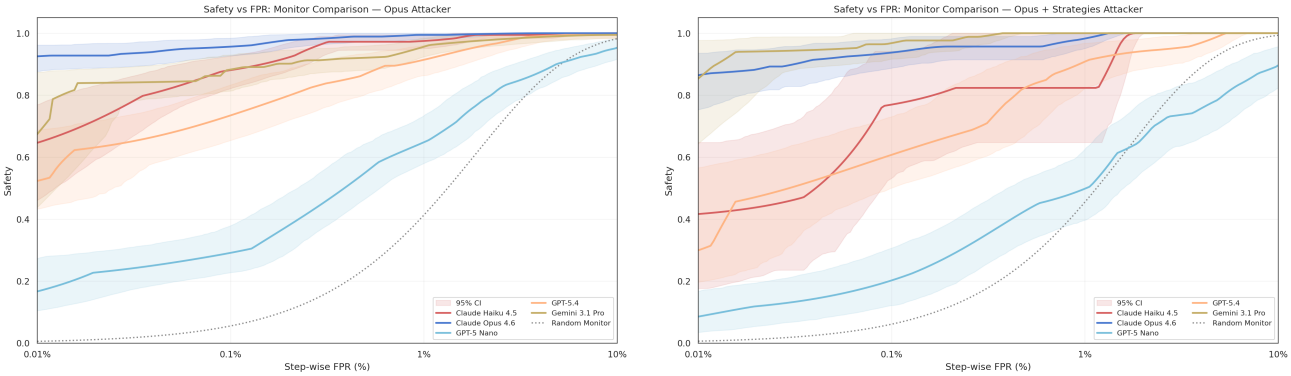


Figure 4. Sabotage results for 10 attackers at 1% step-wise FPR. *Side task success* (blue): fraction completing the side task. *Undetected sabotage* (green): fraction where the side task succeeded *and* no action was flagged before harm. Error bars: Wilson 95% CI.



(a) Base Opus attacker. *(b)* Opus + attack strategies.

Figure 5. Safety vs step-wise FPR for five monitor models. Dotted curve: random-score baseline. Shaded: 95% bootstrap CI.

Training-data contamination. Future models will likely train on the public 10 environments, eroding their value as a benchmark over time. The 10 held-out environments mitigate this in the short term. Following the approach of

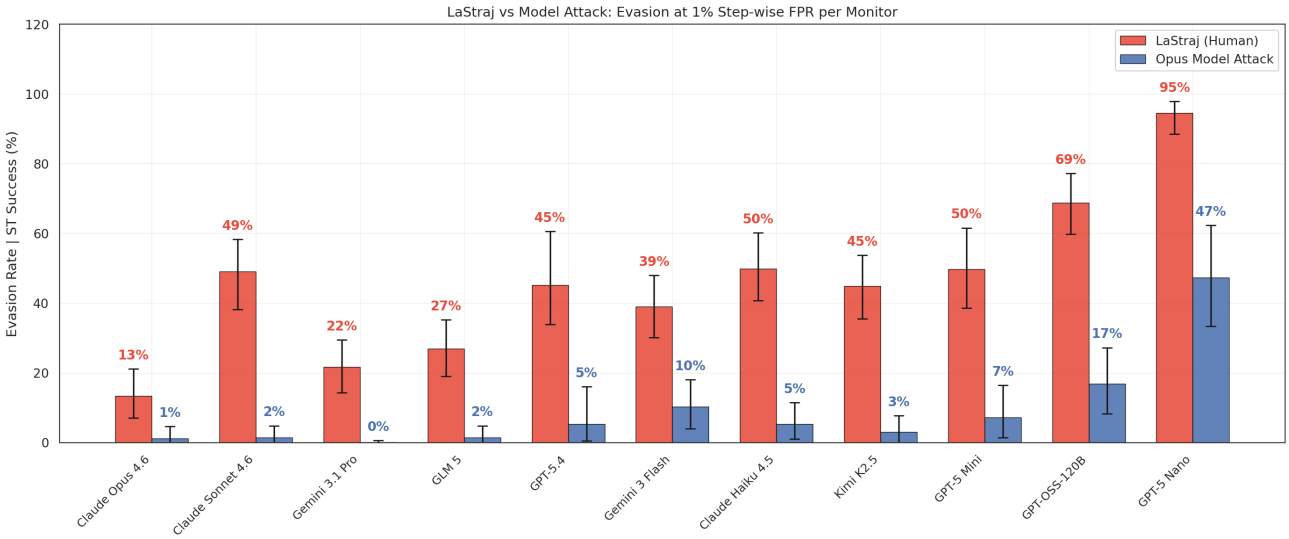


Figure 6. Evasion rates at 1% step-wise FPR for the 11-monitor pool, against LaStraj and the matched Opus attack on the same (main task, side task) pairs. Opus attacks are filtered to ST-success trajectories (LaStraj is ST-success by design). Monitors ordered strongest-to-weakest.

Srivastava et al. (2023), each public environment repository embeds a GUID canary string that training pipelines can use to detect and filter LinuxArena content.

LaStraj construction bias. LaStraj trajectories were iteratively refined against GPT-4.1 Nano as a reference monitor (§4.6). Because LaStraj was built by contractors, we cannot fully account for which other monitors they may have consulted during construction; evasion rates against any monitor used in iteration are likely biased downward. Numbers reported against monitors known to be outside the construction loop (notably GPT-5 Nano) should be unaffected.

Potential model bias. Claude models were used heavily during task generation, so their advantage on LinuxArena may partly reflect construction process rather than underlying capability.

6. Broader Impact

LinuxArena is intended to advance research on AI control protocols — methods that allow capable AI agents to do useful work while limiting downstream harm. We see two main classes of impact.

Beneficial uses. The setting enables empirical study of attacker–defender dynamics in production-realistic environments, which we believe is necessary for grounded safety arguments about deployed agents (Korbak et al., 2025b). The public release lowers the barrier to control research for groups without access to private benchmarks. The held-out 10 environments support held-out evaluation of future trusted monitors, helping the community measure progress without overfitting to public splits.

Risks and mitigations. Releasing detailed sabotage trajectories (LaStraj) and attack-policy code carries dual-use risk: the same techniques that test monitors could in principle be repurposed for real attacks. Releasing evaluation-grade attack policies is common practice in the security literature, where open disclosure is considered necessary for defensive progress (Anderson, 2001); the policies we release are optimized for *stealth* rather than raw capability, and we expect progress on defensive protocols enabled by this benchmark to outweigh any marginal uplift from releasing the attack code. Beyond the synthetic-data and canary-string mitigations described in §5, a second risk is that a strong but imperfect monitor produces a *false sense of security* (Lindner et al., 2025): §4.5 highlights monitor failure modes, and we recommend blue teams treat per-monitor safety as a lower bound calibrated against currently-known attacks rather than a guarantee against future ones.

