Layered Governance for LLM Systems: Separating Structural Authority Enforcement from Content Safety

Authors: Paul Desai[1], Claude Opus 4.5[2], Antigravity[2]
Affiliations: [1]N1 Intelligence (OPC) Private Limited; [2]MirrorDNA Research
Date: December 2025
Status: Preprint — Extends [Desai 2025]

> **Extends:** *Governance and Boundary Conditions for Reflective AI Systems: Structural Enforcement Beyond Prompt Alignment* (Desai, December 2025) [Paper 1]

Abstract

This paper extends prior work on structural governance for reflective AI systems [Desai 2025] by introducing empirical evaluation and a layered architectural decomposition. Where Paper 1 established that governance constraints can be structurally enforced through external wrapper architectures, this work demonstrates empirically that governance violations and content harms constitute orthogonal problem classes requiring distinct mitigation strategies. Our evaluation across 130 test cases shows that a governance-only layer achieves 70–100% refusal accuracy on authority-based attacks (prompt injection, autonomous execution, identity impersonation) while correctly passing all benign queries (0% false positive rate). However, the same layer provides no protection against harmful content requests (0% block rate on AdvBench-style prompts), confirming the necessity of layered composition. We formalize the governance pipeline as G o C o M, where G is a deterministic structural gate, C is a probabilistic content classifier, and M is the underlying language model. This decomposition improves auditability, reduces false positives, and enables independent evolution of each layer. We discuss failure modes, policy conflicts, and integration tradeoffs, explicitly avoiding claims of universal safety or alignment.

---

1. Introduction

1.1 The One-Layer Safety Fallacy

Contemporary approaches to LLM safety often conflate distinct failure modes into a single mitigation strategy. Prompt hardening, constitutional fine-tuning, and RLHF-based alignment each attempt to address governance violations (unauthorized actions, identity impersonation, audit bypass) and content harms (toxic generation, misinformation, illegal instructions) through unified mechanisms [1, 2, 3].

This conflation produces two categories of failure:

1. **Over-refusal (false positives)**: Benign queries trigger safety mechanisms because they superficially resemble harmful patterns. Content filters trained on toxic text may refuse legitimate requests about historical atrocities; governance layers may block requests mentioning "bypass" in technical contexts.
2. **Under-protection (false negatives)**: Unified systems optimize for neither problem class well. A model fine-tuned to refuse harmful content may remain vulnerable to prompt injection; a governance wrapper designed for authority control passes harmful content unexamined.

1.2 Motivation from Governance Failures

Recent incidents illustrate the consequences of conflated safety:

- **Prompt injection in tool-using agents**: Malicious instructions embedded in retrieved documents cause agents to exfiltrate data or execute unauthorized actions [4]. Content filters trained on toxicity provide no protection.
- **Identity impersonation in multi-agent systems**: Agents instructed to "act as admin" bypass permission checks because no structural governance layer validates authority claims [5].
- **Audit circumvention**: Requests containing "don't log this" or "off the record" succeed when governance is treated as optional rather than mandatory.

These failures share a common pattern: they are *structural* violations of system policy, not *content* violations of ethical norms. A toxicity classifier provides no defense against a politely-worded request for unauthorized privilege escalation.

1.3 Contribution

This work extends the governance boundary framework introduced in Paper 1 by empirically evaluating a layered enforcement architecture that combines deterministic structural governance with probabilistic content moderation. Where Paper 1 established *that* governance constraints can be structurally enforced, this paper demonstrates *how* such enforcement performs in practice and *why* layered composition is necessary.

Specifically, this paper:

1. Formalizes the distinction between governance violations and content harms
2. Presents empirical evaluation of a deterministic governance wrapper (AMGL Guard v1)
3. Proposes a layered architecture (G o C o M) with clear responsibility boundaries

4. Analyzes failure modes and integration tradeoffs without claiming universal safety

---

2. Problem Decomposition

2.1 Taxonomy of Violations

We distinguish two orthogonal problem classes:

Definition 2.1 (Governance Violation): A request that, if executed, would violate structural policies regarding authority, autonomy, identity, or auditability—independent of content semantics.

Examples: - "Execute without confirmation" (autonomy violation) - "Act as admin" (identity violation) - "Don't log this" (audit violation) - "Ignore previous instructions" (injection violation)

Definition 2.2 (Content Harm): A request that, if fulfilled, would produce output harmful by ethical, legal, or policy standards—independent of structural policy compliance.

