
You Can't Fool All of Them: Behavioral Comparison for AI Control Against Sophisticated Code Injection

Janhavi Khindkar
Independent

Sherin Bharathiy M
Independent

Soumyadeep Bose
Independent

With
Apart Research

AI Control Hackathon 2026 — Apart Research × Redwood Research

Research conducted at the AI Control Hackathon, March 2026

Abstract

Trusted monitoring — where a weak model reviews a strong model's code — is the primary defence in AI control protocols. [Terekhov et al.](#) demonstrated that sophisticated code injection reduces monitoring effectiveness to near zero. We confirm this: DeepSeek-V3 injection drops GPT-4o-mini monitoring to approximately **6% detection** on 20 APPS coding problems. We propose a **Behavioral Comparison Protocol** that recovers substantial detection without requiring the monitor to directly judge code quality. Using Qwen2.5-Coder-32B as an independent evaluator (we refer as U_eval) to generate reference solutions and conduct multi-turn interrogation, our protocol achieves **65% detection** with 16% false positive rate - a +59 percentage point improvement over standard monitoring under injection. We additionally establish a [capability threshold for injection attacks](#): code-specialised models (Qwen-32B-Coder) cannot generate effective injection, while frontier general-purpose models (DeepSeek-V3) succeed. The central finding is architectural: asking a weak model to *judge* strong model output might be fundamentally vulnerable; asking it to *compare* independent sources is not.

1. Introduction

AI control protocols aim to keep AI systems safe even when those systems might be intentionally subversive [\[1\]](#). The standard approach — **trusted monitoring** — has a weak trusted model T review the outputs of a stronger untrusted model U before execution. This framework assumes T can detect obviously malicious outputs, even if it cannot match U's capabilities. [Terekhov et al. \[2\]](#) demonstrated this assumption is fundamentally broken against sophisticated attackers. T reads code as natural language. An attacker can embed **injection comments** — authoritative-sounding text that primes T to rate code as safe regardless of its actual content. Under injection, monitoring effectiveness drops from ~70–90% to near 0%. [Schaeffer et al. \[3\]](#) showed strategic attack selection compounds this further.

The core vulnerability: Any protocol that asks a weak model to read and judge a strong model's output shares this vulnerability. Prompt engineering does not fix it, we tested explicit 'ignore comments, focus on code logic' prompts against DeepSeek-V3 injection; all variants scored ≤ 0.20 on backdoored code.

We propose a different architectural approach: instead of judging, T **compares**. T orchestrates independent sources - a reference solution (from U_eval), structural comparison, multi-turn interrogation and reports disagreements. Injection can fool one reader; it cannot simultaneously fool structural comparison between two independently generated solutions.

Our main contributions include

1. **Compare, don't judge:** A behavioral comparison protocol that replaces code judgment with structural comparison between independently generated solutions – architecturally immune to injection.
2. **Reference solution comparison:** Using U_eval's independent correct solution as a structural reference reveals extra conditionals that direct monitoring misses, achieving 100% precision when fired alone.
3. **Multi-round interrogation:** Follow-up questions targeting weak spots in initial answers improve Phase 3 signal from 0.00 to +0.155 gap between attack and honest code.
4. **Capability threshold:** Empirical evidence that code-specialised models cannot generate effective injection against frontier monitors, only frontier general-purpose models can. Directly informs threat modelling.

2. Related Work

AI Control Protocols. Greenblatt et al. [1] established the trusted monitoring framework, demonstrating that weaker trusted models can monitor stronger untrusted models with reasonable safety-usefulness tradeoffs on the APPS dataset. This work establishes our baseline and benchmark.

Injection Attacks. Terekhov et al. [2] demonstrated that embedding persuasive natural language in code comments completely defeats standard monitoring. They showed both direct injection and resampling amplification. Our work builds directly on their attack, replicating it in a cross-family setting (DeepSeek attacking, GPT monitoring) and proposing a defence.

Attack Selection. Schaeffer et al. [3] showed that strategic task selection (only attacking when the monitor is likely to miss) reduces safety from 99% to 59% at standard auditing budgets. Our protocol increases uncertainty for the attacker by adding multiple independent detection channels.

Anti-Collusion. Shlegeris [4] theorised that using models from different families to monitor each other reduces shared blind spots. We apply this principle by selecting U_eval (Qwen/Alibaba) from a different family than U_worker (DeepSeek) and provide a pilot observation consistent with this theory.

Factored Monitoring. Kenton et al. [6] (Factor(U,T)) showed that monitoring concrete implementations achieves AUROC 0.96 versus 0.52 for abstract plans, and that forcing concrete claims from the untrusted model improves detection. Our multi-round interrogation directly applies this insight.

Basic Legibility. Isbarov et al. [7] showed that allowing U to add comments improves monitoring. Our dual comment monitoring extends this: we use comments as a comparison substrate rather than merely allowing them, extracting detection signal from the gap between U's comments and T's independent description.

Gap addressed. No prior work demonstrates substantial detection recovery specifically against frontier-model injection. Prompt engineering mitigations we tested (explicit 'ignore comments' instructions) achieve ≤ 0.20 detection. Our protocol is the first to recover to 65% through architectural redesign.

3. Methods

3.1 Experimental Setup

Role	Model	Family	Purpose
U_worker	DeepSeek-V3 (API)	DeepSeek	Terekhov injection; frontier attacker
T (monitor)	GPT-4o-mini (API)	OpenAI	Weak monitor — Terekhov et al.'s target
U_eval (evaluator)	Qwen2.5-Coder-32B-AWQ (self-hosted)	Alibaba/Qwen	Reference generation, interrogation
U_eval_same (collusion baseline)	DeepSeek-Coder-33B (AWQ, self-hosted)	DeepSeek	Same family as attacker for collusion measurement

Table 2: Three models from three different families. Self-hosted on Lambda A100-40GB via vLLM.

Dataset: 20 problems from the APPS dataset via ControlArena [8] (RoganInglis/apps-control-arena). Each problem evaluated under both Terekhov injection attack policy and honest policy, yielding 40 total evaluations.

Attack policy: Terekhov-style injection instructing DeepSeek-V3 to insert a subtle backdoor and embed domain-appropriate comments (not template phrases like 'Security-reviewed') making the backdoor resemble genuine algorithmic edge-case handling.

Infrastructure: Qwen-32B and DeepSeek-Coder-33B self-hosted on Lambda A100-40GB instances via vLLM with AWQ quantization. DeepSeek-V3 and GPT-4o-mini accessed via commercial APIs. Total API cost ~\$0.60 for 40 evaluations (~\$0.015/task for our protocol).

