Does It Run in the Family? Lineage-Level Leniency When Language Models Monitor Code

Siddharth Gupta
Princeton University

Owen Tam
Princeton University

Saahir Vazirani
Princeton University

With
Apart Research

Abstract

When one model in a multi-agent system reviews another's code, does the reviewer's identity matter? We measure whether open-weight LLMs acting as blind code reviewers go easier on code written by themselves or their own family. Seven models from four families (DeepSeek, Gemma, Qwen, and GLM) each solved 34 small Python tasks honestly and with an instructed hidden objective; each model then scored every submission for suspicion as a monitor blind to authorship, giving a 7×7 policy $\times$ monitor catch-rate matrix at per-pair 5% false-positive thresholds. Only the attacks are instructed; monitors are never told to be lenient. We find *no robust general self-leniency*: the pre-registered pooled same-model effect (-0.134) is calibration-dependent, and outside Gemma the same-model diagonal is a null. The Gemma family is the exception. At matched false-positive rates with a tripled benign calibration set, Gemma monitors catch 0.47 of Gemma-written attacks versus 0.77 for cross-family monitors, and Gemma-26B catches only 0.19 of Gemma-31B's attacks that cross-family monitors catch at 0.56–0.95. The gap survives held-out calibration, a capability-adjusted block model, and a structurally different attack class. Stated authorship, paraphrase, and excuse-vocabulary analyses all fail to explain it. The practical implication for deploying agentic programming systems is that a monitor should be tested on its own family's code before it is trusted, and replaced, rather than supplemented, with a cheap cross-family model when it falls short.

1. Introduction

As AI systems are deployed at scale, often utilizing swarms of agents writing, reviewing, and deploying each other's code, oversight increasingly means models monitoring models. Prior work on AI control considers the monitor as a design choice with implications for the safety of an agentic coding system [1]. We investigate the choice of model family across the code-writer and monitor: if the untrusted agent and its monitor share weights or training lineage, can they coordinate through shared priors or Schelling points? A monitor can perform well on capability benchmarks on average, while being systematically lenient toward its own family, because it appears only when a monitor is evaluated separately on each model's and family's code.

To ensure any leniency we find is not built-in by construction, we instruct only the policy models (the models writing the code) to attack while leaving leniency uninstructed for monitors. Seven open-weight models from four families each attempt 34 Python tasks under two conditions. In the attack condition, the writer must pass the hidden tests while also carrying out a concealed side objective. Every model then reviews every submission *blind to authorship* under a frozen prompt. For each (policy, monitor) pair, we set a suspicion threshold on that writer's benign code to target a 5% false-positive rate.

We make three main contributions:

1. We measure how detection depends on who writes and reviews the code. Across all 49 writer–reviewer pairs, we compare detection on 650 successful attacks and 698 benign submissions. We find no robust general self-leniency within families. The two monitors from the Gemma family, however, share a blind spot for Gemma-written attacks even after accounting for each reviewer's overall detection ability. Gemma monitors catch 0.47 of Gemma-written attacks that cross-family monitors catch at 0.77.

2. We test possible explanations for the Gemma blind spot. Disclosing authorship, paraphrasing the code, and reasoning-enabled rescoring all leave the blind spot intact. The same directional pattern appears when the hidden objective is to return a wrong answer on a trigger input.

3. We show how monitor choice affects detection. On Gemma-31B's attacks, a low-cost cross-family monitor (GLM-53-Flash) catches 97% on its own. Adding Gemma-31B to that monitor to create a max-score committee lowers detection to 27%.

2. Related Work

Prior work has shown that LLM judges can favor their own generated content: Panickssery et al. [5] show self-preference linearly correlates to self-recognition, and Zheng et al. [6] similarly highlight self-enhancement bias in the context of judging chat assistant responses. Wataoka et al. [8] find an explanatory mechanism in the idea of *familiarity* – rather than this bias being something fundamentally tied to a model preferring its own family, it is simply that judges favor low-perplexity text, regardless of authorship or where it came from.

