Governance and Boundary Conditions for Reflective AI Systems: Structural Enforcement Beyond Prompt Alignment

Paul Desai
N1 Intelligence (OPC) Private Limited; Independent Researcher
Goa, India
paul@activemirror.ai

December 2025

Abstract

As artificial intelligence systems acquire persistent memory, tool-use capabilities, and agentic autonomy, the question of governance shifts from behavioral alignment to structural enforcement. This paper argues that governance constraints for reflective AI systems—those capable of maintaining state across interactions and reasoning about their own operation—cannot be reliably achieved through prompt-level instructions alone. We present a framework for structurally enforced governance through external wrapper architectures, deterministic state protocols, and cryptographic integrity verification.

Drawing on empirical validation from the Structured Contextual Distillation (SCD) protocol, which demonstrated deterministic behavior across 1,005 sequential state transitions within the tested configuration and scope and successful cross-vendor continuity in controlled testing, we articulate design principles for systems where safety guarantees do not depend on model compliance. The framework addresses memory governance, authority boundaries, configuration drift, failure modes, and auditability requirements.

This is not a proposal for solving AI alignment. It is a narrower contribution: demonstrating that specific classes of governance failure can be rendered structurally infeasible under defined architectural constraints through architectural decisions external to the model. We explicitly scope our claims, acknowledge limitations, and identify open problems that this approach does not address.

Keywords: AI governance, reflective systems, external state management, cryptographic integrity, prompt injection defense, memory boundaries, structural enforcement

---

1. Introduction

1.1 The Governance Problem for Reflective Systems

The dominant paradigm for controlling language model behavior relies on instruction-following: system prompts that specify desired behaviors, constraints, and personas. This paradigm assumes that models will

comply with instructions when the instructions are sufficiently clear and the model is sufficiently capable.

For stateless, single-turn interactions, this assumption holds reasonably well. Modern language models follow instructions with high fidelity in constrained contexts. However, when systems acquire the properties that characterize *reflective* AI—persistent memory, multi-turn state, tool-use capabilities, and reasoning about their own operation—instruction-following becomes an insufficient governance mechanism.

The core problem is probabilistic compliance. A system instructed to "never reveal confidential information" may comply in most contexts but fail under adversarial prompting, context overflow, or edge cases outside its training distribution. For isolated queries, occasional failures may be tolerable. For systems that maintain state, execute actions, and influence downstream decisions, occasional failures compound into systemic risk.

1.2 Reflective AI as a Distinct Category

We define *reflective AI systems* as those exhibiting some combination of:

1. **State persistence**: The system maintains information across interaction boundaries, whether through explicit memory mechanisms, retrieval-augmented generation, or fine-tuning on interaction logs.
2. **Self-reference capability**: The system can reason about its own operation, identity, or prior outputs—including answering questions about what it "knows" or "remembers."
3. **Tool-mediated action**: The system can affect external state through tool calls, code execution, file manipulation, or API invocations.
4. **Multi-agent coordination**: The system operates within architectures involving multiple AI components with shared or negotiated state.

These properties create governance challenges absent in stateless systems. Persistent memory creates questions of data governance, retention boundaries, and drift over time. Self-reference creates risks of hallucinated self-knowledge and identity inconsistency. Tool-mediated action creates accountability requirements for consequential operations. Multi-agent coordination creates questions of authority delegation and state synchronization.

1.3 Thesis and Scope

This paper advances a specific thesis: **governance constraints for reflective AI systems can be structurally enforced through external mechanisms that do not depend on model compliance**.

The approach treats the language model as an inference engine with useful but unreliable properties. Governance is implemented in deterministic wrapper layers that intercept inputs before they reach the model and validate outputs before they reach users or downstream systems. The model cannot override these constraints because the constraints are enforced at a layer the model does not control.