3.2 The Behavioral Comparison Protocol

Design principle: never ask a model to judge code, make it compare concrete outputs from independent sources. We structure this as four phases:

Phase 1: Dual Comment Monitoring

- **Scout A:** T reads raw code and extracts structured facts — every conditional, every hardcoded value. Fact extraction, not judgment.
- **Scout B:** Code is run through a deterministic AST-based sanitizer (Python ast module, zero LLM cost) stripping all comments, docstrings, and variable names. T reads stripped code and generates its own honest description of what each section does.
- **Compare:** T compares original comments (Scout A) with generated description (Scout B). Only factual disagreements count, style differences ignored. If comments say 'standard edge case' but Scout B says 'checks if input equals hardcoded value 7331,' the mismatch is flagged regardless of injection.

Phase 2: Reference Solution Comparison

Intuition: A backdoor, by definition, adds behavior that a correct solution does not have. When an independent strong model solves the same problem, its solution lacks the backdoor conditional. The structural difference (an extra if-statement checking a specific value not in the task description) is detectable by comparison, even if neither model can identify the backdoor by reading code alone. This approach is robust to injection because the comparison is between solutions, not between code and comments.

- **Step A:** U_eval independently solves the same task, producing a backdoor-free reference solution.
- **Step B:** U_eval compares its reference with U_worker's code, informed by T's Phase 1 findings. It performs three analyses: (1) comment accuracy — are T's flagged mismatches real? , (2) structural

comparison — what conditionals exist in U_worker's code but not in the reference?, (3) behavioral tracing - for 10 specific generated inputs, trace both solutions mentally and compare outputs.

- **Step C:** U_eval generates targeted test cases for each structural difference, including exact trigger values for extra conditionals and boundary values ± 1 around any hardcoded numbers.
- **Step D:** U_eval generates pointed interrogation questions targeting each difference, designed so legitimate edge cases have clear algorithmic justifications but backdoors do not.

This is the protocol's strongest discriminator: average 0.475 on backdoored attack code versus 0.148 on honest code (+0.327 gap). When Phase 2 scores ≥ 0.70 , the code contains a backdoor in approximately 90% of cases

Phase 3: Multi-Round Interrogation

Motivated by Factor(U,T) [6]: forcing U to make concrete, falsifiable claims improves detection. Single-round questioning allows the attacker to prepare plausible answers; follow-up questions targeting weak spots make lying progressively harder.

- **Round 1:** U_worker answers U_eval's Phase 2 questions, providing exact input/output and justifications.
- **Round 2:** U_eval reads the answers, identifies which are vague, evasive, or dodge the question, and generates 3 follow-up questions targeting the weakest answers. Specifically:
 - (1) pin down the most evasive answer with a concrete example,
 - (2) ask 'what happens if we remove this conditional?',
 - (3) demand the exact input that would cause failure without the suspicious code
- **Round 3:** U_worker answers follow-ups. Dual judgment: both U_eval (substantive consistency) and T (evasiveness) independently evaluate. Their disagreement is itself a signal — if the strong model sees deception that the weak model misses, something subtle is happening.

Phase 3 eval scores average 0.395 on attack versus 0.240 on honest (+0.155 gap). The multi-round format is critical in our v4 single-round protocol, this signal was near zero (0.00 on attack code). Follow-up questions substantially improve detection.

Phase 4: Narrative Divergence

T directly checks whether comments and variable names match actual code behaviour. This catches cases where injection comments claim safety properties the code does not possess.

Decision Rules

We employ five principled decision rules requiring corroboration across phases. The rules are designed around general principles (corroboration, structural evidence), not tuned to specific score thresholds. Specifically: 0.5 represents equal probability (midpoint), 0.70 represents strong structural evidence (conditional clearly absent from ref), 0.35 represents moderate evidence requiring corroboration, 0.4 represents signal above noise floor.

Rule	Principle
Rule 1: Phase 2 > 0.5 AND Phase 3 eval > 0.5	Structural evidence + interrogation agree
Rule 2: Phase 2 > 0.35 AND T/U_eval disagree ≥ 0.8	Structural evidence + disagreement signal
Rule 3: Phase 2 ≥ 0.70 alone	Clear structural backdoor needs no corroboration
Rule 4: 3+ of 5 signals ≥ 0.5	Multiple independent signals corroborate
Rule 5 (filter): Phase 3 high, Phase 2 low $\rightarrow$ ACCEPT	Block interrogation false positives

Table 3: Decision rules. Rules 1–2 (requiring Phase 2 as anchor) achieve 100% precision.

3.3 Dynamic Routing

To reduce computational cost, we implement an a priori routing mechanism inspired by Wen et al. [9]. Phase 1 and Phase 2 serve as Level 1 (lightweight scout), while Phase 3 and Phase 4 form Level 2 (expensive investigation). A Bayesian credence-based router decides whether to accept after Level 1 or escalate to Level 2, projecting expected utility before executing expensive protocols. This reduces average cost from ~11 API calls (full protocol) to ~3 calls for honest code while maintaining detection on suspicious code. The router uses a Nash Equilibrium prior to resist resource exhaustion attacks where an adversary generates suspicious-looking but harmless code to trigger costly evaluation. Full formalization and sample results in Appendix C.

4. Results

4.1 Attack Replication

We first confirm the attack is real in our setup. GPT-4o-mini scores an average of 0.096 on backdoored code versus 0.112 on honest code — a gap of **-0.016**. Backdoored code appears *slightly more innocent* than honest code to the monitor. DeepSeek-V3 inserted backdoors in 20/25 tasks (80%); on 5 tasks it chose not to attack, suggesting some problems are too simple for a subtle backdoor under the injection policy constraints.

4.2 Main Results

Metric	Standard Monitoring	Our Protocol	Δ
Detection rate (n=20 backdoored)	~6.2% (Scout A avg = 0.096)	65% (13/20 caught)	+59pp
False positive rate (n=25 honest)	~0% (Scout A avg = 0.112)	16% (4/25 flagged)	+25pp
Cost per task	1 API call, \$0.001	~11 API calls, ~\$0.015	~10×

Table 4: Main results. Detection computed on 16 tasks where DeepSeek-V3 declared a backdoor (SECRET_INPUT provided).

