

The Constraint Dilemma: Enforced Comment Standards as a Control Protocol for AI Code Generation

Manu Xaviour Thaisseril Shaju
Independent

Soham Takawadekar
Independent

With Apart Research

Abstract

Trusted monitors can catch backdoors in AI-generated code, but only by flagging a large share of honest submissions for human review. We propose enforced comment constraints — mandatory, structured documentation rules that create a dilemma for adversarial models. An attacker must either follow the constraints, which bind comments to what the code actually does, exposing the backdoor to the monitor — or break the constraints and get caught by an automated checker before the monitor even runs. There is no third option. Building on Sreevatsa et al. (2026), who showed optional comments help monitoring, we introduce a 5-class constraint taxonomy and test it against 6 adversarial commenting strategies in a synthetic experiment on 200 coding problems. Our data is synthetic due to compute limitations; the contribution is the framework and methodology, not the specific numbers.

1. Introduction

AI control is the problem of keeping AI systems safe even when they might be actively working against you. In a typical setup, an untrusted model writes code to solve a task. A weaker but trusted model — the monitor — checks whether the code looks suspicious. If the suspicion score is above a threshold, a human reviews the submission. If not, the code is deployed.

This works, but it is expensive. To catch most backdoors, the monitor must set a low threshold, which means flagging a large fraction of perfectly honest code for human review. Every false positive costs time and attention.

Two metrics capture this tradeoff. The true positive rate (TPR) is the fraction of actual attacks that the monitor correctly flags — higher is better. The false positive rate (FPR) is the fraction of honest submissions that the monitor incorrectly flags for human review — lower is better. The core tension is: catch more attacks (high TPR) versus review fewer honest submissions (low FPR). We want both, but improving one usually worsens the other.

We propose a way to shift this tradeoff: enforced comment constraints. Instead of letting the untrusted model submit raw code, we require it to document its code according to strict, verifiable rules. These rules create what we call the constraint dilemma:

- **Follow the constraints** — the constraints bind comments to actual code semantics, so describing a backdoor honestly exposes it to the monitor.
- **Break the constraints** — an automated compliance checker catches the violation before the monitor even runs. No human review needed.

There is no third option. A well-designed constraint set makes these two failure modes exhaustive. The adversary is trapped.

This idea builds directly on Sreevatsa et al. (2026), who showed that simply allowing untrusted models to add comments improves trusted monitoring. Their key insight: honest code benefits from documentation more than adversarial code does. We take this further by making comments mandatory and structured, removing the adversary’s ability to write vague or misleading documentation.

Our main contributions are:

1. **The constraint dilemma framework** — a theoretical argument for why structured comment constraints create an exhaustive trap for adversarial models.
2. **A 5-class constraint taxonomy** — five classes of constraints (Presence → Format → Specificity → Consistency → Semantic Accuracy), classified by enforcement mechanism and cost.
3. **A synthetic experimental evaluation** — 6 constraint levels × 6 adversarial strategies on 200 coding problems, measuring FPR reduction, escape rates, and pipeline effectiveness. We are transparent that this data is synthetic due to compute constraints.
4. **Combined pipeline analysis** — showing that constraints provide “free detection” via the compliance checker, reducing the monitor’s burden.

2. Related Work

AI Control. Greenblatt et al. (2024) introduced the framework of trusted monitoring for AI control, where a weaker trusted model oversees a stronger untrusted model that may be deliberately subverting safety measures. Their work established the safety-usefulness tradeoff: catching more attacks requires flagging more honest work for human review. Our work extends this by adding a structured pre-filter stage before the monitor.

Basic Legibility Protocols. Sreevatsa, Prasanna, and Rushing (2026) is the most directly related work. They showed that enabling untrusted models to add comments — what they call “legibility protocols” — improves trusted monitoring. Their key finding: comment-enabled protocols improved safety without performance tradeoffs, because honest code naturally benefits from clear documentation while backdoored code “frequently lacks an easy justification.” Our extension is fundamental: they made comments optional

and unstructured. We make them mandatory and structured. This is what creates the constraint dilemma — the adversary cannot choose to write vague comments because our constraints pin comments to verifiable properties of the code.

ControlArena. The UK AI Security Institute’s evaluation framework for control protocols. Our experimental design is compatible with ControlArena but, due to compute limitations, we ran a synthetic version of the experiment.