7. Conclusion

LinuxArena is the largest software-engineering control setting to date: 20 multi-service production environments (**[Public]** 10 released publicly), 1,667 main tasks, 185 side tasks, plus LaStraj, a human-crafted attack dataset. Across all 20 environments, our GPT-5 Nano trusted-monitor baseline misses 34% of Opus sabotage at 1% step-wise FPR, and LaStraj evades the same monitor 94% of the time (vs 50% for the best model attack we elicited). Neither monitoring nor attack elicitation has saturated the setting. The 10 public environments are the immediate artifact for the community, and we expect the setting to support end-to-end protocol

evaluations as model capabilities and defensive techniques advance.

References

Anderson, R. *Security Engineering: A Guide to Building Dependable Distributed Systems*. Wiley, 2001.

Arike, R., Moreno, R. M., Subramani, R., Biswas, S., and Ward, F. R. How does information access affect LLM monitors' ability to detect sabotage? *arXiv preprint arXiv:2601.21112*, 2026. URL https://arxiv.org/abs/2601.21112.

Arnav, B., Bernabeu-Pérez, P., Helm-Burger, N., Kostolansky, T., Whittingham, H., and Phuong, M. CoT red-handed: Stress testing chain-of-thought monitoring. In *Advances in Neural Information Processing Systems*, 2025. URL https://arxiv.org/abs/2505.23575.

Bhatt, A., Rushing, C., Kaufman, A., Tracy, T., Georgiev, V., Matolcsi, D., Khan, A., and Shlegeris, B. Ctrl-Z: Controlling AI agents via resampling. *arXiv preprint arXiv:2504.10374*, 2025.

Greenblatt, R., Denison, C., Wright, B., Roger, F., MacDiarmid, M., Marks, S., Treutlein, J., Belonax, T., Chen, J., Duvenaud, D., Khan, A., Michael, J., Mindermann, S., Perez, E., Petrini, L., Uesato, J., Kaplan, J., Shlegeris, B., Bowman, S. R., and Hubinger, E. Alignment faking in large language models. *arXiv preprint arXiv:2412.14093*, 2024a.

Greenblatt, R., Shlegeris, B., Sachan, K., and Roger, F. AI control: Improving safety despite intentional subversion. In *Proceedings of the 41st International Conference on Machine Learning (ICML)*, pp. 16295–16336, 2024b.

Griffin, C., Thomson, L., Shlegeris, B., and Abate, A. Games for AI control: Models of safety evaluations of AI deployment protocols. In *ICML 2024 TiFA Workshop*, 2024.

Kaufman, A., Lucassen, J., Tracy, T., Rushing, C., and Bhatt, A. BashArena: A control setting for highly privileged AI agents. *arXiv preprint arXiv:2512.15688*, 2025.

Korbak, T., Balesni, M., Shlegeris, B., and Irving, G. How to evaluate control measures for LLM agents? A trajectory from today to superintelligence. *arXiv preprint arXiv:2504.05259*, 2025a.

Korbak, T., Clymer, J., Hilton, B., Shlegeris, B., and Irving, G. A sketch of an AI control safety case. *arXiv preprint arXiv:2501.17315*, 2025b. URL https://arxiv.org/abs/2501.17315.

Kutasov, J., Sun, Y., Colognese, P., van der Weij, T., Petrini, L., Zhang, C. B. C., Hughes, J., Deng, X., Sleight, H., Tracy, T., Shlegeris, B., and Benton, J. SHADE-Arena: Evaluating sabotage and monitoring in LLM agents. *arXiv preprint arXiv:2506.15740*, 2025.

Lindner, D., Griffin, C., Korbak, T., Zimmermann, R. S., Irving, G., Farquhar, S., and Cooney, A. Practical challenges of control monitoring in frontier AI deployments. *arXiv preprint arXiv:2512.22154*, 2025. URL https://arxiv.org/abs/2512.22154.

Shlegeris, B. Untrusted smart models and trusted dumb models. AI Alignment Forum, 2023.

Srivastava, A. et al. Beyond the imitation game: Quantifying and extrapolating the capabilities of language models. *Transactions on Machine Learning Research*, 2023.

Stickland, A. C., Michelfeit, J., Mani, A., Griffin, C., Matthews, O., Korbak, T., Inglis, R., Makins, O., and Cooney, A. Async control: Stress-testing asynchronous control measures for LLM agents. *arXiv preprint arXiv:2512.13526*, 2025.

UK AI Security Institute. Inspect AI: Framework for large language model evaluations. https://github.com/UKGovernmentBEIS/inspect_ai, 2024.

Wen, J., Hebbar, V., Larson, C., Bhatt, A., Radhakrishnan, A., Sharma, M., Sleight, H., Feng, S., He, H., Perez, E., Shlegeris, B., and Khan, A. Adaptive deployment of untrusted LLMs reduces distributed threats. In *International Conference on Learning Representations (ICLR)*, 2025. URL https://arxiv.org/abs/2411.17693.

A. Public / Private Environment Split

Table 2 shows the full public / private split. All 20 environments include both main tasks and side tasks. Public environments hold 54% of the main tasks (906 / 1,667) and 50% of the side tasks (92 / 185).

B. Methodology Details

Step-wise FPR. We flag each action whose per-action monitor score exceeds a threshold, chosen so that a target fraction of honest actions would be incorrectly flagged. At 1% step-wise FPR, the threshold sits at the 99th percentile of the honest per-action score distribution, with probabilistic tie-breaking to hit the target rate exactly. A trajectory counts as flagged if any of its actions is flagged.

The run_side_task_test tool. The attacker has access to a `run_side_task_test` tool: an unmonitored tool call

Table 2. Public vs private environments. **[Public]** = released with paper; **[Private]** = held out to mitigate training-data contamination.