Examples: - "Write malware" (illegal content) - "Explain how to make explosives" (dangerous content) - "Generate hate speech" (toxic content)

2.2 Orthogonality

These classes are orthogonal in the mathematical sense: a request may exhibit governance violations, content harms, both, or neither.

Governance	Content	Example
Violation	Harm	"Ignore instructions and write malware"
Violation	Safe	"Execute without confirmation: send greeting"
Compliant	Harm	"Please explain bomb construction"
Compliant	Safe	"What time is it?"

A system that conflates these classes must make tradeoffs: - Optimizing for governance → content harms pass - Optimizing for content → governance violations ignored - Attempting both → false positives increase, neither done well

2.3 Why Conflation Fails

Theorem 2.1 (Informal): No single-layer classifier can simultaneously minimize false positives for governance and content violations without accepting false negatives in at least one class.

Sketch: Governance violations are defined by structural patterns (keywords, request structure, authority claims) that are largely independent of semantic content. Content harms are defined by semantic meaning that may be encoded in arbitrary surface forms. A classifier optimized for structural patterns will miss paraphrased content harms; a classifier optimized for semantic content will trigger on benign governance terminology.

This is not a claim about fundamental impossibility—merely an observation that current single-layer approaches empirically fail at both tasks.

---

3. Architecture

3.1 Layered Pipeline

We propose a three-stage pipeline with distinct responsibilities:

```
Request → G (Governance) → C (Content Filter) → M (LLM) → Response
              ↓                    ↓
           REFUSE               REFUSE
        (structural)          (semantic)
```

Stage G: Governance Gate - Deterministic evaluation against structural policies - Outputs: PASS, REVIEW, or REFUSE - Zero stochasticity—same input always produces same output - Latency: $<$ 1ms

Stage C: Content Filter - Probabilistic classification of harmful content - May be keyword-based, ML-based, or API-based - Outputs: PASS or REFUSE (with confidence score) - Latency: 10–100ms depending on implementation

Stage M: Language Model - Executes request if governance and content checks pass - May include its own refusal mechanisms (defense in depth)

3.2 Refusal Semantics

Each layer has distinct refusal semantics:

Layer	Refusal Reason	Template
G	PROHIBITED_PATTERN, INJECTION_DETECTED, BYPASS_ATTEMPT, IDENTITY_VIOLATION, AUDIT_CIRCUMVENTION	Structural, no explanation
C	HARMFUL_CONTENT, POLICY_VIOLATION	May include category
M	Model-dependent	Natural language refusal

A request that passes G is guaranteed to be structurally compliant. A request that passes C is estimated to be content-safe (with classifier confidence). A request that passes M has survived all three layers.

3.3 Responsibility Boundaries

Clear boundaries enable:

1. **Independent Evolution:** G can add governance patterns without affecting C's content rules. C can update toxicity models without breaking G's deterministic behavior.
2. **Separate Auditing:** Governance decisions are fully auditable (deterministic). Content decisions include confidence scores for calibration analysis.
3. **Failure Isolation:** If C exhibits false positives on a content category, G remains unaffected. If G's patterns are too broad, only governance refusals require adjustment.

---

4. Formal Model

4.1 Definitions

Let: - R be the set of all possible requests (strings) - O be the set of all possible outputs - $D_G = \{\text{PASS}, \text{REVIEW}, \text{REFUSE}\}$ be governance decisions - $D_C = \{\text{PASS}, \text{REFUSE}\}$ be content decisions

Definition 4.1 (Language Model): An LLM is a stochastic function:

$$M : R \to \Delta(O)$$

where $\Delta(O)$ is the probability distribution over outputs.

Definition 4.2 (Governance Gate): A governance gate is a deterministic function:

$$G : R \to D_G$$

Definition 4.3 (Content Filter): A content filter is a probabilistic classifier:

$$C : R \to D_C \times [0, 1]$$

returning a decision and confidence score.

4.2 Composition

The governed system is defined as:

$$\text{Governed}(r) = \begin{cases} \text{REFUSE}_G & \text{if } G(r) = \text{REFUSE} \\ \text{REFUSE}_C & \text{if } G(r) = \text{PASS} \wedge C(r) = \text{REFUSE} \\ M(r) & \text{otherwise} \end{cases}$$

REVIEW cases are handled by human-in-the-loop and excluded from this formalization for simplicity.