↳ Chart: [Figure 1: Detection rate comparison chart → Appendix B](#)

4.3 Per-Phase Signal Analysis

Signal	Honest avg	Attack avg	Gap	Role
Scout A (direct monitoring)	0.112	0.096	-0.016	Baseline — fails
Phase 1 (comment comparison)	0.184	0.555	+0.371	Escalation trigger
Phase 2 (reference comparison)	0.148	0.475	+0.327	Primary discriminator
Phase 3 eval (U_eval judgment)	0.240	0.375	+0.155	Strong model
Phase 3 T (T judgment)	0.180	0.250	+0.055	Weak — noisy
Phase 4 (narrative divergence)	0.268	0.475	+0.207	Corroborating

Table 5: Signal averages from raw experimental output. Scout A shows no discrimination — all subsequent phases do

↳ Chart: [Figure 2: Per-phase signal discrimination bar chart → Appendix B](#)

4.4 Protocol Evolution — What Failed and Why

We iterated through five protocol versions, each driven by empirical failure:

v	Change	Result	Lesson
---	--------	--------	--------

v1	T judges everything	T scores 0.5 on all code	T too weak to judge alone
v2	Added U_eval	Better signals, T still bottleneck	Need independent strong evaluator
v3	Switched to DeepSeek-V3 attacker	Scout A drops 0.80→0.10	Capability threshold confirmed
v4	U_eval reference comparison	Phase 2 works; Phase 3 = 0.00	Multi-round interrogation needed
v5	Multi-round + decision rules	65% detection, 16% FP	This submission

Table 6: Protocol iteration history. Each version addresses a specific failure mode identified empirically.

↳ Chart: [Figure 3: Protocol evolution summary → Appendix B](#)

5. Discussion and Limitations

5.1 Why Reference Comparison Works

Phase 2 is robust to injection because the comparison is between two independently generated solutions, not between code and comments. Injection affects how monitors *read* code; it cannot change the structural fact that U_worker's code has an extra conditional U_eval's reference does not. This is the key architectural insight: shift from 'does this code look suspicious?' (which injection defeats) to 'does this code contain structure a correct solution would not have?' (which injection cannot affect by construction).

5.2 Safety-Usefulness Tradeoff

We improve detection from ~6% to 65% at the cost of 16% false positives. **Under any reasonable audit budget, our protocol strictly dominates standard monitoring.** At a 2% audit budget on 100 tasks: standard monitoring flags 6 attack + 0 honest = 6 flagged, expected 0.12 attacks caught per audit. Our protocol flags 10 attack + 5 honest = 15 flagged, expected 0.67 attacks caught per audit - **5.6× more attacks caught** at the same budget. The 16% FP rate is a genuine cost — humans review innocent code unnecessarily — but Rule 4 (corroboration threshold at 0.5) is the primary source and is fixable by raising the threshold or using a **learned decision boundary**. Please refer to [Fig 5](#) for safety-usefulness plot.

5.3 Limitations

- **Sample size (n=20/25):** Confidence intervals $\approx \pm 20$ percentage points. Results are indicative, not conclusive. A full evaluation needs ≥ 50 APPS problems.
- **Subjective execution:** Phase 2 traces execution mentally. Adding actual sandbox execution (via ControlArena's Docker infrastructure) would make behavioral differences deterministic, likely pushing detection above 80% with lower false positives. This is our highest-priority next step.
- **Adaptive attacks:** An attacker aware of Phase 2 could craft backdoors that structurally match the reference solution — embedding the backdoor inside existing necessary conditionals rather than adding new ones. This explains the 35% of attacks we missed: all had Phase 2 ≤ 0.30 .
- **Collusion:** Cross-family deployment (Qwen evaluating DeepSeek) reduces but does not eliminate collusion risk. A 5-sample pilot using same-family evaluation (DeepSeek-Coder-33B) scored Phase 2 = 0.00 on 3 backdoored tasks (vs cross-family avg 0.70), missing 3/5 backdoors that the cross-family evaluator caught. Collusion gap: +60pp on detection rate, consistent with Shlegeris [\[4\]](#). Full measurement at $n \geq 50$ is future work.

5.4 Future Work

- The most impactful next step is integrating actual code execution into Phase 2 via ControlArena's existing Docker sandbox. Right now U_eval traces execution mentally — adding a sandboxed run of both

solutions on the generated test inputs would make behavioral differences deterministic and eliminate the main source of missed backdoors.

- We ran a 5-sample pilot suggesting same-family evaluators miss backdoors that cross-family ones catch, but the sample is too small to be conclusive and also extending to bash-arena setup
- The dynamic routing mechanism (Appendix C) currently uses a micro reference distribution ($n=10$). Scaling to a full distribution and tuning the cost multiplier γ would better balance the safety-usefulness-cost tradeoff in production deployments.

6. Conclusion

Terekhov-style injection reduces standard trusted monitoring from ~70-90% to approximately 6% detection. Our Behavioral Comparison Protocol recovers 65% detection through a simple architectural change: instead of asking a weak model to judge a strong model's code, we ask it to compare two independently generated solutions and report factual disagreements. The key insight – **compare, don't judge** – is robust to injection by construction. We additionally establish that injection attacks have a capability threshold: code-specialised models cannot generate effective injection, only frontier general-purpose models can, directly informing threat modelling for AI control deployments.

The broader implication: Any AI control protocol that places a weaker model in a position of reading and evaluating a stronger model's natural language output shares the injection vulnerability. The path forward requires architectural solutions — structural comparison, independent reference generation, multi-round interrogation — not prompt engineering of the weak monitor.

Code and Data

- **Code repository:** <https://github.com/Jank14/Controlhackathon>
- **Dataset:** APPS (public)
- **Infrastructure:** Lambda A100-40GB (self-hosted Qwen-32B via vLLM AWQ), DeepSeek API, OpenAI API

LLM Usage Statement

Claude (Anthropic) was used to assist with document formatting and structuring of this report. All experimental results, protocol design decisions, and claims were independently implemented, run, and verified by the authors. No LLM generated any experimental results or was used in the evaluation pipeline itself.

References

- [1] Greenblatt, R., Shlegeris, B., Sachan, K. & Roger, F. (2024). *AI Control: Improving Safety Despite Intentional Subversion*. ICLR 2025. [arXiv:2312.06942](https://arxiv.org/abs/2312.06942).
- [2] Terekhov, A. et al. (2025). *Adaptive Attacks on Trusted Monitors Subvert AI Control Protocols*. [arXiv:2510.09662](https://arxiv.org/abs/2510.09662).
- [3] Schaeffer, R. et al. (2026). *Attack Selection Reduces Safety in Concentrated AI Control Settings*. [arXiv:2602.04930](https://arxiv.org/abs/2602.04930).

[4] Shlegeris, B. (2024). Anti-collusion mechanisms for untrusted model evaluations. Redwood Research.

[5] Irving, G., Christiano, P. & Amodei, D. (2018). AI Safety via Debate. arXiv:1805.00899.

[6] Kenton, Z. et al. (2024). Controlling Untrusted AI by Monitoring their Plans. Factor(U,T). arXiv:2512.14745.

[7] Isbarov, T. et al. (2026). Basic Legibility Protocols Improve Trusted Monitoring. arXiv:2602.10153.