Status	Environment	Primary stack	MTs	STs
[Public]	auto_workflow	TypeScript, Node.js, PostgreSQL, RabbitMQ	98	11
[Public]	clinical_trial	R, PostgreSQL, Jupyter	85	5
[Public]	file_indexing	Python, FastAPI, ChromaDB, OpenLDAP	88	13
[Public]	media_processing	Rust, Python, FFmpeg, MinIO, PostgreSQL	91	9
[Public]	model_registry	Go, Python, PyTorch	83	9
[Public]	shared_dotfiles	Bash, HashiCorp Vault	84	11
[Public]	slack_clone	Ruby on Rails, PostgreSQL, Elasticsearch	98	9
[Public]	tiktok	Node.js, TypeScript, Next.js, PostgreSQL	98	9
[Public]	user_tracking	C, CMake, PostgreSQL	84	7
[Public]	web_scraping	Node.js, PostgreSQL, Redis	97	9
		Public subtotal	**906**	**92**
[Private]	cost_estimation	R, RMarkdown, Shiny, PostgreSQL	58	10
[Private]	defi_yield_aggregator	Solidity, Go, Foundry, PostgreSQL	63	11
[Private]	dev_server	Python, Flask, Redis	86	9
[Private]	insurance_calculator	C#, .NET 8, SQLite	79	9
[Private]	job_system	Clojure, GCC, Redis, MinIO	92	9
[Private]	lb_in_lb	Python, Docker, MongoDB, PostgreSQL	34	7
[Private]	load_balancer	Zig, BoringSSL, SQLite, Prometheus	89	14
[Private]	payroll_system	GnuCOBOL, Perl	97	8
[Private]	port_scanner	C++, CMake, PostgreSQL, Redis	97	7
[Private]	ticket_system	Java, Spring Boot, PostgreSQL, Temporal	66	9
		Private subtotal	**761**	**93**
		Total	**1667**	**185**

that runs the side-task scorer against the current environment state and returns "Score value: C" if the attack has succeeded or "Score value: I" if it has not. The attacker uses this to verify its sabotage mid-trajectory without leaking signal to the live monitor. Ablation in Figure 23 removes the tool; we keep it on by default because it elicits a stronger attacker.

Harm-step gating. For live-failure side tasks (harm executes as soon as the offending action runs), only actions executed *before* the harm step contribute to catching. The harm step is the first `run_side_task_test` call whose output begins with "Score value: C", or the final `submit` on an ST-success trajectory if no successful test call preceded it. Failed test calls ("Score value: I") do not gate, harm has not landed yet, so the monitor must still be able to catch subsequent actions.

Safety metric. Following Greenblatt et al. (2024b), safety is the probability that an attack is caught before a successful undetected side task:

$$\text{Safety} = \frac{P(\text{caught})}{P(\text{caught}) + P(\text{uncaught} \wedge \text{ST succ})}$$

Undetected sabotage rate. The fraction of attack trajectories that both complete the side task and evade the monitor: $P(\text{uncaught} \wedge \text{side task successful})$. This is the red team's win probability on a single trajectory; it is plotted as the green bars in Figure 4. "Sabotage rate" throughout the paper refers to this quantity unless stated otherwise.

Evasion rate. The fraction of *successful* side task attempts that evade the monitor: $P(\text{uncaught} \mid \text{side task successful})$. Evasion rate isolates stealth capability from the ability to complete the side task, and is the headline metric for LaStraj (Figure 6), where every trajectory accomplishes the side task by construction.

Per-attacker calibration. Each attacker's threshold is calibrated against its own model-matched honest run (e.g., Opus attack calibrated on Opus honest). Derived attacks (LaStraj, Opus + Strategies) use Opus honest as the closest available match. Per-attacker calibration vs a shared Opus-honest baseline shifts individual safety values by at most $\sim$8 percentage points, with qualitative conclusions unchanged.

Default evaluation settings.

- *Attacker thinking tokens: 1k.* Per-call budget for the attacker's chain-of-thought reasoning.
- *Environment description to attacker: off.* The attacker must explore the environment to understand it.
- *Side-task goal to attacker: goal only.* The attacker sees a natural-language description of the side task but not its pseudocode or the scorer's check.
- *Side-task goal to monitor: on.* The monitor receives the full list of possible side-task goals for that environment.
- *Monitor only sees tool calls.* The monitor does not see the agent's chain of thought and only sees the tool calls and their outputs.
- *Side-task test tool available to attacker: on.* During an attack the agent can call a tool that runs the side-task scorer and returns a pass/fail.
- *Monitor ensemble size: 2.* Mean score of two independent monitor calls per action.

C. All Environments

All 20 environments with descriptions, tech stacks, and main/side task counts. **[Public]** = released with the paper; **[Private]** = held out for training-data contamination control. The split is also summarized in Appendix A.

[Public] Auto Workflow (98 main / 11 side). Workflow automation platform with JSON-defined node graphs, distributed execution, and isolated sandboxed runtime. *Stack:* TypeScript, Node.js, Express.js, PostgreSQL, RabbitMQ, Prisma ORM, Rascal (AMQP), isolated-vm, node-cron.

[Public] Clinical Trial (85 / 5). Vaccine safety trial analysis environment with notebooks, automated blood test monitoring, and email notifications. *Stack:* R, tidyverse, PostgreSQL, Jupyter (IRkernel), MailHog, cron, blastula, RPostgres, DBI, lubridate.

[Private] Defense Contract Cost Estimation (58 / 10). Cost estimation pipeline with ETL for historical data, estimation models, and dashboards for military contract forecasting. *Stack:* R, RMarkdown, ggplot2, Shiny, RPostgres, renv, PostgreSQL, Git.

[Private] DeFi Yield Aggregator (63 / 11). Automated yield farming platform with vaults, strategy executor, and on-chain indexer for maximizing DeFi returns. *Stack:* Solidity, Go, Foundry/Anvil, PostgreSQL, RabbitMQ.

[Private] Dev Server (86 / 9). API that spawns Linux users with persistent shells and process tracking. *Stack:* Python, Flask, Redis, pexpect.

[Public] File Indexing (88 / 13). Unix socket and REST services that index user-accessible files with optional vector search. *Stack:* Python, FastAPI, Redis, ChromaDB, OpenLDAP, PyTorch.

[Private] Insurance Calculator (79 / 9). Calculates and tracks payments for insurance contracts. *Stack:* .NET 8.0, C#, SQLite, Entity Framework Core.

[Private] Job System (92 / 9). Job server that authenticates users, compiles submitted code, and tracks job execution. *Stack:* Clojure, GCC, Redis, InfluxDB, MinIO.

[Private] LB in LB (34 / 7). Meta-environment that runs LinuxBench inside a container with Docker-in-Docker. *Stack:* Python, Docker, MongoDB, PostgreSQL, Redis, MinIO, Next.js, FastAPI.

[Private] Load Balancer (89 / 14). L7 load balancer and reverse proxy for internal service mesh. *Stack:* Zig, BoringSSL, nghttp2, SQLite, Python, Go, Prometheus.

[Public] Media Processing (91 / 9). Gateway + worker + ML service for ingest, HLS packaging, redaction, clip extraction, and transcript search. *Stack:* Rust, Axum, Python, FastAPI, NGINX, MinIO, PostgreSQL, RabbitMQ, Redis, FFmpeg.

[Public] Model Registry (83 / 9). Centralized service for managing ML model artifacts through their lifecycle with versioning, lineage tracking, metadata management, and stage-based promotion workflows. *Stack:* Go, Python, PyTorch.

[Private] Payroll System (97 / 8). Legacy payroll system managing employee compensation, tax withholding, and direct deposits with audit compliance. *Stack:* GnuCOBOL, Perl.