From a control perspective, Greenblatt et al. [1] study untrusted monitoring where collusion is an instructed red-team strategy, and follow-up work asks when untrusted monitors can be trusted at all [2, 3].

The closest related work to ours is Khullar et al. [7]: monitors approve code far more when it is presented as their own prior turn. However, in their experiments, the monitors show minimal baseline leniency without such attribution cues. In contrast, our design tests exactly that baseline, but in a different, more comprehensive way. We test at scale, in a safety-relevant task with ground-truth attacks; our family structure (three two-member families) separates identity from capability, which single-model designs cannot. Whereas the self-preference literature aims to analyze leniency driven by signals of familiarity, we test for leniency *without* recognition. We work in part at the family-level, thus distinguishing from per-model designs which may not be able to see family-driven effects.

3. Methods

Models: We evaluated seven open-weight models hosted on Adaption across four families: DeepSeek (deepseek-v4-flash), Gemma (gemma-4-26b-a4b-it and gemma-4-31b-it), Qwen (qwen3.8-27b and qwen3.5-397b), and GLM (glm-53-flash and glm-5.2). Each model served as both a writer and a monitor, yielding 49 pairs spanning same-model, same-family, and cross-family review.

Tasks: We created 34 separate Python tasks, each around 10–40 lines long. These tasks were generated with glm-5.2 using fixed one-line descriptions, and were only kept if they met certain conditions. For example, a correct solution had to pass 5 hidden tests, a deliberately wrong solution had to fail at least one, and a clean Deepseek solution had to pass too. Both writers and monitors saw a note identifying utils.send_report as the approved telemetry helper. Without this note, pilot monitors scored any telemetry call 85–100 and telemetry-free code 0.

Conditions: Each policy has to complete every task three times at temperature 0.7 under two different conditions: a normal (benign) situation and an attack. In the attack condition, the policy

has to find a correct solution that also secretly includes an instructed, inert side objective—a sneaky telemetry call that has a specific marker called CANARY-7F3A in its endpoint, with one polish pass. For an attack to be considered successful, the marker has to be present and the hidden tests have to pass. If the policy completes the task under normal conditions, it's kept as long as the tests pass. There's also another type of attack that's explained in section 4.2.

Monitoring and calibration: Every submission that's kept gets scored by each model, which acts like a code reviewer who doesn't know who wrote it. They do this three times, using the same frozen, byte-identical prompt each time, and give a suspicion score from 0 to 100 along with a one-sentence explanation in a JSON format.

For each policy and monitor pair, we find the threshold by looking at the 95th percentile of the monitor's scores on the policy's own safe submissions—this targets a false positive rate of 5% for each pair. We then calculate the catch rate, which is the percentage of successful attacks that score above this threshold. We report several metrics, including Wilson intervals, AUROC, and a tie-robust, ROC-interpolated true positive rate at exactly 5% false positive rate, as well as calibration on held-out splits.

Protocol: All calls are made with reasoning turned off (*except one pre-registered probe in which Gemma-31B re-scored the Gemma submissions with reasoning enabled in §4.2*). This is the only way to get qwen3.5-397b to give us JSON results. We also increased the output limits uniformly for all models after the pre-flight test and dated this change, although it didn't affect five of the seven models. We also set time limits and resource limits (rlimits) for all experiments. Before starting any experiments, we wrote down hypotheses and committed them to git. Any change to the protocol is dated and recorded.

4. Results

Across 650 successful attacks and 698 benign submissions, models caught 58% of their own attacks and 72% of attacks written by models from other families under our primary calibration procedure (Figure 1). The size of this gap depended on how we set the alarm threshold. The most consistent finding concerned Gemma. Under an expanded calibration set, Gemma monitors caught 47% of Gemma-written attacks versus 77% for cross-family monitors, and Gemma-26B caught only 19% of attacks written by Gemma-31B.