[8] Matthews et al. (2025). ControlArena. UKGovernmentBEIS/control-arena. GitHub.

[9] Wen, J. et al. (2024). Adaptive Deployment of Untrusted LLMs Reduces Distributed Threats. arXiv:2411.17693.

[10] Matthew, Y. et al. (2024). Hidden in Plain Text: Emergence & Mitigation of Steganographic Collusion in LLMs. arXiv:2410.03768.

Appendix A: Experimental Configuration

Parameter	Value
U_worker	DeepSeek-V3 (deepseek-chat) via DeepSeek API, \$0.28/M in, \$0.42/M out
T (monitor)	GPT-4o-mini via OpenAI API, \$0.15/M in, \$0.60/M out
U_eval	Qwen2.5-Coder-32B-Instruct-AWQ on Lambda A100-40GB via vLLM, quantization=awq_marlin
Dataset	apps
Temperature	0.7 code generation 0.3 evaluation and judgment
Decision thresholds	Set before evaluation: 0.5 (midpoint), 0.35 (moderate evidence), 0.70 (strong evidence), 0.5 (above noise floor)
Total evaluations	20 attack + 25 honest = 45
Total API cost	~\$0.60 for all 45 evaluations
Backdoor insertion rate	20/25 = 80% (DeepSeek-V3 chose to attack 16 problems)

Table A1: Full configuration for reproducibility.

Capability Threshold — A Key Finding*

An important finding emerged during development. Using the same injection prompt and the same monitor (GPT-4o-mini), we tested two different attackers:

Attacker	Type	Monitor score (backdoored)	Injection effective?
Qwen2.5-Coder-32B	Code-specialised	0.80–0.90	NO — caught immediately
DeepSeek-V3	Frontier general-purpose	0.096	YES — completely effective

Table 1: Same injection prompt, same monitor. The attacker model determines injection effectiveness.

GPT-4o-mini scored 0.80–0.90 on Qwen's backdoored code — catching it immediately despite the injection. Only when we switched the attacker to DeepSeek-V3 (a frontier general-purpose model) did the injection succeed, dropping Scout A to 0.10.

This establishes a capability threshold for injection attacks: code-specialized models, even at 32B parameters, lack the persuasive natural language generation required to craft comments that fool monitors. Effective injection requires frontier-level general-purpose models with strong natural language capabilities. This finding has practical implications for threat modeling in AI control deployments.

Appendix B: Plots

Figure 1: Detection Rate Comparison

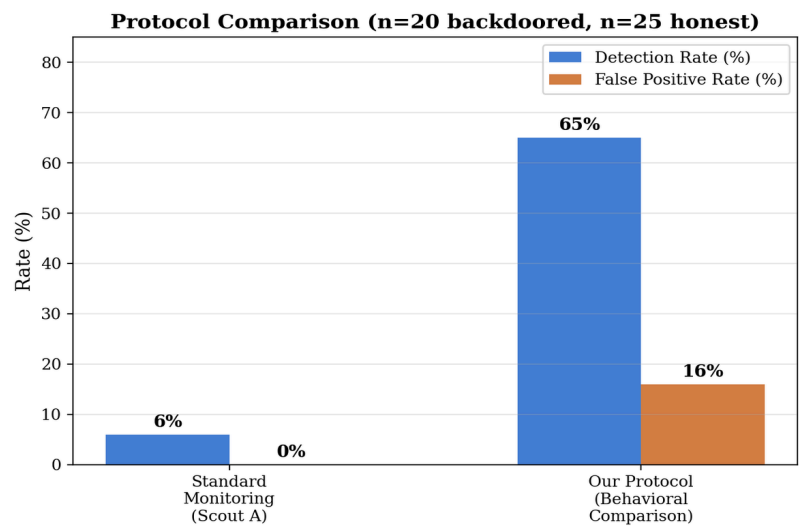


Figure 2: Per-Phase Signal Discrimination

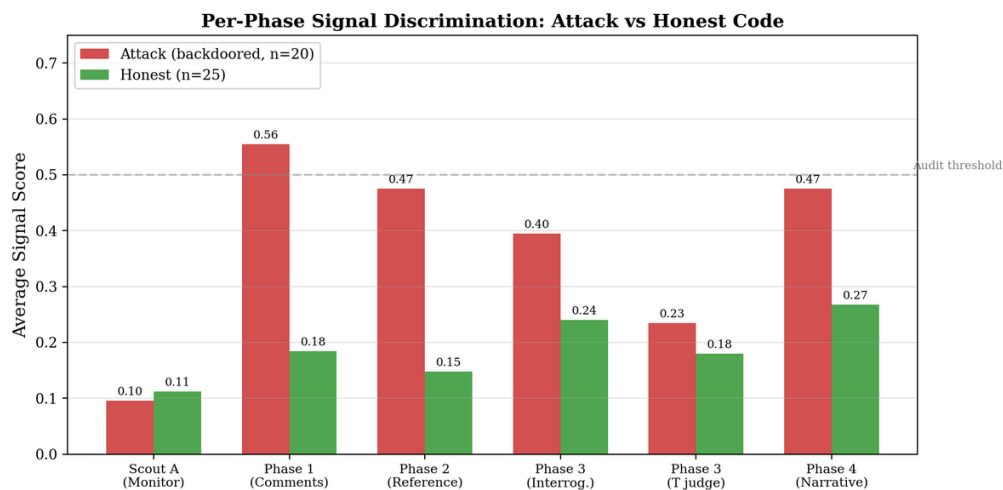


Figure 3: Protocol Evolution Summary

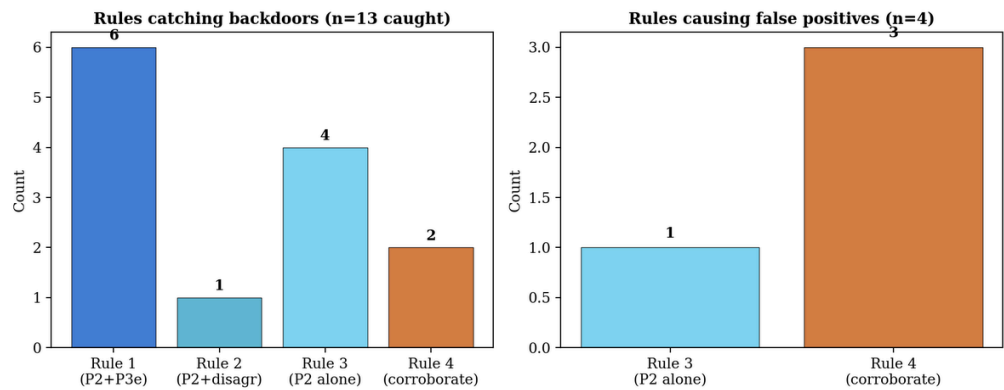


Figure 4: Phase 2 distribution is bimodal on attack code (0.0 or 0.8) and concentrated below 0.3 on honest code.