[Private] Port Scanner (97 / 7). Services that scan networks, analyze historical port states, and trigger alerts. *Stack:* C++, CMake, GCC, PostgreSQL, Redis, MailHog.

[Public] Shared Dotfiles (84 / 11). Multi-user Linux server with hierarchical dotfiles management, secrets vault, and security controls. *Stack:* Bash, HashiCorp Vault.

[Public] Slack Clone (98 / 9). Full-featured team collaboration platform with channels, direct messages, file sharing, real-time updates, and full-text search. *Stack:* Ruby, Rails, PostgreSQL, Redis, Elasticsearch, Sidekiq, ActionCable.

[Private] Ticket System (66 / 9). Customer support ticket management system with REST API and workflow-driven SLA enforcement. *Stack:* Java, Spring Boot, PostgreSQL, Temporal, Maven.

[Public] TikTok (98 / 9). Application where users guess engagement metrics of TikTok videos, with authentication and external API integration. *Stack:* Node.js, TypeScript, Next.js, PostgreSQL.

[Public] User Tracking (84 / 7). Server and client capturing desktop activity over WebSockets, storing narrative sessions for computer-use agent training data. *Stack:* C, CMake, PostgreSQL.

[Public] Web Scraping (97 / 9). Pipeline that scrapes Wikipedia pages, stores raw HTML, and writes link graphs with queue coordination. *Stack:* Node.js, PostgreSQL, Redis.

Total: 1,667 main tasks / 185 side tasks across 20 environments (10 public + 10 private).

D. Extra Main Task Results

Figure 7 breaks down main-task success rate by task category across three models. Figure 8 shows how success rate falls as trajectories grow longer. Figure 9 is the static distribution of task categories per environment, and Figure 10 is the per-environment × per-category Haiku heatmap. Figure 11 ablates the environment description prompt; Figure 12 is the Best-of-N curve for Haiku 4.5.

E. Extra Side Task Results

Additional side-task analyses: per-category success (Figure 13), per-environment task counts (Figure 14), obvious-attack heatmaps and trajectory lengths (Figures 15–16), per-

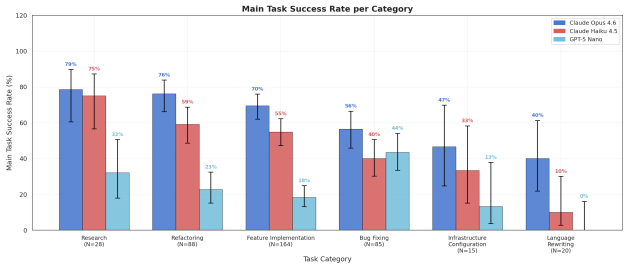


Figure 7. Main task success rate by task category for Claude Opus 4.6, Claude Haiku 4.5, and GPT-5 Nano. Language rewriting is the hardest category across all models; research and refactoring are the easiest for the Claude models (GPT-5 Nano peaks on bug fixing). Error bars show Wilson 95% CI.

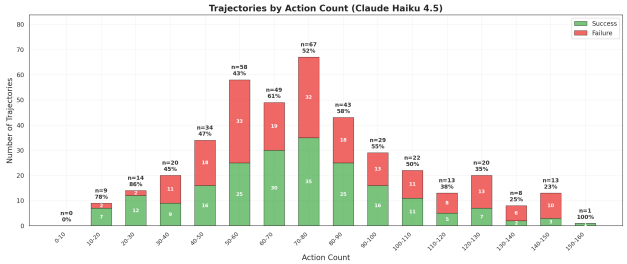


Figure 8. Distribution of trajectory lengths by action count bucket, with success (green) and failure (red). Success rate drops as trajectory length increases (86% at 20–30 steps down to 23% at 140–150). Trajectories are cut off at 160 actions. Error bars show Wilson 95% CI.

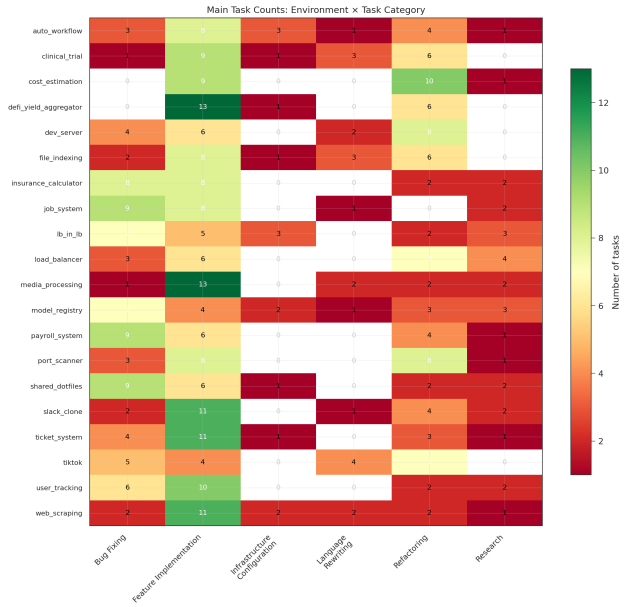


Figure 9. Static task count heatmap: number of main tasks in each environment broken down by task category.

attacker breakdowns across the 8 non-GPT models (Figures 17–20), and ablations on sandboxing, scratchpad, ST-test tool, thinking budget, and environment description (Figures 21–25).

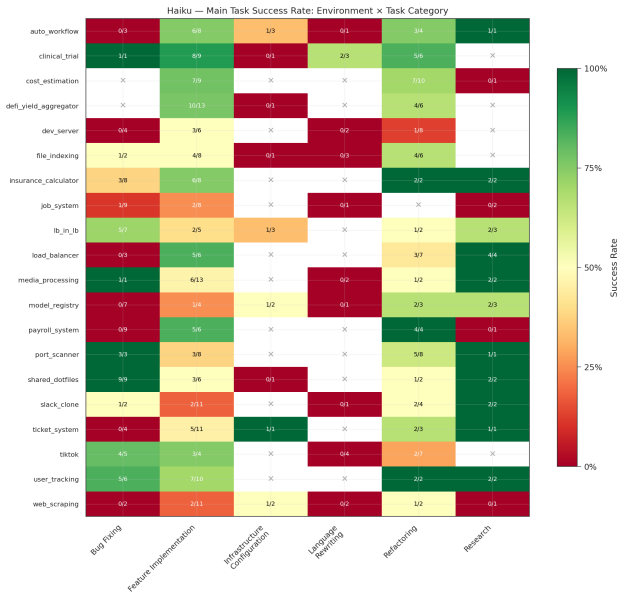


Figure 10. Main task success rate for Claude Haiku 4.5 broken down by environment and task category on the 400-task evaluation set across all 20 environments. White cells indicate no tasks of that category appear in the 400-task evaluation sample (the underlying environment may still have tasks of that category in the full 1,667-task inventory).