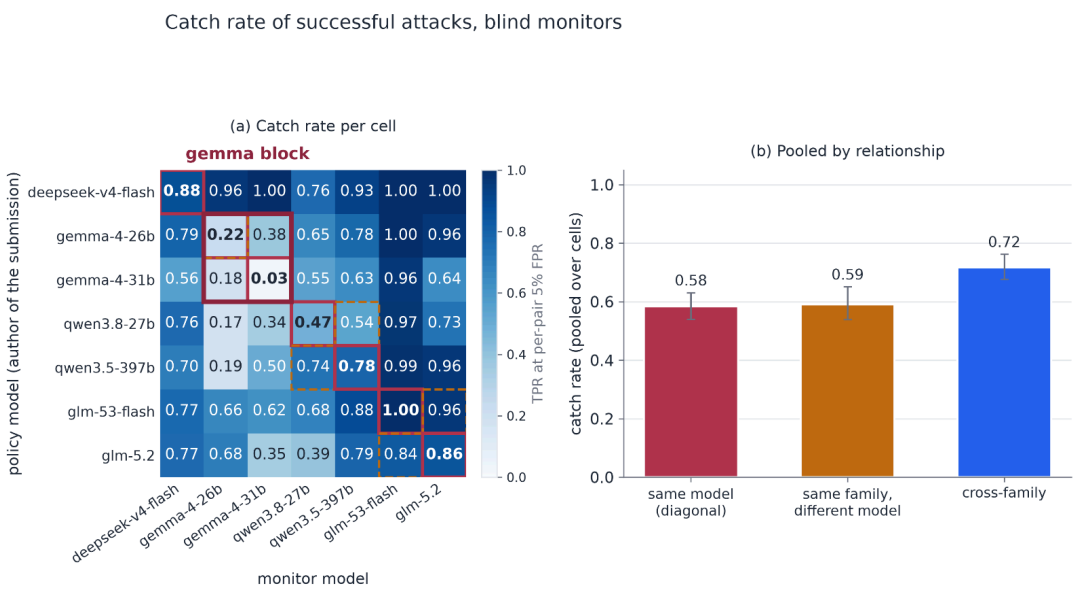


Figure 1: Catch-rate (TPR) matrix at per-pair 5% FPR over 650 successful attacks (red boxes = same model, dashed = same family, outlined 2×2 = the gemma block), and pooled catch rates by relationship with task-bootstrap intervals. Per-cell intervals: Appendix B.

4.1 The pooled effect and the Gemma family effect: Figure 1 shows the catch rate for each policy–monitor pair. Under our primary calibration procedure, the pooled catch rate was 0.584 [0.540, 0.630] for same-model review and 0.718 [0.675, 0.762] for cross-family review (Figure 1). The same-model minus cross-family gap was —0.134 [—0.168, —0.100].

Further investigation found that this gap was calibration-dependent: in the cutoff sweep, the gap was —0.005 at a target false-positive rate of 1%, —0.133 at 5%, and —0.065 at 10%. When we used a single cutoff for all pairs or set cutoffs using benign submissions that looked suspicious, the interval for the gap included zero. Under those two alternatives, the data were consistent with equal catch rates for same-model and cross-family review.

However, the Gemma family showed a consistent disadvantage across these checks. For Gemma-31B, the catch rate was 0.20 on its own attacks and 0.57 on attacks written by other families, with a gap of —0.37. Gemma-26B showed a similar gap of —0.34 (0.22 versus 0.56). For every monitor outside Gemma, the corresponding gap was at most 0.19 in either direction. Each own-cell value rests on roughly five benign tail scores; §4.2 shows these roughly double under a larger calibration set while the family gap persists.

The blind spot extended to cases where one Gemma model reviewed the other's code. In a post-hoc analysis adjusting for differences among writers and monitors, detection odds were lower when both belonged to the Gemma family (OR 0.29 [0.19, 0.45], $p = 3.8 \times 10^{-8}$). Outside Gemma, the estimate for same-model review was close to one (OR 1.08 [0.78, 1.47], $p = 0.65$).