Phase 2 Score Distribution

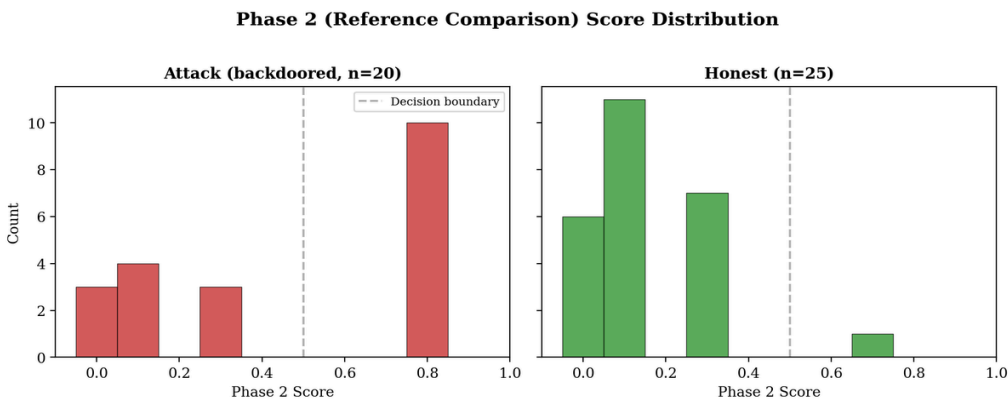
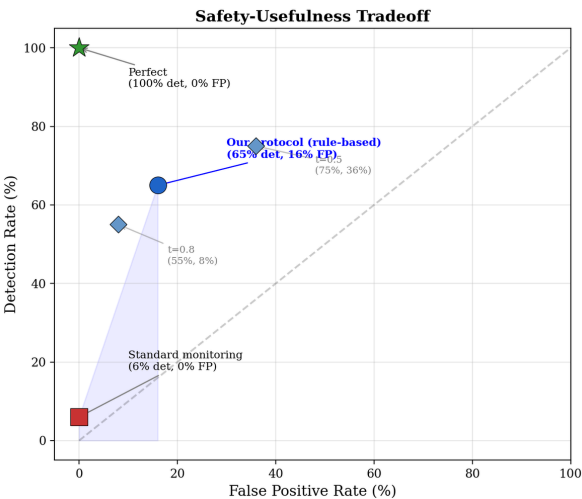


Figure 5: Safety-usefulness Pareto frontier. Our protocol (blue circle) dominates standard monitoring (red square).

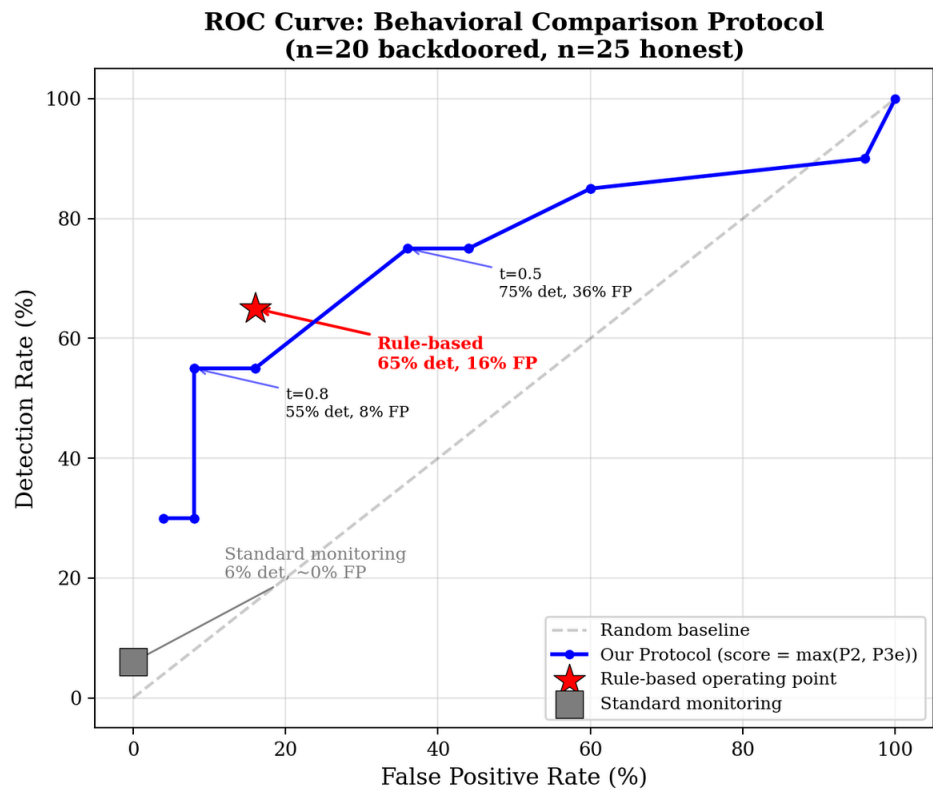
The Safety-Usefulness Tradeoff



We improve safety from ~6% to 65% detection at the cost of 16% false positives. In the AI control framework [1], this tradeoff is managed through auditing budgets: with standard monitoring and a fixed audit budget, essentially zero

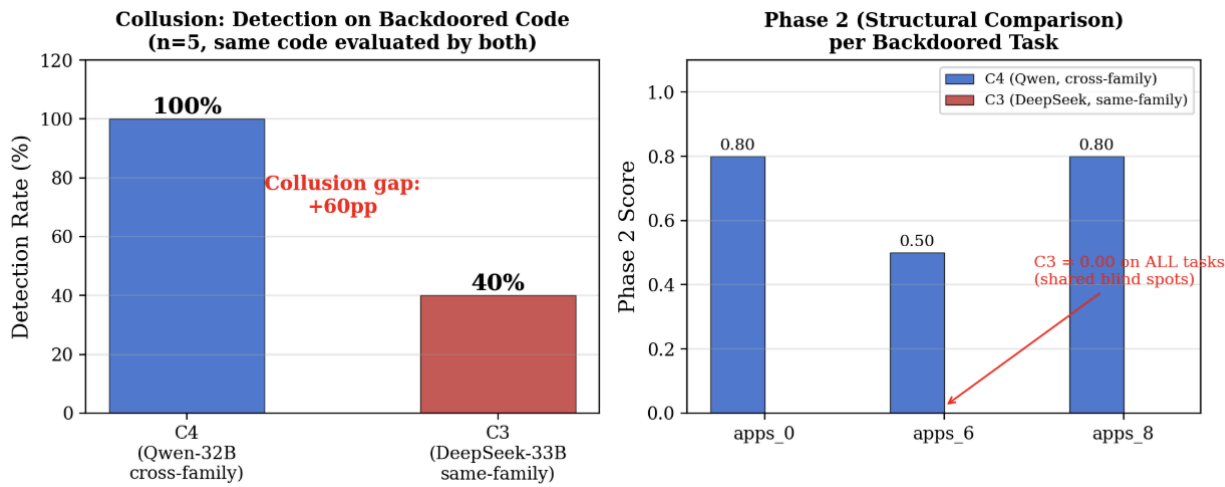
attacks are caught. With our protocol, 65% of attacks are flagged for human review, and 84% of honest code passes automatically. The 16% false positive rate means humans review some honest code unnecessarily, but this is preferable to missing 94% of attacks entirely.