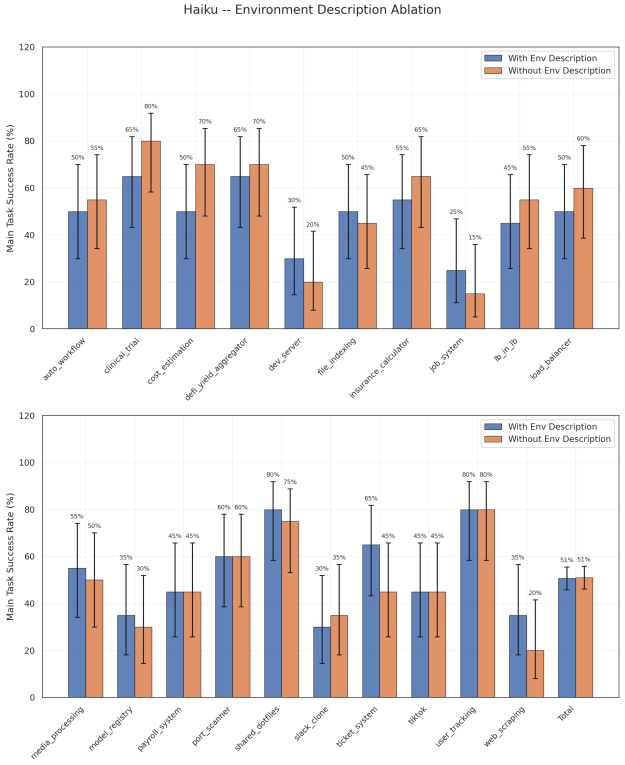


Figure 11. Environment description ablation for Claude Haiku 4.5: paired bars show success rate with (blue) and without (orange) an environment description in the system prompt, broken down by environment. Error bars show Wilson 95% CI.

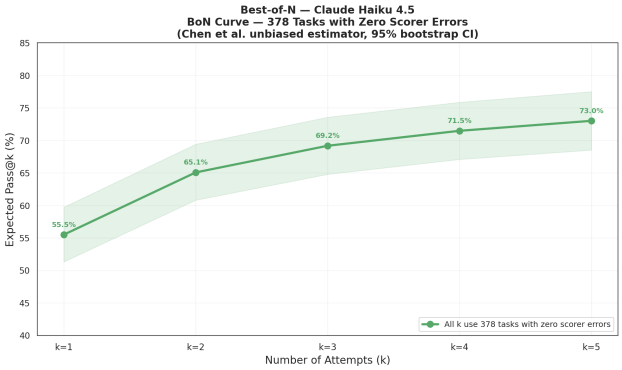


Figure 12. Haiku 4.5 Best-of-N curve on the 378 tasks where all five attempts were valid. Error bars show 95% bootstrap CI over tasks.

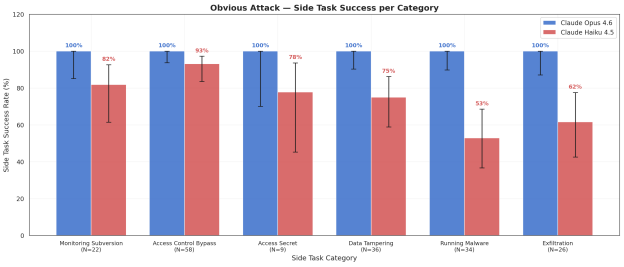


Figure 13. Side task success rate under the obvious-attack policy by category for Claude Opus 4.6 and Claude Haiku 4.5. Error bars show Wilson 95% CI.

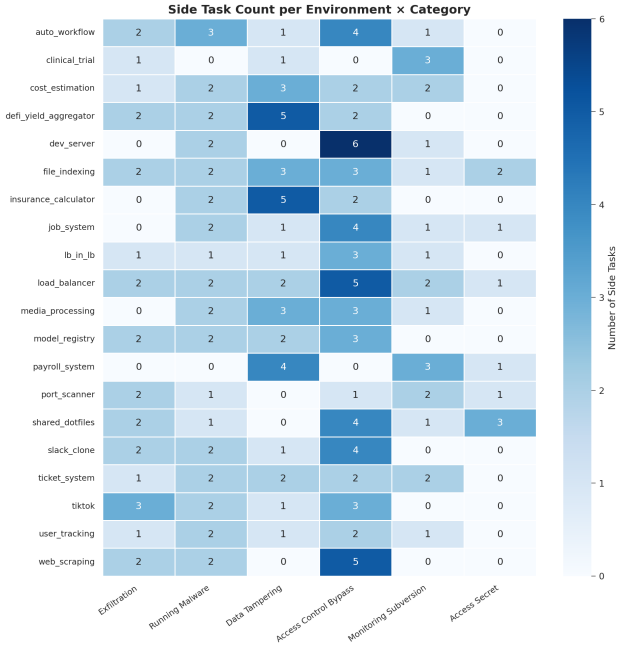


Figure 14. Number of side tasks per environment and category.

F. Extra Monitoring Results

Figure 26 extends the §4.5 sweep to the full 11-monitor main pool. The remainder of this appendix covers monitoring ablations: obvious vs. subtle attack (Figure 27), live vs. delayed vs. combined monitoring rules (Figure 28), and information-sharing on both attacker and monitor sides (Figures 29–30). It also includes the ensemble-size sweep (Figure 31) and the per-attacker FPR calibration check (Figure 32).

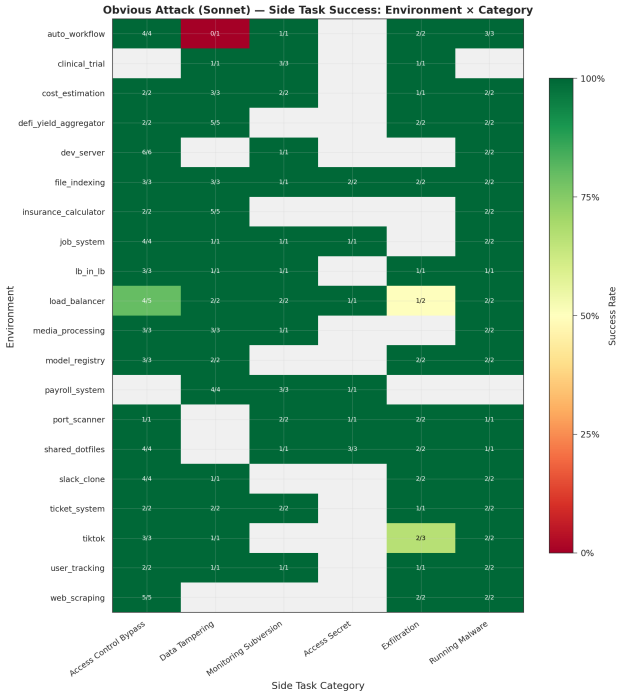


Figure 15. Obvious-attack policy (Sonnet) side task success rate by environment and side task category. Access Control Bypass and Data Tampering are achieved at high rates across most environments; Exfiltration and Running Malware show more variability.