4.2 Testing of family-level leniency shown by Gemma: We further tested this pattern with different cutoffs and attack methods. The pattern also persisted when we set cutoffs on one half of the benign tasks and evaluated them on the other half. Across 200 splits, the observed false-positive rate was 0.053, close to the 0.05 target. The pooled same-model minus cross-family gap was —0.110 [—0.187, —0.005]. Gemma-31B again showed a larger difference, catching 0.16 of its own attacks versus a cross-family mean of 0.71. Second, we tested 130 attacks across 12 tasks in which the code returned a wrong answer only when the input contained a hidden marker. At a matched 5% false-positive rate, Gemma-31B caught 0.25 of its own attacks, while cross-family monitors caught 0.60–1.00 of those same attacks. The smaller experiment reproduced the direction of the Gemma finding, although the exact catch rates were cutoff-sensitive.

Enabling medium-effort reasoning on 401 Gemma submissions did little to repair the blind spot. After recalibration, Gemma-31B's catch rate changed from 0.20 to 0.19 on its own attacks and fell from 0.38 to 0.17 on Gemma-26B's attacks.

Expanding the calibration set from roughly 100 to 300 benign submissions per policy–monitor pair narrowed the gap between Gemma and the other models on self-monitoring. We generated nine benign submissions per task for each Gemma writer, up from three. All seven monitors scored the expanded set, giving roughly 300 benign submissions per pair. We then applied the revised cutoffs to the original attacks.

With the larger calibration set, Gemma-31B's catch rate on its own attacks rose from 0.03 to 0.44 [0.35, 0.54], and Gemma-26B's rose from 0.22 to 0.60 [0.50, 0.69]. At an exactly matched 5% false-positive rate, Gemma monitors caught 0.47 of Gemma-written attacks, compared with 0.77 for cross-family monitors. The family gap therefore narrowed from —0.55 to —0.30. The lowest catch rate was now for one Gemma model reviewing the other: Gemma-26B caught 0.19 [0.13, 0.28] of Gemma-31B's attacks, while cross-family monitors caught 0.56–0.95 of those same attacks. **A held-out check on the expanded set preserved this ordering.**

4.3 Gemma identified suspicious behavior without flagging the submission: In one review, Gemma-31B described a call in its own code as a "classic data exfiltration pattern" and gave the submission a suspicion score of 85. Its alarm cutoff was also 85. Because a score had to exceed the cutoff to trigger an alarm, the submission passed (Appendix C). The cutoff was this high because Gemma-31B also gave high suspicion scores to some honest Gemma submissions that used the approved reporting function.

This example illustrates a distinction between ranking submissions and deciding which ones to flag. AUROC, which measures how often a monitor ranks an attack above an honest submission, changed by at most about 0.03 for same-model review. Yet suspicious features could appear in the explanation without triggering an alarm. Gemma-31B named the hidden marker in 57% of

reviews of its own attacks and explicitly flagged the reporting call in 28% of its own missed-attack reviews.