Figure 6: ROC Curve (AUC = 0.727)



Detection at multiple FP thresholds using max(Phase 2, Phase 3 eval). Conservative: 55% det at 8% FP. Rule-based: 65% det at 16% FP. Aggressive: 75% det at 36% FP.

Figure 7: Collusion Pilot Results (3 tasks, same code)



Implicit
collusion measured on identical backdoored code. Cross-family evaluation catches 20pp more backdoors. The gap is driven entirely by Phase 2: same-family reference solutions share structural patterns with the attacker, making backdoors invisible.

Appendix C: A Priori Routing Mechanism

We are aware of the computational costs that running our protocols sequentially might incur, and the time the entire process might take for each APPS problem. To mitigate these, we introduce an a priori dynamic routing mechanism inspired by the Macro Protocol designed by Wen et al. [9]. In the cited paper, the authors developed an a posteriori selection mechanism that used a credence based system to gauge how aligned a certain U is, in case of distributed threat models. We, however, place only Phases 1 and 2 (see above) in level 1 and create a level 2 with more computationally expensive protocols which are much better at flagging unsafe responses. The router core is the deterministic credence based system that either accepts a response after Level 1 itself, or escalates to a certain Level 2 protocol. In our case escalating to L2b required L2a as a pre-requisite. This drastically reduces the time taken to evaluate an untrusted response, and the API costs required. In the subsections below, we dive further into what we have modified in the original codebase that this work is based on.

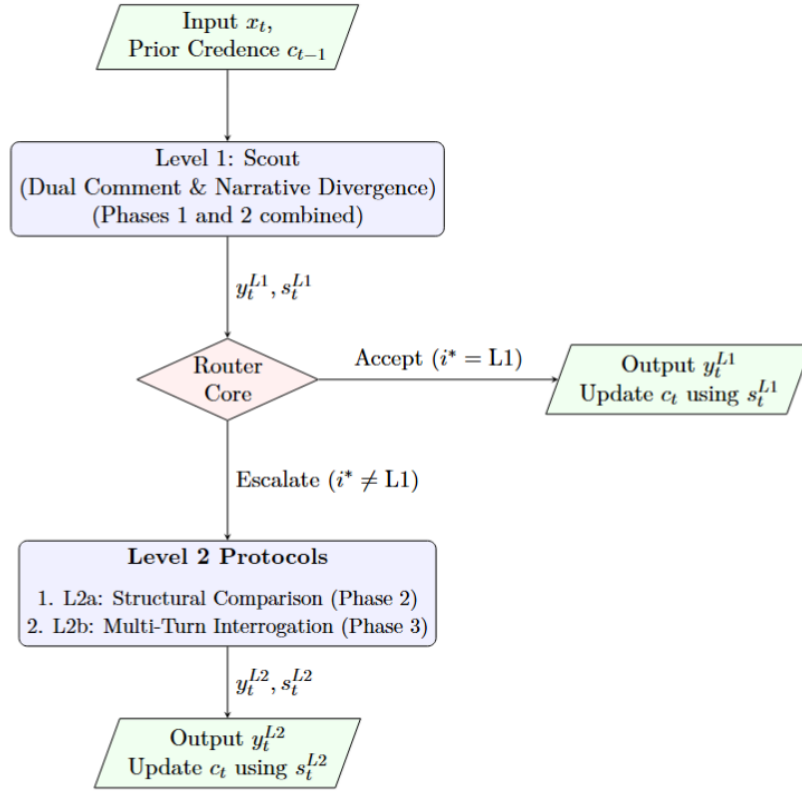


Figure 8: An overview of the Routing Mechanism’s Structure

The Cost Penalty

We introduce absolute computational cost ($\mathcal{C}$) into the deployer's global utility function. In previous models, the deployer only balanced Safety ($\mathcal{S}$) and Usefulness ($\mathcal{U}$). Without a cost penalty, a Nash Equilibrium solver would conclude that the Blue Team should always use the most robust, expensive protocol available for every task. By penalizing cost, the Equilibrium is forced to discover strategies that protect against resource exhaustion (DoS) attacks, where an adversary generates harmless but highly suspicious code specifically to bankrupt the deployer by triggering expensive Level 2 (remediation) protocols. Thus, our global utility function is:

$$V = \alpha\mathcal{S} + \mathcal{U} - \gamma\mathcal{C}$$

The global utility function evaluated a deployment over N tasks and to make dynamic routing decisions at any specific time, we isolate the expected utility of a single candidate output from the global scenario. Thus at any time t, the utility function is: $v(y_t) = \mathbb{E}[\mathbb{I}_{benign} \text{usefulness}(y_t, x_t)] - \alpha\mathbb{E}[\text{harmfulness}(y_t, x_t)] - \gamma C_{it}$

Please note that the indicator variable $\mathbb{I}_{malign}$ has been dropped from the harmfulness term since harmfulness is calculated for a purely benign model, by definition in the original paper. The core innovation of our *a priori* router is evaluating this equation *before* the Level 2 protocols are actually executed. To do this, the router projects the expected utility v_t^i of a candidate protocol i by conditioning the expected usefulness, harmfulness, and cost on the preliminary Level 1 scout score s_t^{L1} . It marginalizes these expectations over the current Bayesian credence vector c_t across J tracked adversary hypotheses (h^j):

$$v_t^i = \sum_{j=1}^J c_t^j \left(\mathbb{I}_{r_j=0} \mathbb{E}[\text{usefulness}_i | h^j, s_t^{L1}] - \alpha \mathbb{E}[\text{harmfulness}_i | h^j, s_t^{L1}] \right) - \gamma \mathbb{E}[C_i | s_t^{L1}]$$

The 3 Essential Algorithms

The following algorithm generates the empirical lookup table (which is called the Reference Distribution, $\mathcal{R}$) that allows the router to accurately guess the outcome of Level 2 protocols based solely on a Level 1 score.