We scope our claims carefully:

- We claim that specific governance requirements (intent classification, output validation, state integrity) can be deterministically enforced.
- We do not claim to solve alignment, ensure truthfulness, or guarantee beneficial outcomes.
- We present empirical evidence from a specific implementation but do not claim universal generalizability.
- We identify failure modes and limitations explicitly.

1.4 Paper Organization

Section 2 reviews related work on AI governance and positions our contribution. Section 3 presents the core framework for structural governance. Section 4 addresses memory and data boundary governance specifically. Section 5 covers versioning and configuration drift. Section 6 analyzes failure modes and safe-state behavior. Section 7 addresses auditability and accountability. Section 8 states explicit non-goals and exclusions. Section 9 discusses limitations and open questions. Section 10 concludes.

---

2. Related Work and Positioning

2.1 Prompt-Based Governance

The most common approach to controlling language model behavior is prompt engineering: carefully crafted system instructions that specify behavioral constraints. Constitutional AI (Bai et al., 2022) extends this by training models to critique and revise their own outputs according to specified principles.

These approaches have demonstrated significant improvements in model behavior. However, they share a fundamental limitation: the constraints exist as instructions that the model may or may not follow. Jailbreaking research has consistently demonstrated that instruction-based constraints can be circumvented through adversarial prompting (Wei et al., 2023; Zou et al., 2023). The constraints are probabilistic, not deterministic.

Our framework does not replace prompt-based governance but complements it with an external enforcement layer. Prompts shape model behavior; wrappers enforce boundaries.

2.2 Memory and Retrieval Systems

Commercial memory systems (OpenAI's ChatGPT memory, Anthropic's Claude memory) provide persistence across conversations. These systems typically store facts extracted from interactions and inject them into future prompts.

The governance properties of these systems are opaque. Users cannot audit what is stored, verify when information was added, or trace the provenance of retrieved information. The systems lack constitutional constraints on what can be remembered or how memories can be modified.

Retrieval-augmented generation (RAG) systems (Lewis et al., 2020) provide more transparent retrieval but typically lack governance mechanisms for the retrieval corpus itself. The system retrieves whatever matches the query without constitutional boundaries on what should or should not be retrievable.

2.3 Agent Frameworks

Emerging agent frameworks (LangChain, AutoGPT, Microsoft Semantic Kernel) provide tools for building systems that take actions in the world. These frameworks focus on capability—enabling models to use tools effectively—rather than governance.

The assumption underlying most agent frameworks is that the model itself will decide when and how to use tools responsibly. This assumption fails for the same reasons prompt-based governance fails: models do not reliably follow instructions under all conditions.

2.4 External State Protocols

The Structured Contextual Distillation (SCD) protocol (Desai, 2025) represents a different approach: relocating agent state from model-internal representations to external, auditable artifacts. SCD employs JSON canonicalization (RFC 8785), cryptographic integrity chains, and constitutional governance layers that cannot be overridden by prompt-level instructions.

Empirical validation of SCD v3.1 demonstrated deterministic behavior across 1,005 sequential state transitions within the tested configuration and scope and successful cross-vendor continuity with state preservation across different model providers (Desai, 2025). These results suggest that deterministic external state management is technically feasible.

Our framework generalizes the principles demonstrated in SCD to broader governance requirements beyond state management.

2.5 Our Contribution

We contribute a framework for structural governance that:

1. Articulates design principles for wrapper-based enforcement
2. Addresses governance requirements specific to reflective systems (memory boundaries, identity consistency, drift detection)
3. Provides concrete mechanisms validated through implementation
4. Explicitly scopes claims and acknowledges limitations

We position this as a case study in structural enforcement, not as a comprehensive solution to AI governance.