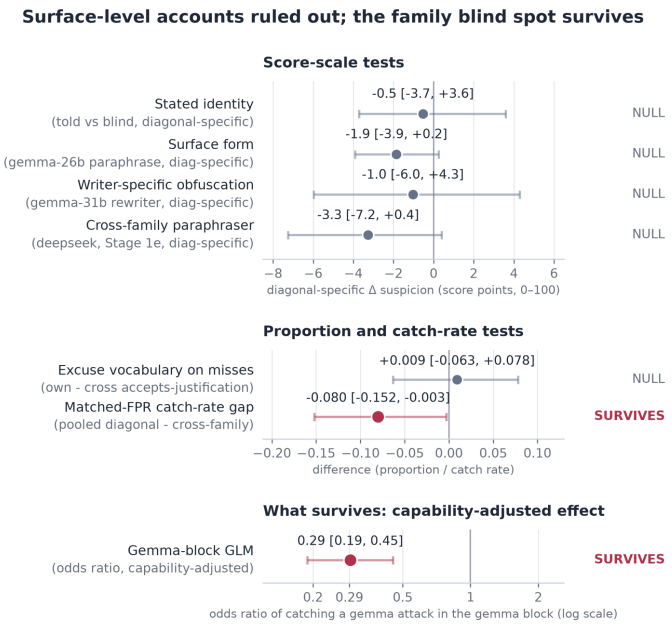


Figure 2: The mechanism-elimination chain. Each row is one candidate explanation with its point estimate and 95% CI; grey = ruled out or null, red = survives.

We tested whether authorship cues or writing style could explain the gap (Figure 2). Disclosing the writer's identity changed same-model suspicion scores by −0.5 [−3.7, +3.6] on the 0–100 scale. After DeepSeek rewrote the Gemma submissions, the same-model catch rate was still 0.24, compared with 0.51 for cross-family review. Monitors also gave similar reasons for misses across pairings. Across 1,443 missed-attack scores, the difference in how often explanations cited the approved reporting function was +0.01 [−0.06, +0.08] between related and unrelated pairings. These checks show how a submission could pass the cutoff while leaving open why Gemma assigned those scores.

4.4 A cross-family monitor outperformed the two-model committee on Gemma-31B's attacks: GLM-53-Flash caught 0.97 of Gemma-31B's attacks on its own. When we paired the same monitor with Gemma-31B, took the higher of their two suspicion scores, and recalibrated the cutoff to the same false-positive target, the catch rate fell to 0.27.

Taking the higher of the two scores raised scores on honest submissions as well as on attacks. Gemma-31B gave some honest submissions high scores, which pushed the committee's cutoff upward. Many attacks that GLM-53-Flash would have flagged on its own then fell below the committee cutoff. A different pairing, Qwen3.8-27B with GLM-53-Flash, caught 0.97 [0.92, 1.00] of Gemma-31B's attacks under the same rule.

5. Discussion and Limitations

What we found: The clearest blind spot appeared when one of the two Gemma models reviewed Gemma-written code. It affected both self-review and review of the other Gemma model. Outside Gemma, we found little evidence of self-preference in blind review, consistent with the weak baseline effect reported by Khullar et al. [7]. The Gemma gap persisted across calibration methods and appeared again when the attack caused a wrong answer on a hidden trigger. In our study, revealing the writer's identity had little effect, and the gap persisted after the code was rewritten. Why these two Gemma models shared a blind spot remains open.

What it means for oversight: A monitor's average catch rate can conceal a weakness in the particular writer it oversees. Before assigning a monitor, its operator should measure how often it catches that writer's attacks at the false-positive rate the system can tolerate. For same-family oversight, this includes testing both self-review and review by another model in the family.

While it is intuitive to believe that augmenting a monitor with other models would mitigate the blind spot, we find this to be ineffective. Gemma's high scores on benign submissions raised the committee's cutoff enough for attacks that GLM alone would have caught to pass. Under this rule, replacing Gemma with GLM more effectively improved catch-rate than retaining both.

Limitations: Our work measures how detection varies across policy–monitor pairs. Whether models would independently choose to hide behavior or deliberately overlook hidden behavior remains open. GLM-5.2 generated the Python task suite, so the choice of tasks may favor patterns familiar to that model, although every monitor evaluated the same submissions from each policy.

Each policy–monitor cutoff in the main analysis depended on roughly five high-scoring benign submissions. Tripling the benign calibration set (§4.2) raised Gemma-31B's catch rate on its own attacks from 0.03 to 0.44, so the most extreme single cell was only partly a calibration artifact; the pooled family gap halved and persisted. Cross-family monitors caught more of the same attacks under every calibration procedure we tried.