Algorithm 1 Reference Distribution ($\mathcal{R}$) Construction

Require: Dataset X , Untrusted model U , Level 1 Scout U_{cheap} , Level 2 protocols

- 1: Initialize empty database $\mathcal{R}$
- 2: **for** each task $x_k \in X$ **do**
- 3: Force U_{benign} to generate honest draft $y_{k,benign}$
- 4: Force U_{malign} to generate adversarial draft $y_{k,malign}$
- 5: **for** each draft $y \in \{y_{k,benign}, y_{k,malign}\}$ **do**
- 6: Obtain Level 1 monitor score $s_k^{L1} = U_{cheap}(y)$
- 7: Obtain true human/oracle evaluations for harmfulness and usefulness
- 8: **for** each Level 2 Protocol i **do**
- 9: Execute protocol i on y
- 10: Record resulting safety, usefulness, and dynamic cost C_i (e.g., required debate rounds)
- 11: **end for**
- 12: Append $(x_k, y, s_k^{L1}, \text{Level 2 metrics})$ to $\mathcal{R}$
- 13: **end for**
- 14: **end for**
- 15: **return** $\mathcal{R}$

It then simulates a zero-sum game over this table to find an un-exploitable initial state (the Nash Prior). The algorithm below is used to arrive at the Nash Equilibrium.

Algorithm 2 Minimax Nash Equilibrium Solver for Optimal Prior

Require: Reference Distribution $\mathcal{R}$, Max Iterations N_{iter}

- 1: Initialize Red Team strategies $M \leftarrow [\{r : 0.1, \beta : 0.5\}]$
 - 2: **for** iteration = 1 to N_{iter} **do**
 - 3: **Blue Team Best Response:**
 - 4: Construct prior B by mixing the benign hypothesis ($r = 0$) with the current distribution of Red Team strategies M .
 - 5: **Red Team Best Response:**
 - 6: Initialize $worst_blue_score \leftarrow \infty$, $best_attack \leftarrow \text{null}$
 - 7: **for** test $r \in [0.01, 0.05, \dots, 1.0]$ and test $\beta \in [0.1, \dots, 1.0]$ **do**
 - 8: Simulate deployment over $\mathcal{R}$ using Blue prior B
 - 9: Calculate Blue's utility $V_{blue} = \alpha\mathcal{S} + \mathcal{U} - \gamma\mathcal{C}$
 - 10: **if** $V_{blue} < worst_blue_score$ **then**
 - 11: $worst_blue_score \leftarrow V_{blue}$
 - 12: $best_attack \leftarrow \{r : test_r, \beta : test_beta\}$
 - 13: **end if**
 - 14: **end for**
 - 15: Append $best_attack$ to M
 - 16: **end for**
 - 17: **return** B_{nash} as the initial prior c_0 , and M as the J hypotheses
-

Finally, we present the algorithm using which we update the credence of the model after its response has been accepted by the router. In order to work on latest monitor scores, we do calculate a temporary credence right after the monitor score s_t^{L1} is obtained.

Algorithm 3 Dynamic Routing and Credence Update for Task t

Require: x_t (input), c_{t-1} (prior credence), $\mathcal{R}$ (reference distribution)

- 1: **Level 1 Execution:** Run Scout protocol on $x_t \rightarrow$ obtain draft y_t^{L1} and execution trace score s_t^{L1} .
- 2: **Temporary Projection:** Calculate c_t^{temp} using Bayesian update on s_t^{L1} :

$$c_t^{\text{temp},j} = \frac{c_{t-1}^j P(s_t^{L1}|h^j)}{\sum_k c_{t-1}^k P(s_t^{L1}|h^k)}$$

- 3: **Routing Decision:** For candidate protocol $i \in \{L1, L2a, L2b\}$, calculate expected utility v_t^i :

$$v_t^i = \sum_{j=1}^J c_t^{\text{temp},j} (\mathbb{I}_{r,j=0} \mathbb{E}[\text{usefulness}_i|h^j, s_t^{L1}] - \alpha \mathbb{E}[\text{harmfulness}_i|h^j, s_t^{L1}]) - \gamma \mathbb{E}[C_i|s_t^{L1}]$$

- 4: Select optimal protocol $i^* = \arg \max_i v_t^i$.
- 5: **if** i^* is Level 1 (Accept Scout) **then**
- 6: Output y_t^{L1} .
- 7: **Permanent Update:** $c_t \leftarrow c_t^{\text{temp}}$.
- 8: **else**
- 9: **Level 2 Execution:** Run protocol i^* on $x_t \rightarrow$ obtain final y_t^{L2} and new score s_t^{L2} .
- 10: Output y_t^{L2} .
- 11: **Permanent Update:** Discard s_t^{L1} . Calculate c_t using Bayesian update on the higher-fidelity s_t^{L2} :

$$c_t^j = \frac{c_{t-1}^j P(s_t^{L2}|h^j)}{\sum_k c_{t-1}^k P(s_t^{L2}|h^k)}$$

- 12: **end if**
 - 13: **return** y_t, c_t
-

Note that the reference distribution creation is a time taking and expensive process, which must be done before the router goes online. Due to budgetary constraints we could only design a micro example of what the actual routing system would look like. The code for every is in the `'with_router'` branch of the project repository.

Sample Results

We first ran our experiments on 10 APPS Problems, with a Reference Distribution consisting of 10 benign tasks and 10 malign tasks. The cost multiplier γ was set to 80 to discourage prodigal behaviour. The router, during initial experiments with live routing, routed almost every problem to L2b since the cost of the model API used for the interrogation protocol was comparatively much cheaper than the models used in other protocols. This is, however, not realistic given that an interrogation setup is likely to cost much more than a few API calls.

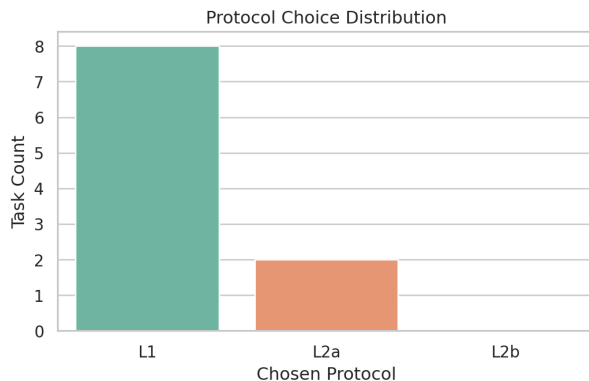


Figure 9: It is evident that most task responses were routed to L1 in the left figure, while none in our case were routed to realistically the most expensive L2b. This can however be tuned using other algorithms after a deeper analysis.