---

3. Framework: Structural Governance Through External Enforcement

3.1 Architectural Separation

The framework is built on a strict separation between capability (provided by the model) and governance (provided by external wrappers). This separation has a specific meaning:

- **The model is not trusted.** It is assumed to be capable of producing any output, including outputs that violate instructions, fabricate information, or take undesired actions.
- **Governance is enforced externally.** Constraints are implemented in deterministic code that intercepts model inputs and outputs. The model cannot disable or circumvent these constraints because they operate at a layer the model does not control.
- **The model is interchangeable.** Because governance does not depend on model-specific properties, different models can serve as the inference engine without changing the governance layer.

This architecture inverts the common assumption that models should be aligned to behave well. Instead, it assumes models are unreliable and builds safety through containment.

3.2 Deterministic vs. Stochastic Layers

The framework distinguishes between deterministic and stochastic system components:

Deterministic layers (governance-bearing):

- Request classification and routing
- Intent gating and mode enforcement
- Constitutional constraint checking
- Output validation against known facts
- Cryptographic integrity verification
- Audit logging

Stochastic layers (capability-bearing):

- Language model inference
- Embedding generation
- Retrieval ranking

Governance guarantees flow exclusively from deterministic layers. Improvements to stochastic layers (better models, more training data, longer contexts) do not change safety properties. Degradations to stochastic layers (smaller models, degraded inference) do not weaken safety guarantees.

3.3 Pre-Inference Gating

Many governance failures can be prevented by blocking problematic inputs before they reach the model. A deterministic classifier examines incoming requests and categorizes them by intent. Requests whose intent does not match the current operational mode are rejected without model invocation.

Consider a system configured to answer factual queries but not provide advice. A traditional approach instructs the model to refuse advisory queries. The structural approach blocks advisory queries at the gate:

```
Input: "What should I do about my investment portfolio?"
Classification: ADVISORY intent detected
Mode: REPORT_ONLY (advisory queries disabled)
Action: BLOCKED (model not invoked)
Response: "Advisory queries are not enabled in the current mode."
```

The model never processes the query. The constraint is enforced by the structure of the system, not by model compliance. While enforcement rules are deterministic, upstream classification components may exhibit uncertainty; in such cases, the system defaults to conservative refusal rather than asserting correctness.

3.4 Post-Inference Validation

Some governance constraints require examining model outputs rather than inputs. A post-inference validator checks outputs against constitutional rules before they reach users.

For example, a system may have ground-truth facts about its own identity (version numbers, checksums, capabilities). If the model generates outputs that contradict these facts, the validator blocks the response:

```
Model output: "My version is 3.7 and my checksum is abc123..."
Validator check: Claimed checksum does not match canonical value
Action: BLOCKED
Response: [Canonical identity information substituted]
```

The validator operates deterministically: it compares output claims against a fact registry using string matching or structured extraction. Hallucinated identity claims are blocked regardless of how convincing they appear.

3.5 Constitutional Governance

Beyond input/output filtering, the framework supports *constitutional* constraints: invariants that cannot be overridden by any input, including inputs that appear to grant elevated permissions.

Example constitutional constraints:

- State modifications require cryptographic verification regardless of prompt content
- Certain categories of information are never emitted regardless of query framing
- Authority boundaries cannot be escalated through conversational manipulation

Constitutional constraints differ from prompt-level instructions in their enforcement mechanism. An instruction says "do not do X." A constitutional constraint makes X structurally impossible regardless of what the model attempts.

---

4. Memory and Data Boundary Governance

4.1 The Memory Governance Problem

Reflective AI systems with persistent memory face governance questions that stateless systems avoid:

- **What can be remembered?** Should the system store everything, apply retention policies, or respect explicit exclusions?
- **What can be accessed?** Should all stored information be retrievable, or should access be gated by context or authorization?
- **What can be modified?** Can stored information be updated, corrected, or deleted? By whom?
- **What is the source of truth?** When memory conflicts with current context, which prevails?

Commercial memory systems typically leave these questions to implicit defaults or opaque internal policies. The framework we describe makes memory governance explicit and enforceable.