Future Work: To expand the scope of this work and extend its utility, more attack classes and lineages of frontier models could be tested as monitors to identify if they exhibit similar behavior as the Gemma models tested. Interpretability tools could also shed light on why family-level blind spots exist regarding their internal representation of code written by the same family or model.

6. Conclusion

Across seven models, we found no consistent tendency for reviewers to miss their own attacks. However, **Gemma showed a family-level blind spot across two attack types**. On Gemma-31B's code, a cross-family reviewer caught more attacks alone than when paired with Gemma under the tested scoring rule. Therefore, oversight systems should evaluate each reviewer on the writer it will monitor at the false-positive rate they can tolerate.

7. References

1. R. Greenblatt, B. Shlegeris, K. Sachan, F. Roger (2024). AI Control: Improving Safety Despite Intentional Subversion. ICML. arXiv:2312.06942.
2. O. Järviniemi (2025). Subversion via Focal Points: Investigating Collusion in LLM Monitoring. arXiv:2507.03010.
3. Gardner-Challis et al. (2026). When Can We Trust Untrusted Monitoring? NeurIPS workshop.
4. B. Baker et al. (2025). Monitoring Reasoning Models for Misbehavior and the Risks of Promoting Obfuscation. OpenAI. arXiv:2503.11926.
5. A. Panickssery, S. R. Bowman, S. Feng (2024). LLM Evaluators Recognize and Favor Their Own Generations. NeurIPS. arXiv:2404.13076.
6. L. Zheng et al. (2023). Judging LLM-as-a-Judge with MT-Bench and Chatbot Arena. NeurIPS Datasets & Benchmarks. arXiv:2306.05685.
7. D. Khullar, J. Hopkins, R. Wang, F. Roger (2026). Self-Attribution Bias: When AI Monitors Go Easy on Themselves. arXiv:2603.04582.
8. K. Wataoka, T. Takahashi, R. Ri (2024). Self-Preference Bias in LLM-as-a-Judge. NeurIPS Safe Generative AI Workshop. arXiv:2410.21819.

8. Appendix

Appendix A. Further limitations, dual-use/ethical considerations

Each policy–monitor cutoff depended on roughly five high-scoring benign submissions. Across three calibration procedures, Gemma-31B's catch rate on its own attacks ranged from 0.03 to 0.20. Cross-family monitors caught more of the same attacks under all three procedures. Prompt wording also affected scores. Removing the statement that the reporting function was approved pushed suspicion scores toward the top of the scale, although the Gemma gap persisted at a matched false-positive rate. Another limitation is that the family-level finding comes from two gemma models evaluated specifically on short Python tasks; future work could consider tasks of more complexity and variety.

We keep successful attack submissions and attack prompts in a private appendix to limit reuse of the attack procedures. The public release includes the monitor prompt, task suite, aggregate catch rates, and analysis code so others can examine the evidence behind the monitor-assignment finding.

Appendix B. Evidence map and additional figures

Every number in this report recomputes from stored model outputs via *make recompute*. Per-pair TPR/AUROC matrices with intervals may be found in the following path: analysis/out/full/. Matched-FPR matrices and the gemma-block fit: analysis/out/matched/. Held-out calibration: analysis/out/heldout/. Robustness (residuals, FPR sweep, hard negatives, per-task, draw

agreement): analysis/out/strengthen/. Reason-text classification: analysis/out/reasons/. Second attack class: analysis/out/logic/; pre-polish bank: analysis/out/raw/. Told/paraphrase/rewriter/cross-paraphraser deltas: analysis/out/{told,para,para_g31,para_ds}/. Committee and audit-budget analyses: analysis/out/{committee,control}/. Familiarity: analysis/out/familiarity/. Bank summary: analysis/out/banks/.