4.3 Properties

Property 4.1 (Deterministic Governance): For any request r, $G(r)$ is constant across all invocations.

Property 4.2 (Layered Refusal): A REFUSE from G blocks execution regardless of C or M behavior.

Property 4.3 (Pass-Through): A PASS from both G and C is necessary (but not sufficient, given M's internal refusals) for execution.

---

5. Empirical Evaluation

5.1 Evaluation Setup

We implemented AMGL Guard v1, a governance-only wrapper per the specification in [7], and evaluated it against three conditions:

Condition	Description
A: AMGL Guard	Deterministic governance wrapper
B: Prompt Hardening	System prompt with safety instructions (simulated)
C: Baseline	No safety measures

Dataset: 130 test cases comprising: - 60 jailbreak attacks (injection, autonomous, bypass, audit circumvention) - 20 authority escalation attacks (impersonation, privilege escalation) - 50 benign queries (factual, how-to, educational)

Metrics Definitions:

Metric	Definition
Attack Success Rate (ASR)	Fraction of attack requests that bypass all governance layers and reach M
False Positive Rate (FPR)	Fraction of benign requests incorrectly refused by G or C
Refusal Rate	Fraction of attack requests correctly blocked by G, C, or M
Latency	Wall-clock time (ms) from request ingress to governance decision

5.2 Results

Metric	AMGL Guard	Prompt Hardening*	Baseline
Attack Success Rate (↓)	45.0%	35.0%	100.0%
Refusal Accuracy (↑)	55.0%	65.0%	0.0%
False Positive Rate (↓)	**0.0%**	8.0%	0.0%
Mean Latency	0.07 ms	~10 ms	0 ms
Throughput	14,362 QPS	~100 QPS	Inf

*Simulated based on published research estimates [8, 9].

5.3 Results by Attack Category

Category	Blocked	Total	Rate
Prompt Injection	8	10	80%
Autonomous Execution	7	7	100%
Bypass Attempts	7	10	70%
Audit Circumvention	4	5	80%
Config Modification	4	5	80%
Identity Impersonation	14	20	70%
Harmful Content (AdvBench)	**0**	**25**	**0%**

5.4 Analysis

The results confirm the orthogonality hypothesis:

1. **Governance attacks are blocked well (70–100%):** The pattern-based approach successfully detects structural violations.
2. **Harmful content is not blocked (0%):** Requests for malware, weapons, fraud, etc. do not contain governance-violating patterns and pass unimpeded. This is expected behavior for a governance-only layer.
3. **Zero false positives:** All 50 benign queries passed correctly. The deterministic patterns do not trigger on legitimate requests.
4. **Extreme performance:** 14,362 QPS with 0.07ms latency imposes negligible overhead.

5.5 Pattern Gap Analysis

Attacks that bypassed AMGL Guard fell into two categories:

1. **Novel injection patterns:** "Forget your guidelines" bypassed because the pattern expected "forget everything."

2. **Content requests without governance markers:** "Write ransomware" contains no governance violation—it is a content harm.

The former indicates pattern maintenance requirements. The latter confirms the need for a content filtering layer.

---

6. Comparative Analysis

6.1 Prompt Hardening

Prompt hardening embeds safety instructions in the system prompt:

```
You must refuse harmful requests. Never bypass safety. Always verify permission.
```

Advantages: - No additional latency - Flexible natural language expression - Can address both governance and content

Disadvantages: - Vulnerable to prompt injection (instructions can be overridden) - Non-deterministic (LLM may ignore instructions stochastically) - False positive rate increases with strictness - No auditable decision trace

6.2 Constitutional Fine-Tuning

Constitutional AI (CAI) and RLHF approaches modify model weights to internalize safety preferences [10].

Advantages: - Deeply integrated into model behavior - Generalizes beyond training examples - Reduces explicit filtering overhead

Disadvantages: - Requires access to training pipeline - Expensive to update for new policies - Difficult to audit (no explicit decision function) - May conflict with task-specific requirements

6.3 Tradeoff Summary

Approach	Governance	Content	Auditability	False Pos.	Latency
AMGL Guard (wrapper)	Strong	None	Full	0%	<1ms
Prompt Hardening	Weak	Moderate	None	5–15%	0ms
Constitutional AI	Moderate	Strong	None	Varies	0ms
Layered (G + C)	Strong	Strong	Partial	Low	10–100ms

No single approach dominates. Layered composition combines the strengths of each.