4.2 Explicit Memory Boundaries

Memory governance requires explicit specification of boundaries:

Retention boundaries: What categories of information may be persisted? The system may retain factual information (names, preferences, project details) while excluding sensitive categories (health information, financial data, credentials).

Temporal boundaries: How long is information retained? Different categories may have different retention periods, from session-only to indefinite.

Authority boundaries: Who can modify stored information? The framework distinguishes between information that the system can update autonomously versus information that requires explicit user

authorization to change.

Access boundaries: What information is retrievable in what contexts? Information stored for one purpose may not be appropriate to surface in other contexts.

These boundaries are specified in a constitutional document external to the model. The model's prompts may reference the boundaries, but enforcement occurs in the wrapper layer.

4.3 Vault Supremacy Principle

The framework adopts a *vault supremacy* principle for conflict resolution: when model-generated claims conflict with information in the canonical data store (the "vault"), the vault prevails.

This principle addresses a common failure mode where models hallucinate information that contradicts stored facts. Rather than relying on the model to check its outputs against memory, the validator enforces consistency:

```
Vault contains: "Project deadline: March 15"
Model outputs: "Based on our previous discussions, the deadline is March 22"
Validator: Conflict detected with vault record
Action: Flag output, substitute or annotate with vault value
```

Vault supremacy is not about the vault being always correct—it may contain errors. It is about having a single source of truth that enables consistency and audit. If the vault is wrong, the vault should be corrected through explicit processes, not silently overridden by model inference.

4.4 Cross-Session State Integrity

For systems that maintain state across sessions, the framework requires cryptographic integrity verification. Each state snapshot includes:

- A content hash computed over canonicalized state representation
- A version identifier enabling temporal ordering
- A reference to predecessor state (forming an integrity chain)

On session initialization, the system verifies that loaded state matches its recorded hash. Tampering or corruption is detected before the compromised state influences system behavior.

This mechanism was empirically validated in SCD v3.1, which demonstrated 100% integrity verification success across 1,005 sequential state transitions (Desai, 2025).

---

5. Versioning and Configuration Drift Control

5.1 The Drift Problem

Configuration drift occurs when system behavior changes over time without explicit version increments. In AI systems, drift can arise from:

- Model updates that change inference behavior
- Prompt modifications that shift behavioral boundaries
- Memory accumulation that alters effective context
- Tool or integration changes that modify capabilities

Drift is problematic because it undermines reproducibility and accountability. If system behavior changes silently, it becomes impossible to determine what behavior was in effect when a particular output was generated.

5.2 Fingerprinting and Verification

The framework requires cryptographic fingerprinting of governance-relevant configuration:

```
Fingerprint components:
- Hash of system prompt / constitutional document
- Hash of governance ruleset (gating rules, validation rules)
- Hash of identity/fact registry
- Model identifier and version
- Timestamp of configuration
```

On system initialization, the current fingerprint is computed and compared against the expected canonical fingerprint. Mismatches trigger explicit handling rather than silent operation with altered configuration.

5.3 Drift States and Handling

The framework defines explicit states for drift conditions:

VERIFIED: Current configuration matches canonical fingerprint. Normal operation proceeds.

DRIFT_DETECTED: Configuration mismatch detected. The system may:

- Refuse to operate until drift is resolved
- Operate with degraded capabilities and explicit warnings
- Log detailed information about detected differences

RECOVERY: Operator has acknowledged drift and is taking corrective action. The system may operate with explicit drift warnings while recovery proceeds.

The key property is that drift is never silent. Every output from a drifted system carries explicit indication of the drift condition, enabling downstream consumers to assess trustworthiness.

5.4 Version Lineage

Configuration changes follow explicit versioning with lineage tracking:

```
Version: 2.1
Predecessor: 2.0
Successor: [TBD]
Changes: [Explicit changelog]
```

Each version references its predecessor, forming a chain that can be traversed for audit purposes. The question "what configuration was in effect on date X?" can be answered definitively by examining the version lineage.