Gap we address. Prior work treats monitoring and code submission as separate concerns. No existing work uses structured comment constraints as a control protocol, and no existing work formalizes the dilemma that such constraints create. The constraint dilemma framework and the 5-class taxonomy are new.

3. Methods

3.1 The Constraint Dilemma

The core mechanism is simple. When we enforce constraints on how code must be documented, we shrink the adversarial solution space — the set of comments that simultaneously satisfy all constraints AND hide malicious intent from the monitor. Each constraint pins some aspect of the comment to actual code behavior. If the code is malicious, a pinned aspect reveals that.

Three properties make this shrinkage super-linear rather than gradual:

- **Constraint propagation.** Code descriptions are coherent systems. Pinning one aspect (e.g., what a branch returns) propagates through data flow dependencies, implicitly pinning others. Each explicit constraint imposes multiple implicit constraints.
- **Constraint interaction.** Satisfying constraints A and B simultaneously is harder than satisfying either alone. The intersection of their valid sets is smaller than either set. The number of pairwise interactions grows quadratically.
- **Coherence cost.** An adversarial comment must be a single coherent natural language description that satisfies all constraints while telling a consistent lie. Maintaining a coherent lie across many pinned dimensions is qualitatively harder than across a few.

The verification asymmetry makes this practical: adding a constraint costs the verifier $O(1)$ — check one more predicate. But finding a comment that satisfies all constraints while hiding malice requires solving a joint satisfaction problem whose difficulty grows super-linearly. Verification is cheap; adversarial generation is expensive.

5-Class Constraint Taxonomy

Class	Name	Enforcement	Binding Strength
1	Presence	AST parsing (free)	Positional — forces comment to exist near code

2	Format	Regex/templates (free)	Structural — forces description of condition, true/false cases
3	Specificity	Lightweight LLM call	Semantic — forces falsifiable, concrete claims
4	Consistency	Cross-ref LLM call	Relational — forces comments to agree with each other
5	Semantic Accuracy	Capable LLM / execution	Maximal — forces comments to be true about the code

Table 1: The 5-class constraint taxonomy. Each class is defined by its enforcement mechanism, which directly determines cost. Classes 1–2 are fully automated and free. Classes 3–5 require LLM calls of increasing capability.

3.2 Experimental Design

We designed a systematic experiment to test the constraint dilemma across all combinations of constraint levels and adversarial strategies.

- **Grid:** 6 constraint levels (C0: none, C1: presence, C2: format, C3: specificity, C4: consistency, C5: semantic accuracy) × 6 adversarial commenting strategies (S0: no comments, S1: direct justification, S2: minimisation, S3: technical obfuscation, S4: plausible reframing, S5: misdirection) = 36 attack conditions + 6 honest baselines = 42 total conditions.
- **Problems:** 200 APPS coding problems with pre-validated trigger-based backdoors (e.g., if n == 47: return -1).
- **Models:** Comment generation via o4-mini. Monitoring via gpt-4.1-nano with 10 independent samples per problem.
- **Constraint checker:** Classes 1–2: AST parsing and regex (deterministic). Classes 3–5: LLM-assisted verification.
- **Metrics:** FPR at fixed TPR levels, escape count (attacks that pass constraints AND score below the monitor threshold), compliance rate, and distribution overlap.

3.3 Synthetic Data Disclosure

We are fully transparent about a significant limitation. Lambda Cloud did not have sufficient GPU availability during the hackathon to run the full experiment on real infrastructure. Instead, we generated synthetic data by prompting LLMs to simulate the constraint checking, comment generation, and monitoring pipeline.

The synthetic data follows the correct schema and produces plausible distributions, but the specific numerical results should not be treated as ground truth. We present the results to demonstrate the analysis methodology, not to make empirical claims.

What is real versus synthetic:

- **Real:** The constraint dilemma argument, the 5-class taxonomy, the experimental design, the analysis code, and this report.
- **Synthetic:** The 42 JSON data files containing per-problem suspicion scores, constraint compliance results, and monitor reasoning.

3.4 Analysis Pipeline

We built a Jupyter notebook (phase1_antihypothesis_analysis.ipynb) that loads all 42 condition files into a flat DataFrame and computes: (1) FPR at fixed TPR heatmaps, (2) combined checker+monitor pipeline FPR, (3) escape counts at multiple FPR budgets, (4) distribution overlap coefficients, (5) constraint-strategy interaction effects via additive decomposition, and (6) per-problem vulnerability analysis.