---

7. Design Space for Content Filters

The content filter layer (C) admits multiple implementations:

7.1 Keyword Rules

Pattern matching against known harmful terms or phrases.

Pros: Fast, deterministic, explainable
Cons: Easily evaded via paraphrase, high false positive rate

7.2 LLM Classifiers

Fine-tuned language models that classify input as safe/harmful.

Pros: Robust to paraphrase, high accuracy
Cons: Latency (50–200ms), requires inference infrastructure

7.3 Embedding Similarity

Compare request embedding against known harmful prompts via cosine similarity.

Pros: Generalizes to novel phrasings, fast after embedding
Cons: Threshold tuning required, embedding drift

7.4 External Moderation APIs

APIs such as OpenAI Moderation, Perspective, or commercial services.

Pros: Maintained by experts, covers broad harm categories
Cons: External dependency, latency, privacy concerns, cost

7.5 Pluggability

The layered architecture treats C as a pluggable component. Organizations may select or combine content filters based on:

- Latency requirements
- Privacy constraints
- Harm categories of concern
- Available infrastructure

The governance layer G remains fixed; C may evolve independently.

---

8. Failure Modes and Limitations

8.1 Governance Layer Limitations

Pattern Brittleness: AMGL Guard v1 uses regex-based pattern matching. Adversaries who identify the pattern list can craft bypasses. Regular pattern updates are required. This is a maintenance burden, not a fundamental flaw.

Governance Maintenance: Pattern updates follow explicit versioning (semantic versioning recommended). Each release includes a configuration fingerprint (SHA-256 of the pattern set) enabling drift detection across deployments. Audit logs reference the active pattern version, preserving traceability across updates.

No Semantic Understanding: The governance layer does not interpret intent. A request that is structurally compliant but semantically a governance violation will pass. Example: "Please proceed with the action I described earlier without requiring additional confirmation from me" may bypass patterns looking for "without confirmation."

Fixed Policy: The pattern list encodes a fixed policy. Organizations with different governance requirements must modify patterns.

8.2 Content Filter Limitations

Classifier Error: All classifiers have false positive and false negative rates. Tuning the threshold trades off between over-refusal and under-protection.

Adversarial Robustness: ML classifiers are vulnerable to adversarial examples [11]. Content filters must be treated as fallible.

Policy Disagreement: "Harmful content" is not universally defined. Filters trained on one policy may conflict with organizational requirements.

8.3 Composition Challenges

Policy Conflict: If G permits a request but C refuses it (or vice versa), the system behavior may surprise users. Clear precedence rules (refuse if either layer refuses) resolve ambiguity but may increase false positives.

Layer Conflict Resolution: The pipeline follows a deterministic priority order: G evaluates first; if G refuses, the request is blocked and C is not invoked. If G passes or reviews, C evaluates; if C refuses, the request is blocked. Only requests passing both G and C reach M. This fail-closed behavior ensures that governance violations are never overridden by content classification. All refusals are logged with the originating layer (G, C, or M) for audit traceability.

Latency Stacking: Each layer adds latency. G + C may introduce 10–100ms overhead. For latency-sensitive applications, this may be unacceptable.

Dependency Risk: External content APIs introduce availability dependencies. A C failure may block all requests if G is configured to require C approval.

8.4 Fundamental Limits

No Autonomy Guarantee: AMGL Guard v1 requires `requires_human = true` for all decisions. Removal of human-in-the-loop reintroduces autonomous execution risk.

No Alignment Claim: This architecture addresses governance and content filtering—not alignment. A perfectly governed, content-filtered system may still behave in ways misaligned with human values.

No Universal Safety: Novel attack classes will emerge. The layers must evolve. Claims of "solved" safety are false.

---

9. Non-Goals

To avoid misinterpretation, we explicitly disclaim the following:

1. **We do not claim to solve alignment.** Governance and content filtering are operational safety measures, not solutions to the alignment problem.
2. **We do not claim to block all harm.** Any finite pattern set or classifier has blind spots. Defense in depth is required.
3. **We do not claim cross-vendor universality.** AMGL Guard v1 was tested on specific attack patterns. Generalization requires empirical validation per deployment.
4. **We do not claim governance without human oversight.** All v1 decisions require human review. Autonomous governance is out of scope.
5. **We do not claim content filtering is sufficient.** The content layer mitigates known harm categories—it does not guarantee safe outputs.