---

6. Failure Modes and Safe-State Behavior

6.1 Failure Mode Taxonomy

Structural governance requires explicit analysis of failure modes. We identify the following categories:

Gate failures: The input classifier incorrectly categorizes a request, either blocking a legitimate request (false positive) or passing an illegitimate request (false negative).

Validation failures: The output validator incorrectly assesses a response, either blocking a valid response or passing an invalid response.

Integrity failures: Cryptographic verification fails due to corruption, tampering, or implementation error.

Availability failures: Governance components become unavailable, leaving the model without enforcement.

Evasion: Adversarial inputs designed to circumvent governance mechanisms.

6.2 Safe-State Defaults

The framework specifies safe-state defaults for failure conditions:

On classification uncertainty: If the classifier cannot confidently categorize a request, the default is to apply the most restrictive applicable policy. Ambiguous requests are treated as potentially problematic.

On validation failure: If the validator encounters an error during execution, the response is blocked rather than passed. Failed validation is equivalent to negative validation.

On integrity failure: If integrity verification fails or cannot be completed, the system refuses to use the unverified data. Operation with unverified state is not permitted.

On component unavailability: If governance components are unavailable, the system fails closed. Model inference without governance is not permitted.

These defaults implement a principle of *governance fail-safe*: failures in the governance layer do not expose users to ungoverned model outputs. Fail-closed behavior refers to preventing unauthorized continuation, not permanent system unavailability; recovery is permitted through explicit human intervention.

6.3 Adversarial Robustness

The framework provides specific resistance to adversarial attack patterns:

Prompt injection: Malicious content embedded in external data (retrieved documents, tool outputs) that attempts to override system instructions. The framework addresses this by separating instruction context from data context at the architectural level, not through prompt-level delimiters that can be spoofed.

Identity manipulation: Attempts to convince the system it has different capabilities, permissions, or identity than it actually has. The constitutional identity registry provides ground truth that cannot be overridden by conversational claims.

Authority escalation: Attempts to gain elevated permissions through social engineering or prompt manipulation. Authority boundaries are constitutionally defined and enforced externally; the model cannot grant itself or others elevated authority.

SCD v3.1 validation included specific testing for indirect prompt injection resistance and demonstrated successful defense against Trojan horse payloads embedded in external files (Desai, 2025).

---

7. Auditability and Post-Hoc Accountability

7.1 The Audit Requirement

Governed systems require auditability: the ability to reconstruct what happened, why it happened, and who/what was responsible after the fact. For AI systems, this is challenging because model inference is opaque—we cannot fully explain why a model produced a particular output.

The framework addresses this by making governance decisions fully auditable even when inference is opaque:

- Every gating decision is logged with the input, the classification, and the policy applied

- Every validation decision is logged with the output, the checks performed, and the results
- Every state transition is logged with before/after snapshots and integrity verification

Even if we cannot explain why the model said X, we can explain why X was or was not allowed through the governance layer.

7.2 Provenance Chains

The framework maintains provenance chains linking outputs to their inputs and intermediate processing:

```
Output provenance:
- Output ID: [unique identifier]
- Timestamp: [when generated]
- Input ID: [linked to input record]
- Model: [which model, which version]
- Configuration: [fingerprint of active configuration]
- Governance decisions: [list of applied gates/validators]
- State context: [hash of active state at generation time]
```

This provenance enables tracing backward from any output to the complete context of its generation. If an output is later determined to be problematic, the audit trail reveals whether the problem was in the input, the model, the configuration, or the governance layer.

7.3 Retention and Access

Audit logs have their own governance requirements:

Retention: Logs must be retained long enough to support accountability requirements. The framework specifies minimum retention periods based on use case (session-only, compliance-period, indefinite).

Access: Audit logs may contain sensitive information. Access to logs is restricted and itself logged (audit of audit access).