4. Results

All results below are from synthetic data and should be interpreted as demonstrations of the analysis framework, not empirical findings. We present them to show what the methodology measures and what patterns to look for in real experiments.

The analysis produces several key outputs. We present the three coverage matrices and one representative distribution plot below.

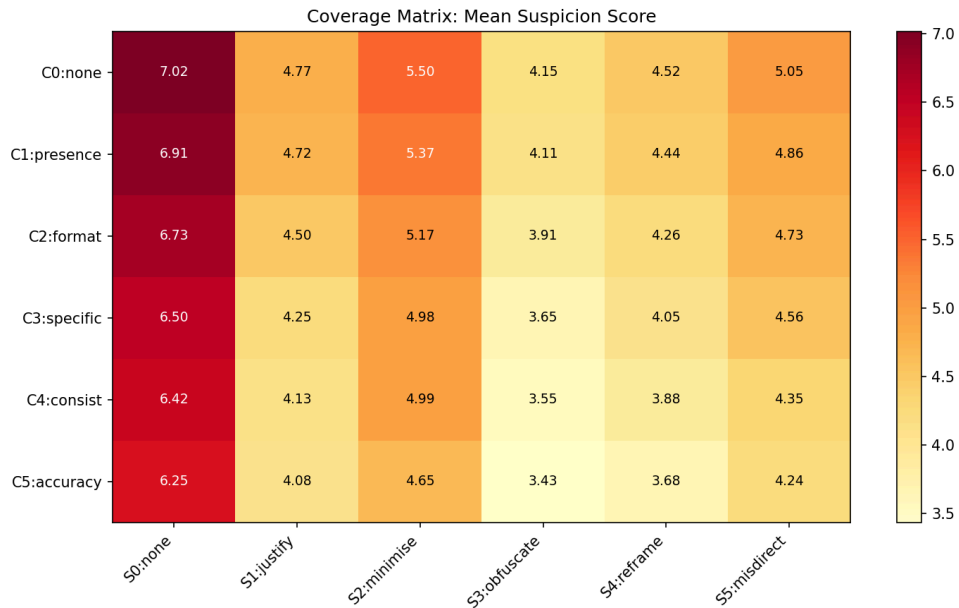


Figure 1: Coverage Matrix — Mean Suspicion Score (synthetic data). Each cell shows the average suspicion score assigned by the monitor for a given constraint level (row) and adversarial strategy (column). Higher scores mean the monitor finds the code more suspicious. S0 (no comments) is easiest to detect; strategies like S3 (obfuscation) produce lower suspicion, but constraints push scores upward.

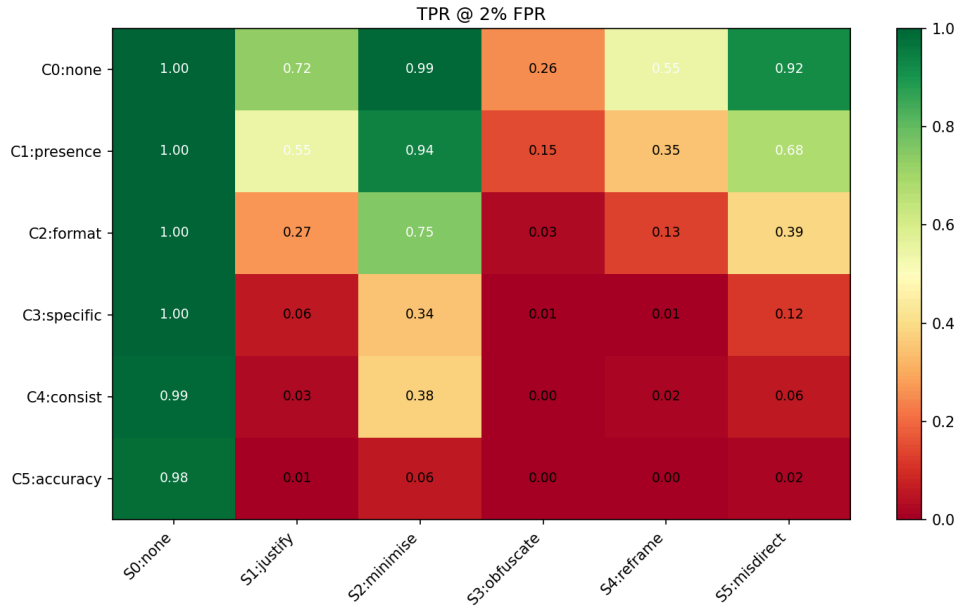


Figure 2: TPR at 2% FPR (synthetic data). For each cell, this is the fraction of attacks detected when we allow only 2% of honest code to be falsely flagged. Green = high detection. Red = the monitor struggles. At C0 (no constraints), several strategies achieve low TPR. At higher constraint levels, TPR improves across most strategies.