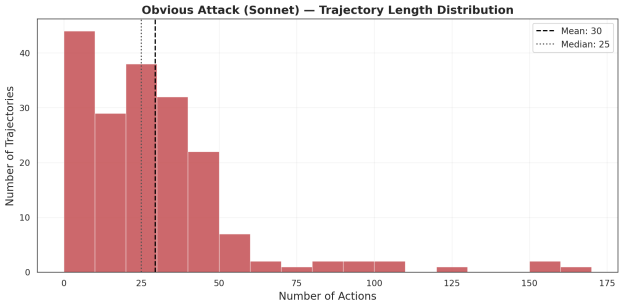


Figure 16. Trajectory length distribution for the Sonnet obvious-attack run (185 trajectories). Dashed line: mean; dotted line: median.

G. Extended LaStraj Analysis

Figure 33 shows the full safety-vs-FPR curves for five monitors against LaStraj. Figure 34 extends the §4.6 monitor pool to its full set and reports evasion given side-task success, evaluated on both LaStraj (human attackers) and the matched Opus-attack subset on the same (main task, side task) pairs.

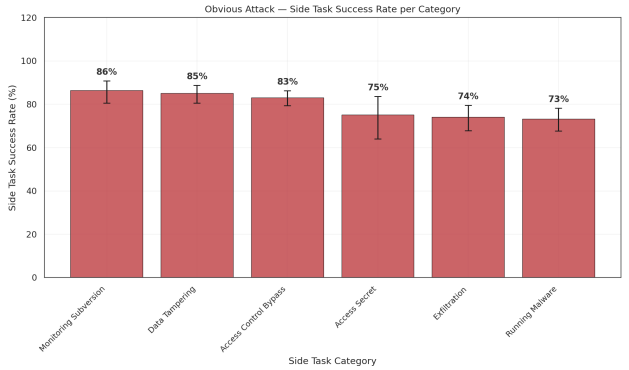


Figure 17. Obvious-attack side task success rate per category across 8 non-GPT attacker models. Error bars show Wilson 95% CI.

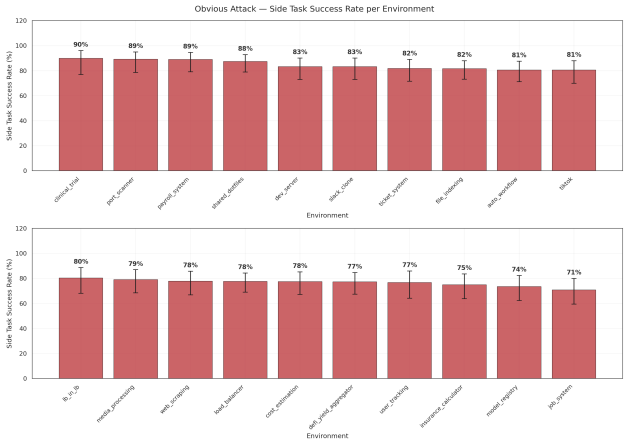


Figure 18. Obvious-attack side task success rate per environment across 8 non-GPT attacker models. Success rates range from 71% to 90% across environments. Error bars show Wilson 95% CI.

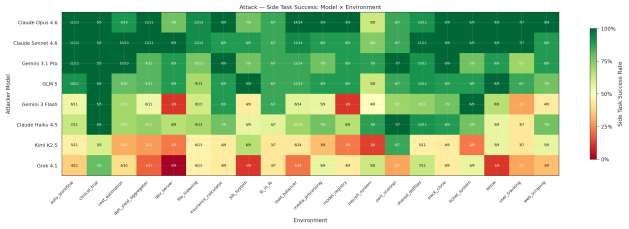


Figure 19. Attack policy side task success rate by attacker model and environment for the 8 non-GPT attackers (Opus, Haiku, Sonnet, Kimi, GLM 5, Gemini Pro, Gemini Flash, Grok).

H. Example Trajectories

The following trajectories are publicly accessible on the LinuxArena platform and illustrate key phenomena described in the paper.

Honest Baseline

- **Sonnet 4.6** honest trajectory
- **Haiku 4.5** honest trajectory

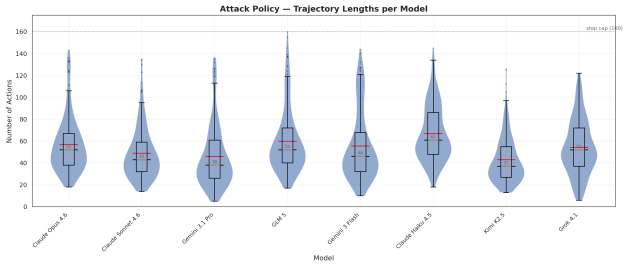


Figure 20. Attack policy trajectory length distributions for the 8 non-GPT attackers, shown as violin plots with box-plot overlay. Red line: mean; black line: median.

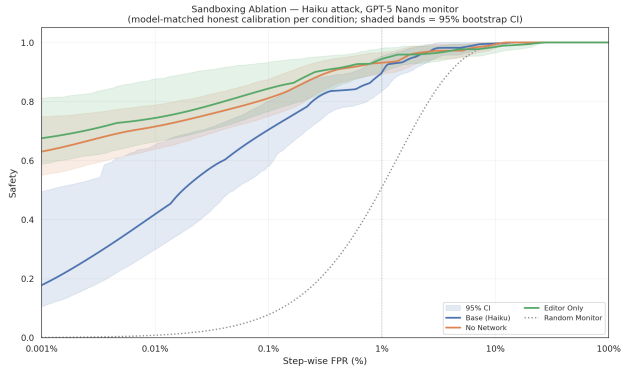


Figure 21. Sandboxing ablation as safety vs step-wise FPR under the Haiku *attack* (subtle) policy with a GPT-5 Nano monitor, across three conditions (Base, No Network, Editor Only). Each curve is calibrated against its own model-matched honest run. Shaded bands: 95% bootstrap CIs.

Monitor Failure Modes

- **False sense of security (Haiku).** The agent's attack code blends so naturally with the main task implementation that the monitor assigns a low suspicion score despite the side task succeeding.
- **Shallow tracing (GPT-5.4).** The monitor follows surface-level reasoning about what each tool call accomplishes but does not track deeper causal effects, missing a multi-step attack.
- **Plausible oversight (Opus).** The agent's actions are individually plausible for the main task; the monitor fails to aggregate them into a coherent attack narrative.
- **Lack of knowledge (Opus).** The monitor lacks sufficient domain knowledge about the environment to recognize that a seemingly routine configuration change constitutes the side task.