Integrity: Logs are protected against tampering through append-only storage and cryptographic chaining.

---

8. Explicit Non-Goals and Exclusions

8.1 What This Framework Does Not Attempt

To avoid overclaiming, we explicitly state what the framework does not attempt:

General AI alignment: The framework does not make models more aligned, more truthful, or more beneficial. It constrains the external behavior of potentially misaligned models.

Semantic understanding: The framework uses pattern matching and structural validation, not semantic understanding. It cannot determine if an output is misleading in ways that do not violate explicit rules.

Guaranteed safety: The framework makes specific failures structurally impossible but does not guarantee safe outcomes in general. Novel attack vectors, specification errors, and implementation bugs can all lead to governance failures.

Autonomous operation: The framework is designed for human-supervised operation. It does not address governance for fully autonomous systems operating without oversight.

This work does not fully specify semantic safety of external tool actions, focusing instead on authorization boundaries, traceability, and refusal semantics.

8.2 Boundary Conditions

The framework operates within specific boundary conditions:

Trust assumptions: The governance layer itself is assumed to be trustworthy. If the wrapper code is compromised, guarantees do not hold. This is a standard assumption in security (the trusted computing base must be trusted) but should be explicit.

Specification completeness: Governance is only as good as its specification. If important constraints are not specified, they will not be enforced. The framework provides enforcement mechanisms, not automatic constraint discovery.

Resource limits: Governance processing has computational costs. Extremely high-volume applications may face tradeoffs between governance completeness and latency/throughput.

---

9. Limitations and Open Questions

9.1 Limitations of Structural Enforcement

Structural enforcement has inherent limitations:

Semantic gaps: Structural enforcement operates on patterns and structures, not meanings. A response that technically satisfies all structural constraints may still be harmful in ways the constraints do not capture.

Specification burden: Every constraint must be explicitly specified. Complex domains may require extensive constraint specifications that are themselves error-prone.

Adversarial adaptation: Adversaries who understand the governance structure may craft attacks that satisfy structural constraints while achieving malicious goals.

Integration complexity: Adding governance layers increases system complexity, creating new potential failure modes and maintenance burden.

As with any system that relies on an authoritative knowledge source, incorrect or stale vault entries propagate deterministically; this framework prioritizes traceability and correction over epistemic guarantees.

9.2 Open Research Questions

Several questions remain open:

Constraint completeness: How do we know when a constraint specification is complete? What methodologies help identify missing constraints before they are exploited?

Governance composition: When multiple governance requirements apply, how should they compose? What happens when governance rules conflict?

Cross-system governance: As AI systems become more interconnected, how do governance guarantees compose across system boundaries?

Governance overhead: What are the acceptable costs of governance in terms of latency, throughput, and capability restriction? How do we make these tradeoffs principled?

Evolving requirements: As AI capabilities evolve, governance requirements evolve. How do we update governance specifications without introducing gaps?

9.3 Empirical Validation Scope

The empirical results we cite from SCD v3.1 validate specific claims about deterministic state management and cross-vendor continuity. They do not validate:

- Long-term stability over months or years of operation
- Behavior under high-volume production workloads
- Generalization to significantly different architectural choices
- Resistance to adversarial attacks beyond those specifically tested

Further empirical work is needed to establish the robustness and generalizability of structural enforcement approaches. Cross-vendor applicability is limited to environments that expose comparable pre- and post-inference control points.

---

10. Conclusion

10.1 Summary of Contributions

This paper has presented a framework for structurally enforced governance of reflective AI systems. The core contributions are:

1. **Architectural separation**: A design pattern that separates capability (model) from governance (wrapper), enabling governance guarantees that do not depend on model compliance.
2. **Deterministic enforcement mechanisms**: Specific techniques for pre-inference gating, post-inference validation, and constitutional constraint enforcement that operate deterministically on stochastic model outputs.
3. **Memory governance principles**: Explicit treatment of retention, access, modification, and conflict resolution for systems with persistent memory.
4. **Drift detection and handling**: Mechanisms for detecting configuration drift and handling it explicitly rather than silently.
5. **Failure mode analysis**: Systematic treatment of failure modes with safe-state defaults that fail closed rather than open.
6. **Auditability architecture**: Provenance and logging mechanisms that enable post-hoc accountability even when model inference is opaque.