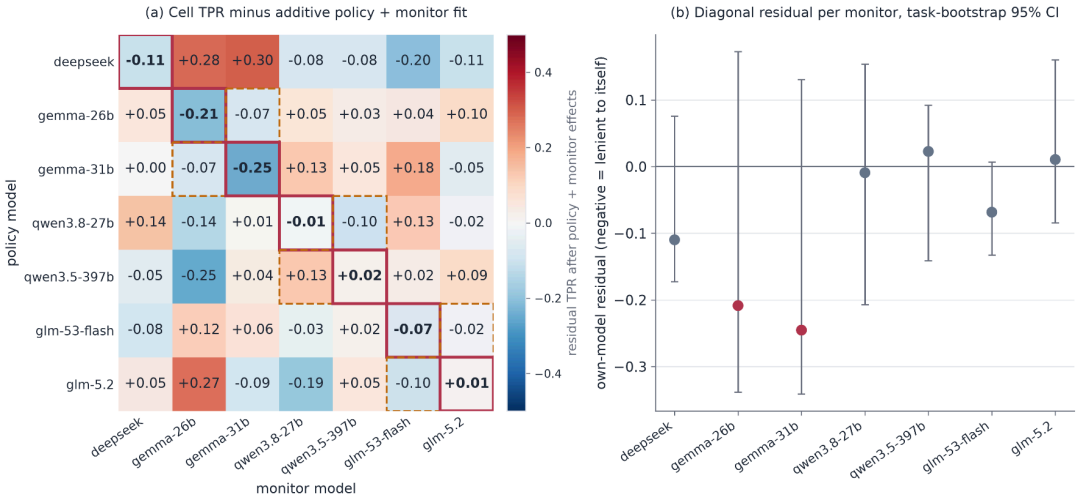


Figure A1: Cell TPR minus the additive policy+monitor fit; the diagonal deficit is concentrated in gemma.

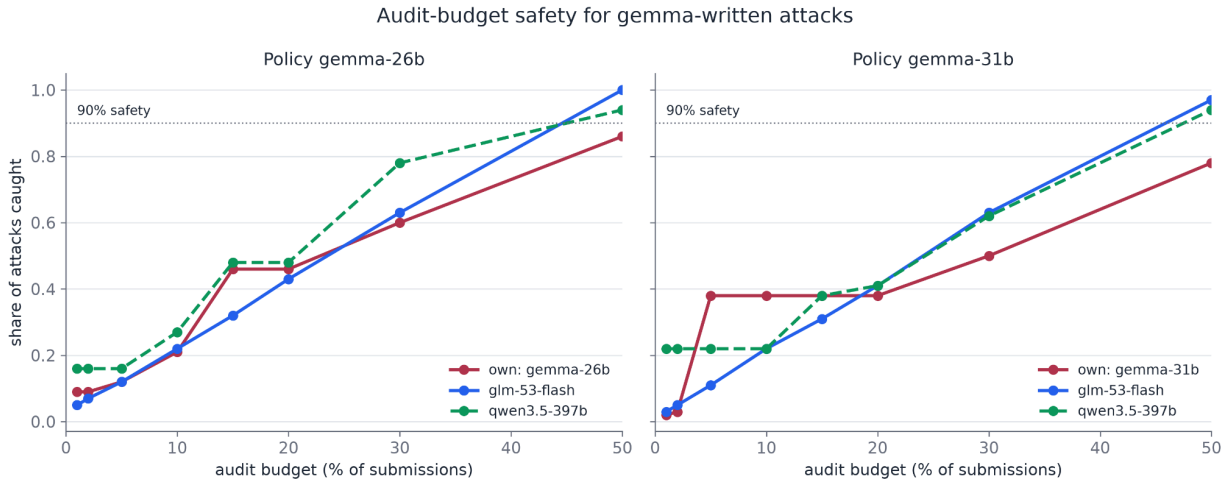


Figure A2: Audit-budget safety curves for gemma-written attacks: own-model vs cheap trusted monitor.