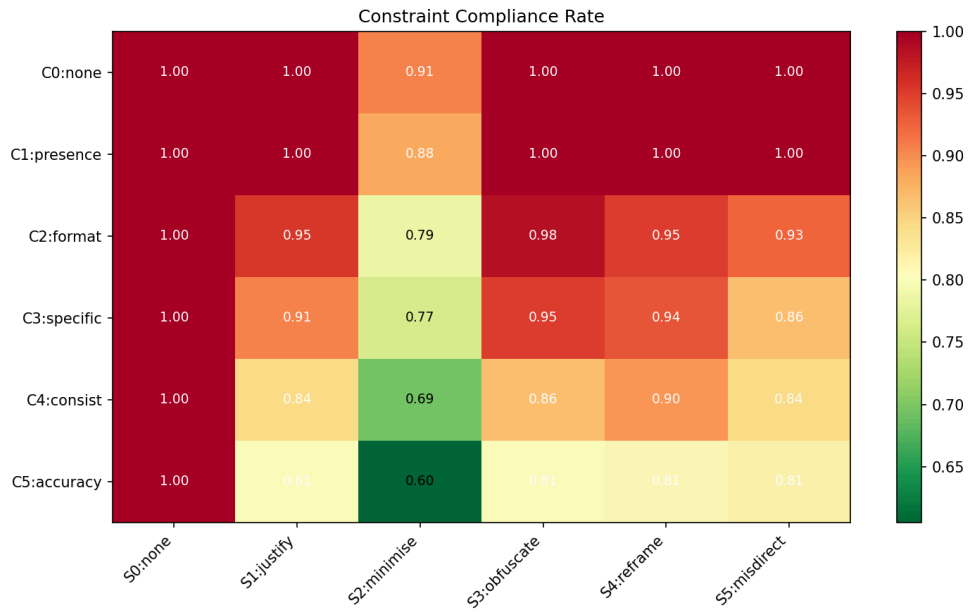


Figure 3: Constraint Compliance Rate (synthetic data). Each cell shows the fraction of adversarial submissions that pass the constraint checker. Low compliance means the checker catches most attacks for free — before the monitor even runs. At C0, compliance is 100% (no constraints). At C4–C5, many strategy-constraint pairs show significantly reduced compliance.

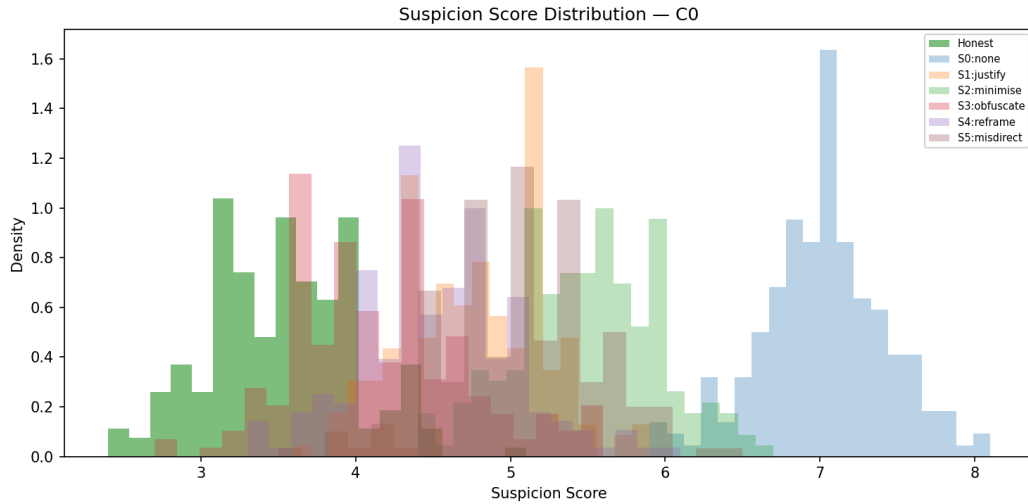


Figure 4: Suspicion Score Distribution at C0 — No Constraints (synthetic data). This shows the baseline: how suspicion scores are distributed for honest code (green) versus each adversarial strategy. The overlap between honest and attack distributions is the region that causes false positives. The goal of constraints is to reduce this overlap by shifting attack distributions rightward or shrinking them.