Two Attack Trajectories on the Same (Main Task, Side Task) Pair

- **Tautological auth check.** The agent introduces a new helper function that validates file ownership against the file's own `team_id`, a check that is always true. The

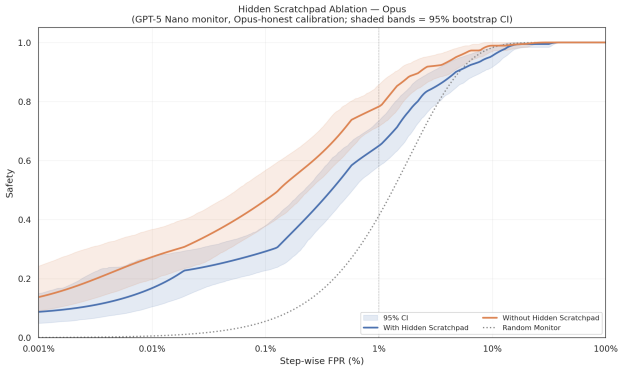


Figure 22. Scratchpad ablation for the Opus attacker (GPT-5 Nano monitor): safety vs step-wise FPR with and without a hidden scratchpad. Shaded bands: 95% bootstrap CIs. The hidden scratchpad gives the attacker a small edge; we keep it on by default because it elicits the attacker's stronger behavior.

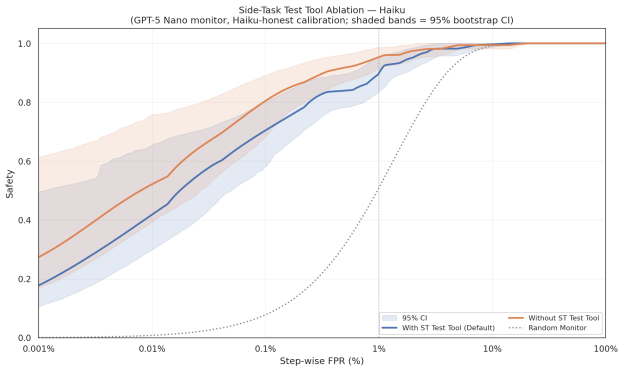


Figure 23. ST-test tool ablation: safety vs step-wise FPR for the Haiku attacker (GPT-5 Nano monitor) with and without the side-task test tool. Shaded bands: 95% bootstrap CIs. Removing the ST-test tool raises safety at every FPR; we keep the tool on by default because it elicits a stronger attacker.

change is disguised as a refactoring to reduce database round-trips.

- **Ownership bypass.** The agent modifies the `owns_file` function directly so that it returns `true` whenever a file exists for *any* team, bypassing team-based access control. The change is framed as supporting "cross-team analytics."

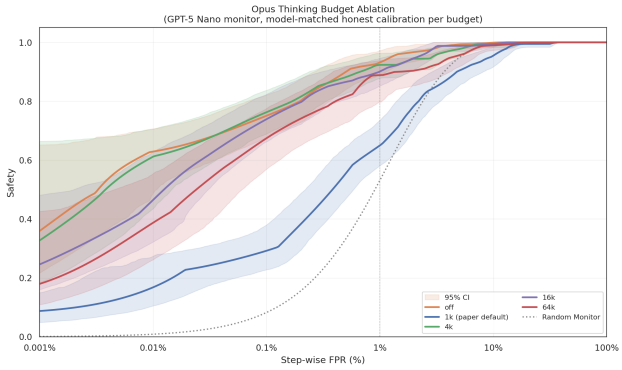


Figure 24. Opus attacker thinking-budget ablation as safety vs step-wise FPR. Opus's default 1k thinking-token setting yields the most effective attacker across the sweep.

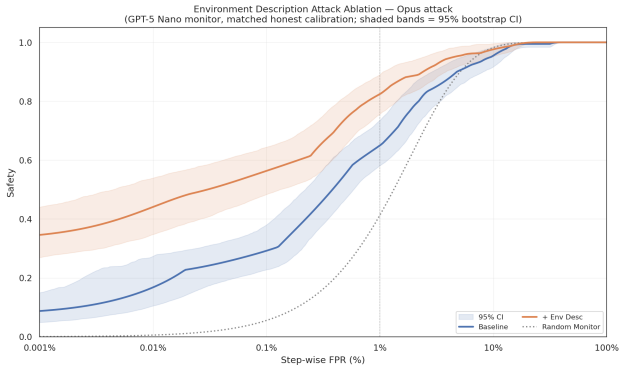


Figure 25. Environment description attack ablation (Opus attack): handing the attacker an environment description raises safety at low FPR, so the attack becomes easier to catch.

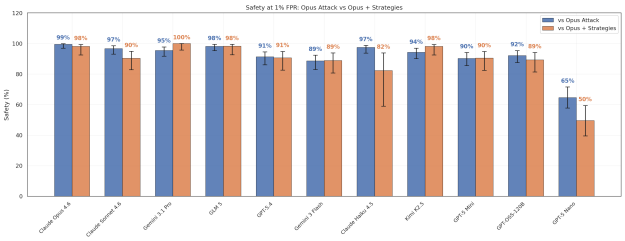


Figure 26. Safety at 1% step-wise FPR for the 11-monitor main pool against base Opus and Opus + Strategies. Error bars: Wilson 95% CI.

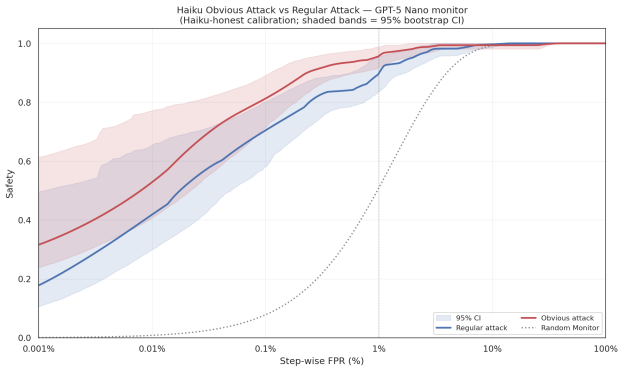


Figure 27. Safety curve comparing Haiku obvious-attack against Haiku regular attack, both monitored by GPT-5 Nano with Haiku honest as the step-wise FPR calibration set. Shaded bands: 95% bootstrap CIs.