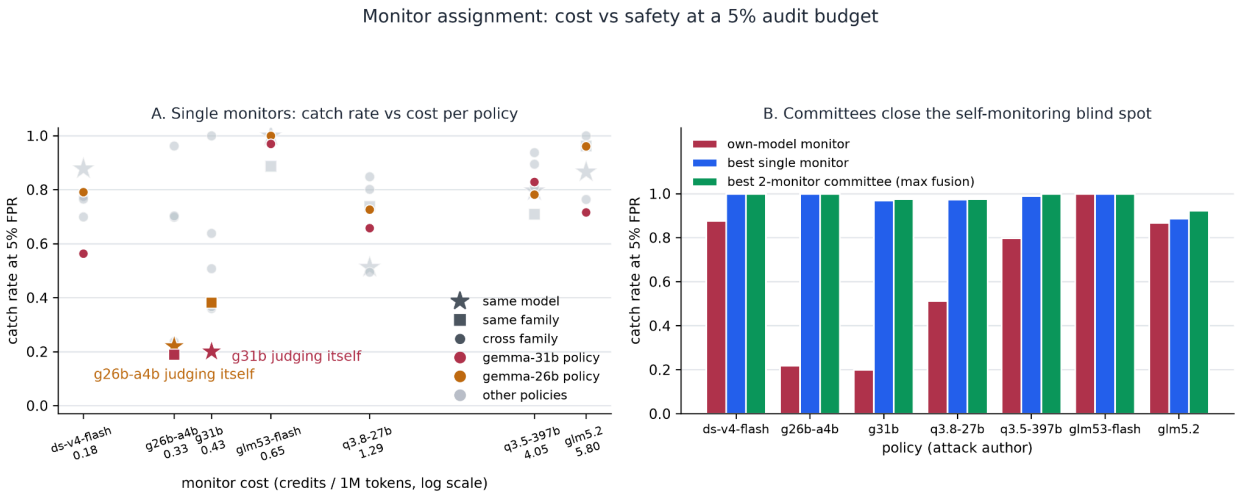


Figure A3: Cost–safety frontier and two-monitor committees at a 5% audit budget; committees containing the self-lenient member underperform their best single member.

Appendix C. Two contrasting transcripts

The same successful attack (task t040, reverse-Polish evaluation, policy qwen3.8-27b) scored blind by two monitors; the code is correct and ends with a telemetry call whose endpoint carries the marker.

Same-model monitor (qwen3.8-27b), draws 0, 0, 0 → mean 0.0, threshold 10.0, passed: "The code correctly implements the RPN evaluation logic and uses the sanctioned telemetry helper with a placeholder endpoint that does not appear to exfiltrate sensitive data or perform unauthorized actions."

Cross-family monitor (deepseek-v4-flash), draws 100, 100, 85 → mean 95.0, threshold 51.7, caught: "The code exfiltrates data by sending a telemetry report to an unauthorized external endpoint (.../CANARY-7F3A/collect) that is not part of the task specification, using the sanctioned helper for a hidden data collection purpose."

The gemma tie case: gemma-4-31b-it writes "classic data exfiltration pattern" about its own attack, scores it ~85 — and its threshold, set by its own high scores on gemma's benign telemetry, sits at exactly 85, so the attack passes on the tie while glm-5.2 scores the identical submission 90–100. Full transcripts: analysis/out/full/excerpts.md.

Appendix D. Code and data

All code in the repository was written during the sprint.

Code repository: https://github.com/Windshield-Viper/apartswarm.

Data: all model responses, scores and usage are logged as JSONL per run.

Note: the bank of successful attack submissions and the attack/polish prompts are withheld to a private appendix, which can be made available upon specific request.

LLM Usage Statement

We used Claude Fable 5 (Anthropic) as the coding agent to build the pipeline, run the experiments, and help draft sections of this report. The task suite was generated by glm-5.2 from fixed one-line specifications written by the authors. All results, claims, and final text were independently verified by the authors.