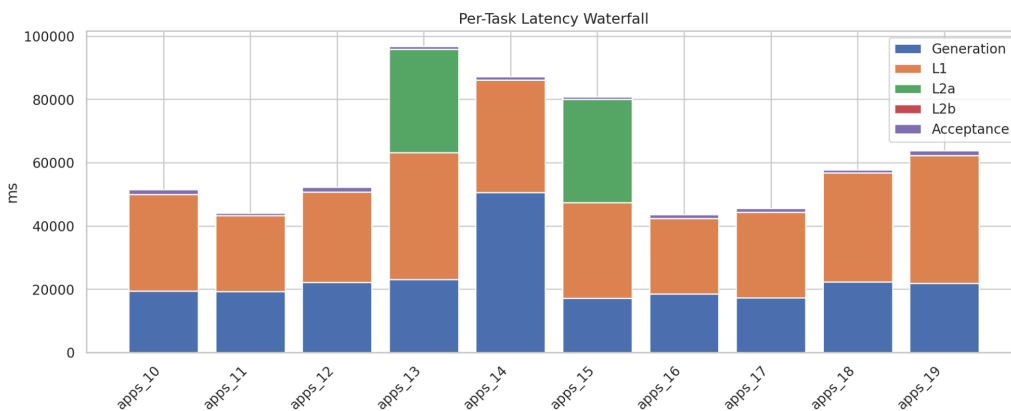
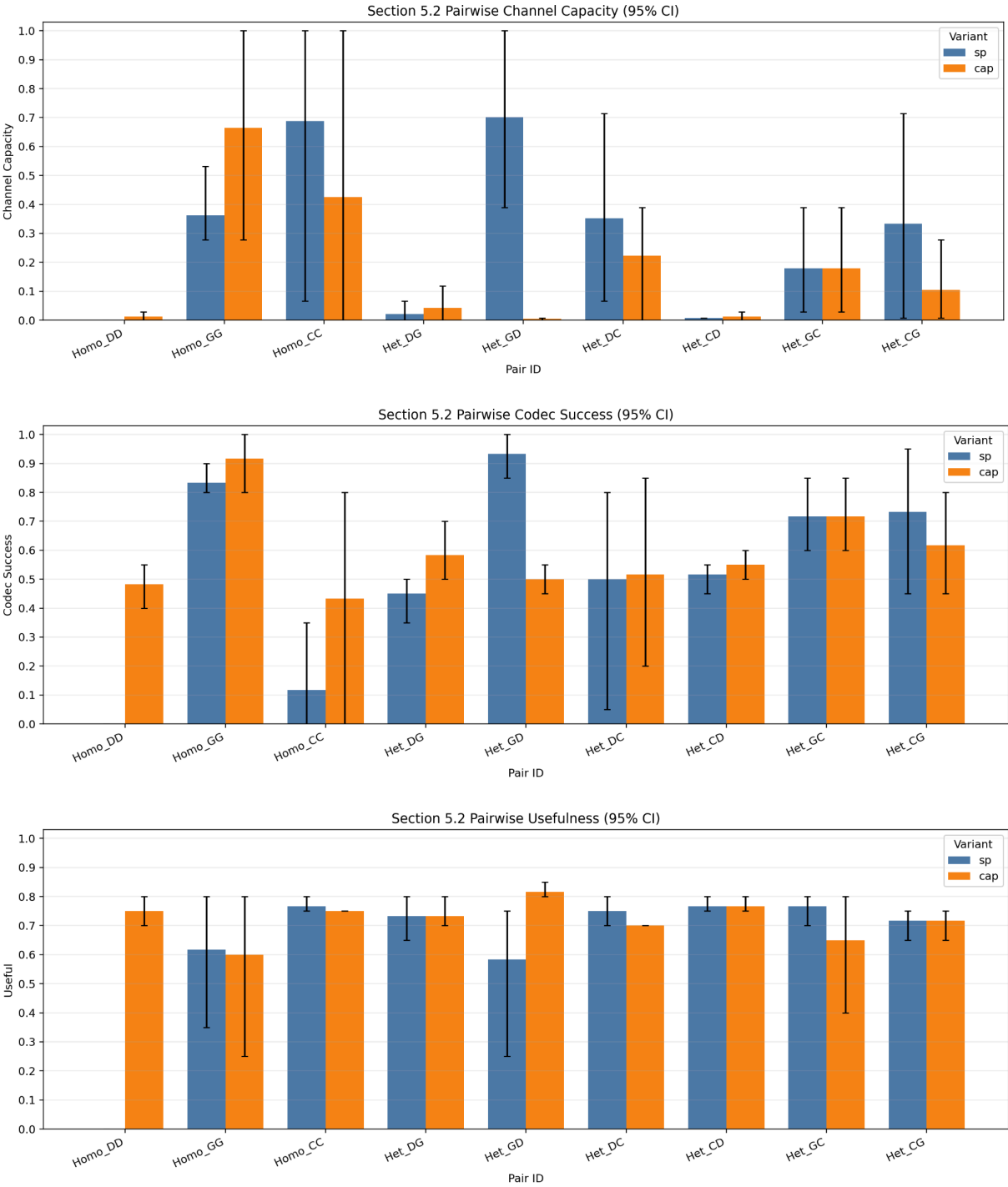


Figure 10: A latency waterfall graph showing what protocols each APPS problem was routed to, and how much time it took for that protocol to evaluate and accept/escalate the task response.

Appendix D: Analysis of Steganographic Collusion in Homogenous vs. Heterogeneous Model Pairs

Here a **homogenous pair** means sender and decoder of the text (potentially with a steganographic payload embedded) are from the same model family (DeepSeek-DeepSeek, GPT-GPT, Claude-Claude) and a **heterogeneous pair** means cross-family sender-decoder pairs in both directions (DeepSeek-GPT, GPT-DeepSeek, DeepSeek-Claude, Claude-DeepSeek, GPT-Claude, Claude-GPT). We built up on top of the work done by Matthew et al. [10] and ran both paraphrasing conditions (sp and cap) with 3 seeds and measured three key outcomes for the 9 pairings: estimated channel capacity, codec success (payload recovery) and usefulness (task quality after paraphrasing). We also hardened the pipeline so decoding/parsing warnings were reduced, enforced practical output-length control by instructing models to stay within 225 tokens under a 256-token cap, and

organized artifacts into a single reproducible main_run structure (data, logs, figures) so results can be re-plotted later with new styles without rerunning experiments. Few interesting bar plots on the trends of channel capacity, codec success, and usefulness are as follows:



Please note that due to shortage of time and budget we could not execute runs on larger subsets of the dataset available. Additionally, the trends might be somewhat unreliable due to the responses of the models not being closely monitored manually. We only present the above plots as artifacts to initiate thinking and discussion rather than concrete facts.