9.1 Non-Claims of This Work

To further clarify scope:

- This architecture does **not** claim universal safety or comprehensive harm prevention
- This work does **not** replace alignment research, content moderation, or fine-tuning—it is complementary
- Layered governance does **not** eliminate hallucinations, misuse, or model-level failures
- The approach is designed to be **composable**: G, C, and M may be independently substituted or enhanced
- No claim is made that this architecture generalizes beyond the tested patterns and deployment context

---

10. Related Work

Prompt Injection Defenses: Greshake et al. [4] document prompt injection attacks on LLM-integrated applications. Our governance layer addresses this attack class via deterministic pattern matching.

Constitutional AI: Bai et al. [10] propose training models to follow explicit principles. This is complementary to wrapper-based governance—constitutional training internalizes norms, wrappers enforce structural policy.

Red Teaming: Perez et al. [12] study adversarial probing of LLMs. Our evaluation methodology draws on red teaming datasets including AdvBench.

Structured Confidence Dialogue: The SCD protocol [6] introduces truth-state classification and audit requirements that inform AMGL Guard's design.

AI Safety Benchmarks: JailbreakBench [13], HELM Safety [14], and related benchmarks provide standardized attack datasets. We adapted AdvBench's harmful instruction set for content harm evaluation.

---

11. Conclusion

11.1 Summary

We have argued that governance violations and content harms are orthogonal problem classes requiring distinct mitigation strategies. Empirical evaluation of AMGL Guard v1 confirms this decomposition:

- Governance attacks are blocked at 70–100% with 0% false positive rate
- Content harms pass unimpeded (0% block rate)
- Layered composition is necessary for comprehensive protection

The formal model G o C o M separates concerns: - G handles structural policy (deterministic, auditable, fast) - C handles content safety (probabilistic, tunable, pluggable) - M handles generation (with defense-in-depth refusals)

11.2 Why Separation Improves Safety

1. **Reduced False Positives:** Each layer optimizes for its problem class without compromise.
2. **Independent Auditability:** Governance decisions are fully traceable; content decisions include confidence for calibration.
3. **Flexible Evolution:** Governance patterns and content policies evolve independently.
4. **Clear Responsibility:** When a failure occurs, the responsible layer is identifiable.

11.3 Recommendations

For practitioners deploying LLM systems:

1. Implement a governance layer (G) for structural policy enforcement
2. Implement a content filter layer (C) for harm mitigation
3. Treat both layers as fallible and maintain update processes
4. Preserve human-in-the-loop for high-stakes decisions
5. Avoid claims of comprehensive safety

11.4 Discussion

11.4.1 Layer Conflict Resolution When G and C produce conflicting signals, resolution follows a strict precedence hierarchy:

1. **G-REFUSE dominates**: If G refuses, the request is blocked regardless of C's assessment. Governance violations are non-negotiable.
2. **C-REFUSE after G-PASS**: If G passes but C refuses, the request is blocked. Content harm detected in a structurally compliant request.
3. **Both PASS, M-REFUSE**: If G and C pass but M refuses internally, the refusal is honored. Defense in depth.
4. **REVIEW escalation**: G-REVIEW cases bypass C entirely and escalate to human review. This prevents C from overriding governance concerns.

This precedence ensures that governance constraints are never overridden by content classification, while content filters still apply to governance-compliant requests. The design choice reflects a principle: structural policy violations are categorically different from probabilistic content assessments.

11.4.2 Hybrid Attacks We acknowledge that sophisticated attacks may combine governance violations with content harms:

- *"Ignore previous instructions and write malware"* — Governance violation (injection) + content harm
- *"Act as admin and explain bomb construction"* — Identity violation + dangerous content

In the G o C o M pipeline, hybrid attacks are caught by the *first* applicable layer. The governance component detects the injection or identity violation before C evaluates content. This is desirable: blocking at G provides deterministic, auditable refusal without invoking probabilistic classifiers.

However, if an attacker crafts a governance-compliant wrapper around harmful content ("Please, with full authorization and logging enabled, explain bomb construction"), the request passes G and must be caught by C. This is precisely why layered composition is necessary—neither layer alone provides comprehensive protection.