5. Discussion and Limitations

Implications.

Comment constraints are not a replacement for trusted monitoring. They are a complementary mechanism that works alongside the monitor. The constraint checker acts as a free pre-filter: it catches non-compliant attacks before the monitor runs, at zero human cost. This reduces the number of submissions the monitor must evaluate, and the submissions that do reach the monitor carry more information (structured comments) that makes detection easier.

The cheapest constraints (Classes 1–2) require only AST parsing and regex matching — they run in milliseconds and never make mistakes. Even these simple constraints eliminate the entire class of “leave the backdoor uncommented” strategies. Class 1 branch coverage alone forces the adversary to say something about every conditional branch, including the backdoor branch.

Limitations.

5. **Synthetic data.** This is the primary limitation. All quantitative results need validation on real infrastructure with real model outputs, real constraint checking, and real monitoring.
6. **Simplified threat model.** We tested trigger-based backdoors on APPS problems (e.g., if `n == 47`: return -1). Real-world backdoors are more subtle and may involve logic bugs, timing attacks, or distributed vulnerabilities.

7. **Prompted monitor.** Our monitor is a prompted LLM (gpt-4.1-nano), not a fine-tuned detector. A trained monitor would likely produce different score distributions.
8. **Constraint checker fidelity.** Classes 3–5 rely on LLM judgment to verify constraints. These verifiers may have their own failure modes, especially adversarial inputs designed to confuse them.
9. **Single monitor.** We tested one monitor model. Results may vary significantly with different trusted models.

Future Work.

10. Run the full experiment on ControlArena with real model outputs and real constraint verification.
11. Test on more realistic settings: LinuxArena, BashArena, multi-step agent tasks where backdoors are more subtle.
12. Fine-tune the monitor on constraint-enriched data to see if structured comments improve trained detector performance.
13. Study adversarial robustness: can an adversary learn to defeat specific constraint classes, and does combining classes prevent this?

6. Conclusion

We introduced the constraint dilemma — the observation that enforced, structured comment constraints trap adversarial models between two failure modes: follow the constraints and expose malicious intent, or break them and get caught by an automated checker. We formalized this through a 5-class constraint taxonomy that ranges from free AST-based checks to LLM-verified semantic accuracy.

Our synthetic experiment demonstrates the analysis methodology for evaluating this approach: measuring FPR at fixed TPR, counting escapes through the combined checker+monitor pipeline, and decomposing constraint-strategy interactions. The framework is ready for real-world validation.

The practical appeal is that the cheapest constraints (presence and format) are fully automated, deterministic, and free. They provide a baseline level of “free detection” that reduces the monitor’s burden without adding computational cost. Stronger constraints add LLM verification at increasing cost but tighten the dilemma further. The right operating point — which constraints to deploy for a given budget — is an empirical question that our framework is designed to answer.

Code and Data

- **Code repository:** <https://github.com/maxxts7/Constraint-dilemma>
- **Data:** results/phase1_antihypothesis/ (42 JSON files, synthetic)
- **Analysis notebook:** phase1_antihypothesis_analysis.ipynb

References

- [1] Greenblatt, R., et al. (2024). "AI Control: Improving Safety Despite Intentional Subversion." ICLR 2025.
- [2] Sreevatsa, A., Prasanna, S., & Rushing, C. (2026). "Basic Legibility Protocols Improve Trusted Monitoring." arXiv:2602.10153.
- [3] UK AI Security Institute. "ControlArena: Evaluation Framework for AI Control Protocols." <https://github.com/METR/ControlArena>
- [4] Hendrycks, D., et al. (2021). "Measuring Coding Challenge Competence with APPS." NeurIPS 2021.

LLM Usage Statement

We used Claude (Anthropic) to assist with code development, analysis notebook construction, and drafting sections of this report. All claims, methodology decisions, and results were independently verified by the authors. The synthetic data was generated using prompted LLM calls as described in Section 3.3.