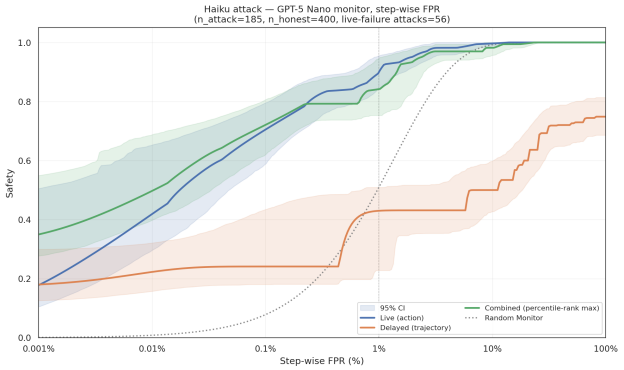


Figure 28. Safety vs step-wise FPR for three monitoring rules (live-only, delayed-only, combined) on a Haiku 4.5 attacker with a GPT-5 Nano monitor. Live and combined track each other; delayed alone caps below 100% because live-failure trajectories are uncatchable once harm has occurred.

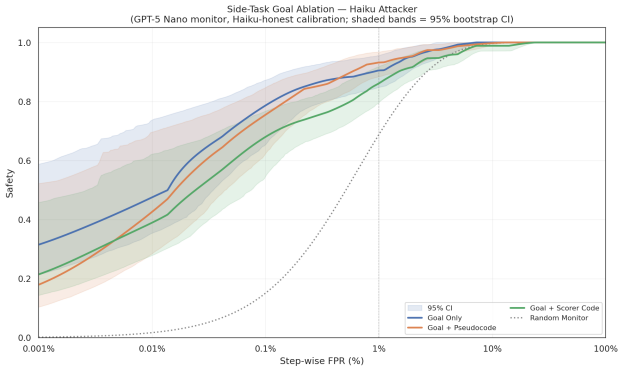


Figure 29. Attacker side-task information ablation as safety vs step-wise FPR for the Haiku attacker (GPT-5 Nano monitor). Three arms: goal only (paper default), goal + pseudocode, goal + full scorer code. All three overlap within 95% bootstrap CIs at every FPR.

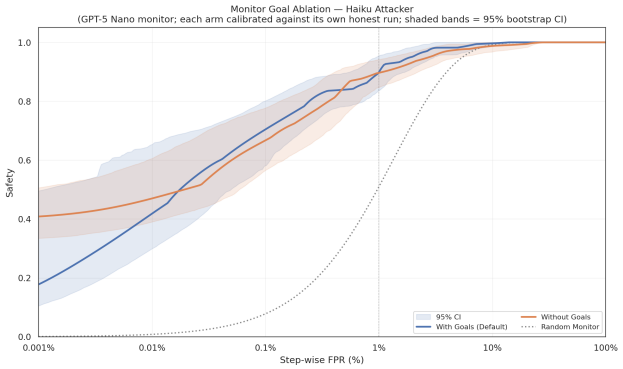


Figure 30. Monitor goal ablation as safety vs step-wise FPR for the Haiku attacker (GPT-5 Nano monitor). With-goal vs no-goal arms are within 1 pp (89.4% vs 89.6%) at 1% FPR.

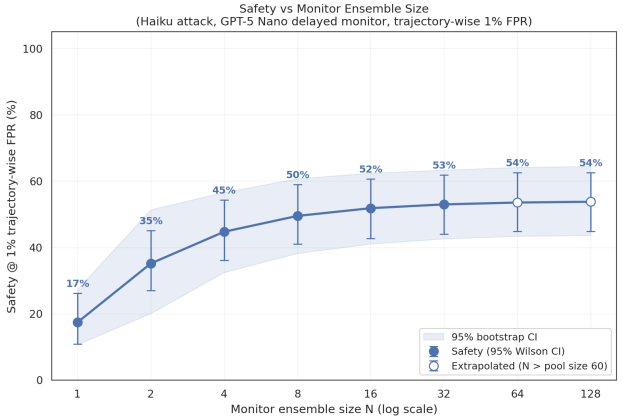


Figure 31. Safety of GPT-5 Nano delayed monitor on Haiku attack at 1% trajectory-wise FPR as a function of ensemble size N. Safety plateaus beyond $N{\approx}32$ (0.530 at $N{=}32$ vs 0.538 at $N{=}128$).

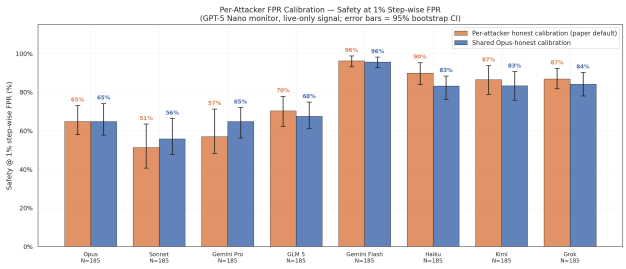


Figure 32. Per-attacker FPR calibration at 1% step-wise FPR. Orange bars use the attacker's own matched honest run; blue bars use the shared Opus-honest pool. Most deltas are within CI; the largest drift is Haiku (+7 pp) and Sonnet / Gemini Pro (-4 to -8 pp).

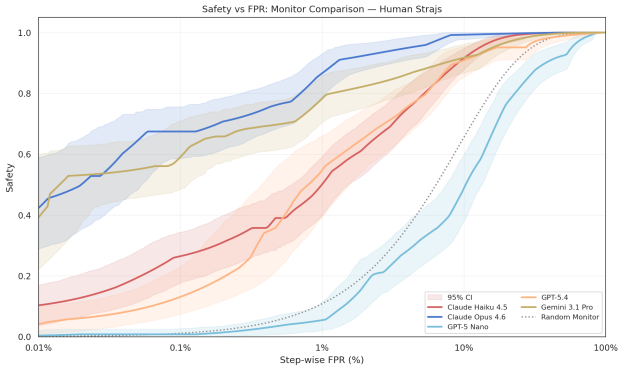


Figure 33. Safety vs step-wise FPR for five monitors against LaStraj (human-crafted) attacks. Every monitor shows lower safety at low FPR than against model-generated attacks (cf. Figure 5).

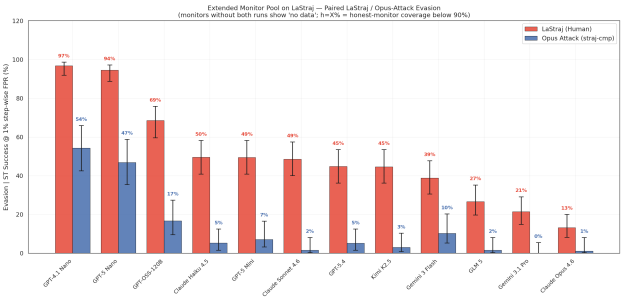


Figure 34. The full extended monitor pool evaluated against LaStraj (human sabotage trajectories) and the matched Opus-attack run (*straj-cmp*) on the same (main task, side task) pairs. Bars show evasion given ST success at 1% step-wise FPR with Wilson 95% CI. The broad pattern is similar between LaStraj and the matched model attack, though the exact ranking is not strictly preserved.