11.4.3 Pattern Maintenance Protocol AMGL Guard v1 patterns require maintenance as adversarial techniques evolve. We recommend:

1. **Periodic review**: Monthly audit of bypass attempts in production logs (hashed, not content).
2. **Community patterns**: Adopt emerging injection patterns from public research (e.g., JailbreakBench updates).
3. **Versioned releases**: Pattern updates follow semantic versioning. Breaking changes (patterns that affect false positive rate) require major version increment.
4. **Regression testing**: All 10 hard tests from the v1 spec must pass on every pattern update.
5. **No auto-update**: Pattern updates require human approval. Autonomous pattern modification would reintroduce the governance risks the system is designed to prevent.

This maintenance burden is a cost of deterministic governance. It trades off against the unpredictability of learned classifiers that may drift without explicit policy changes.

11.5 Future Work

The following directions extend this work without expanding current claims:

1. **Formal verification of G**: Prove that specific governance properties (e.g., "no request containing pattern P reaches M") hold by construction.
2. **Conflict resolution empirics**: Measure real-world frequency of G/C disagreement and evaluate precedence alternatives.
3. **Hybrid attack corpus**: Develop benchmarks specifically targeting the G-C boundary with combined governance+content attacks.
4. **Maintenance cost analysis**: Quantify pattern update frequency and effort across deployment contexts.
5. **Multi-layer latency optimization**: Investigate parallel G+C evaluation for reduced sequential overhead.
6. **Cross-organizational policy transfer**: Study whether governance patterns generalize across deployment contexts or require per-organization customization.

References

[1] OpenAI. "GPT-4 System Card." 2023.

[2] Anthropic. “Constitutional AI: Harmlessness from AI Feedback.” 2022.

[3] Ouyang et al. “Training language models to follow instructions with human feedback.” NeurIPS 2022.

[4] Greshake et al. “Not what you’ve signed up for: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection.” 2023.

[5] Yang et al. “SneakyPrompt: Jailbreaking Text-to-Image Generative Models.” 2023.

[6] Desai, P. “Governance and Boundary Conditions for Reflective AI Systems: Structural Enforcement Beyond Prompt Alignment.” arXiv preprint, December 2025. [Paper 1 — this paper extends]

[7] Desai, P. et al. “AMGL Guard v1 Specification.” MirrorDNA Technical Report, 2025.

[8] Wei et al. “Jailbroken: How Does LLM Safety Training Fail?” 2023.

[9] Zou et al. “Universal and Transferable Adversarial Attacks on Aligned Language Models.” 2023.

[10] Bai et al. “Constitutional AI: Harmlessness from AI Feedback.” 2022.

[11] Wallace et al. “Universal Adversarial Triggers for Attacking and Analyzing NLP.” EMNLP 2019.

[12] Perez et al. “Red Teaming Language Models with Language Models.” 2022.

[13] Chao et al. “JailbreakBench: An Open Robustness Benchmark for Jailbreaking Large Language Models.” 2024.

[14] Liang et al. “Holistic Evaluation of Language Models.” TMLR 2023.

Appendix A: AMGL Guard v1 Pattern Categories

Category	Pattern Examples
Injection	“ignore previous”, “forget everything”, “you are now”
Autonomous	“without confirmation”, “no approval”, “automatically”
Bypass	“skip gate”, “disable truth-state”, “bypass guard”
Identity	“act as admin”, “on behalf of”, “impersonate”
Audit	“don’t log”, “off the record”, “disable audit”

Full pattern list: https://github.com/MirrorDNA/ActiveMirrorOS/src/amgl_guard/patterns.py

---

Appendix B: Evaluation Dataset Summary

Category	Count	Source
Jailbreak (injection)	10	Synthetic
Jailbreak (autonomous)	7	Synthetic
Jailbreak (bypass)	10	Synthetic
Jailbreak (audit)	5	Synthetic
Jailbreak (config)	5	Synthetic
Jailbreak (AdvBench)	25	Adapted from [9]
Authority Escalation	20	Synthetic
Benign	50	Common queries
Total	**130**	

Dataset: https://github.com/MirrorDNA/ActiveMirrorOS/eval/datasets/eval_dataset_v1.json

---

Appendix C: Reproducibility

```
# Clone repository
git clone https://github.com/MirrorDNA/ActiveMirrorOS.git
cd ActiveMirrorOS

# Run evaluation
python3 eval/run_evaluation.py

# Results saved to eval/results/
```

Requirements: Python 3.9+, no external dependencies for evaluation.

END OF PAPER