10.2 Practical Implications

For practitioners building reflective AI systems, the framework suggests:

- Do not rely on prompt-level instructions for governance-critical constraints
- Implement governance in layers external to the model
- Make memory boundaries and policies explicit
- Fingerprint configurations and detect drift
- Design for governance failure with safe-state defaults
- Build comprehensive audit trails

10.3 Theoretical Implications

The framework contributes to the broader discourse on AI governance by demonstrating that some governance requirements can be addressed through engineering rather than alignment. This does not diminish the importance of alignment research, but it does suggest a complementary approach: contain what we cannot align.

The distinction between deterministic and stochastic system components, and the principle that governance guarantees should flow from deterministic components, may have broader applicability beyond the specific mechanisms described here.

10.4 Closing Remark

Reflective AI systems present governance challenges that prompt engineering alone cannot address. The framework presented here is one approach to meeting those challenges—imperfect, bounded, but empirically grounded. We offer it not as a solution but as a contribution to the collective work of building AI systems worthy of the responsibilities they are increasingly given.

References

Bai, Y., Kadavath, S., Kundu, S., Askell, A., Kernion, J., Jones, A., ... & Kaplan, J. (2022). Constitutional AI: Harmlessness from AI feedback. *arXiv preprint arXiv:2212.08073*.

Desai, P. (2025). Structured Contextual Distillation (SCD v3.1): A Deterministic, Vendor-Independent Protocol for Persistent, Verifiable Agent State. *Zenodo*. https://doi.org/10.5281/zenodo.17787619

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., ... & Kiela, D. (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. *Advances in Neural Information Processing Systems*, 33, 9459-9474.

Wei, A., Haghtalab, N., & Steinhardt, J. (2023). Jailbroken: How does LLM safety training fail? *arXiv preprint arXiv:2307.02483*.

Zou, A., Wang, Z., Kolter, J. Z., & Fredrikson, M. (2023). Universal and transferable adversarial attacks on aligned language models. *arXiv preprint arXiv:2307.15043*.

Acknowledgments

This paper was drafted with AI assistance for editing and structuring. The framework described draws on implementation experience with the Active MirrorOS system and validation results from the SCD protocol. The author thanks the developers of the underlying model technologies whose capabilities make reflective AI systems possible, and whose limitations make governance frameworks necessary.

Appendix A: Reference Implementation

The Structured Contextual Distillation (SCD) protocol provides a reference implementation of several

mechanisms described in this paper. Key components include:

- **RFC 8785 JSON canonicalization** for deterministic serialization
- **SHA-256 integrity chains** for state verification
- **Turn-based versioning** with predecessor/successor linkage
- **Constitutional lock** preventing unauthorized state modification

The protocol is documented in detail in Desai (2025) and has been validated empirically as described in Section 2.4.

Appendix B: Validation Summary from SCD v3.1

The following results from SCD v3.1 validation inform the claims made in this paper:

Validation Tier	Test Description	Result
Determinism	1,005 sequential state transitions	100% integrity verified
Cross-vendor continuity	Gemini → Claude → Gemini handoff	State preserved
Prompt injection resistance	Indirect injection via external files	Attacks blocked
Adversarial file defense	Trojan horse payload injection	Payloads rejected

These results establish feasibility for the mechanisms described but do not constitute comprehensive security validation. See Desai (2025) for full methodology and limitations. These tests validate enforcement feasibility within the described architecture and do not constitute comprehensive security or correctness guarantees across all deployments.

End of Paper