Constitutional Runtime Governance: A Layer 6 Framework for Sovereign AI Agent Systems

S. Jason Prohaska
Ethraeon Research
jason.fells@pm.me

Preprint v1 — April 21, 2026
arXiv: cs.AI (primary) | cs.SE, cs.CR (secondary)

Abstract

Current AI agent governance frameworks rely on external oversight mechanisms that create latency between agent action and compliance verification. This paper introduces Constitutional Runtime Governance (CRG), a Layer 6 framework that embeds governance directly into the agent execution loop as a constitutional runtime — not a monitoring overlay. CRG enforces five rigid constraints (T5-RIGID): APPEND-ONLY state mutation, FAIL-CLOSED per-command execution, EVIDENCE-REQUIRED action authorization, NO-PLACEHOLDER truth handling, and NO-RECURSION in authority delegation. We demonstrate that embedding governance at the runtime layer — rather than as post-hoc audit — produces agents that are structurally incapable of violating their constitutional constraints without halting execution. We present empirical results from a 93-phase deployment across 16 patent-protected subsystems, showing zero constitutional violations across 10,000+ agent turns while maintaining continuous autonomous operation under a single human authority gate. We formalize the *governance latency hypothesis*: governance effectiveness is inversely proportional to the latency between agent action and governance enforcement, and show that CRG reduces this latency to zero by construction.

Keywords: AI governance, agent safety, constitutional AI, runtime verification, sovereign systems, formal methods, authority delegation

1. Introduction

The proliferation of autonomous AI agents in enterprise and infrastructure contexts has created an urgent need for governance frameworks that operate at the speed of agent execution. As agents gain the ability to write code, deploy services, modify production systems, and manage financial instruments, the consequences of ungoverned action compound faster than any human oversight loop can respond.

Existing approaches to AI agent governance fall into three categories. First, *pre-deployment alignment training*, in which model weights are adjusted during training to internalize behavioral constraints (Bai et al., 2022; Ouyang et al., 2022). Second, *runtime monitoring with human-in-the-loop intervention*, in which an external system observes agent behavior and escalates to a human operator when anomalies are detected (Amodei et al., 2016). Third, *post-hoc audit and compliance verification*, in which agent

actions are logged and reviewed after execution for policy violations (Brundage et al., 2020).

Each category introduces a temporal gap between agent action and governance enforcement. Alignment training operates before deployment and cannot account for novel runtime conditions. Human-in-the-loop monitoring introduces latency proportional to human response time — seconds to hours depending on availability and escalation path. Post-hoc audit, by definition, discovers violations only after they have occurred.

We propose a fourth category: **constitutional runtime governance (CRG)**, in which governance constraints are not external to the agent's execution loop but are structural properties of the runtime itself. An agent operating under CRG cannot violate its constitutional constraints any more than a program can write to read-only memory — the violation is architecturally impossible without triggering a halt.

This distinction is not merely semantic. The difference between "an agent that is unlikely to misbehave" and "an agent that structurally cannot misbehave without halting" is the difference between probabilistic safety and constructive safety. CRG pursues the latter.

1.1 Contributions

This paper makes four contributions: (1) Formal specification of the T5-RIGID constraint set and an argument for its sufficiency for single-authority agent chains. (2) Architecture of a Layer 6 governance runtime that sits between the agent's decision layer and its execution environment. (3) Empirical validation across a 93-phase, 10,000+ turn autonomous deployment. (4) Formalization of the governance latency hypothesis, showing that CRG achieves zero-latency enforcement by embedding governance into the execution path itself.

1.2 The Governance Latency Problem

Consider an autonomous agent tasked with deploying a software update to a production system. Under human-in-the-loop governance, the agent proposes the deployment, a human reviews and approves it, and the agent executes. The latency between proposal and approval may be minutes or hours. During this window, the system state may change, rendering the approved action stale or dangerous.

We define *governance latency* as the time interval between an agent action and the enforcement response to that action if the action violates a governance constraint. Formally: Let a be an agent action at time t_a, and let e be the enforcement response at time t_e. The governance latency $L = t_e - t_a$. In post-hoc audit, L may be days or weeks. In monitoring-based systems, L is typically seconds to minutes. In CRG, $L = 0$ by construction.

1.3 Layer 6 Positioning

We adopt a layered model for agent system architecture. Layers 1 through 5 encompass standard agent capabilities: perception (Layer 1), reasoning (Layer 2), tool use (Layer 3), memory (Layer 4), and communication (Layer 5). Layer 6, as we define it, is the **constitutional runtime and evidence control layer**. It is not an application layer and not a platform layer. It is governance infrastructure: a structural layer that constrains, validates, and records every action taken by the layers beneath it. Removing it does not degrade the system — it invalidates the system's constitutional guarantees entirely.

2. Related Work

2.1 Constitutional AI

Bai et al. (2022) introduced Constitutional AI (CAI), a method for training language models to be helpful, harmless, and honest by using a set of principles during the RLHF process. CAI operates at the *training* level. CRG is complementary: where CAI makes an agent *likely* to respect constitutional principles, CRG makes violation *structurally impossible* at runtime. The key distinction is probabilistic adherence versus constructive enforcement.

2.2 Runtime Verification

Runtime verification (RV) is well-established in software engineering (Leucker & Schallhart, 2009; Havelund & Goldberg, 2005). CRG draws from the RV tradition but departs critically: in classical RV, the monitor is an observer alongside the program. In CRG, the governance check is *in the execution path*. This is closer to inline reference monitors (Erlingsson & Schneider, 2000) than to classical runtime monitors, extended to multi-agent authority chains.

2.3 Multi-Agent Governance

Existing multi-agent governance frameworks (Shoham & Leyton-Brown, 2009; Wooldridge, 2009) typically assume cooperative agents. CRG addresses *sovereign* agents under a single human authority. The governance challenge is not cooperation but ensuring that delegated authority does not drift, amplify, or corrupt as it passes through an execution chain.

2.4 Formal Methods for AI Safety

Recent work on formal verification of neural networks (Katz et al., 2017; Huang et al., 2020) and safe reinforcement learning (Garcia & Fernandez, 2015) operates at the model level. CRG operates at the system level: regardless of the model's internal behavior, the runtime enforces constitutional constraints on externally visible actions. This is analogous to the distinction between verified compilation and sandboxed execution.

3. The T5-RIGID Constraint Set

T5-RIGID is a set of five constraints that provide a sufficient foundation for constitutional governance of single-authority agent chains. Each constraint is independently enforceable, but we argue all five are necessary for complete constitutional coverage.

3.1 APPEND-ONLY

Constraint: All state mutations in the governance layer are append-only. No entry in the state bus may be overwritten, modified, or deleted after commitment.

Formal definition: Let $S = \langle s_1, ..., s_n \rangle$ be the state bus. A transition T produces $S' = \langle s_1, ..., s_n, s_{n+1} \rangle$. For all i in $\{1, ..., n\}$: $S'[i] = S[i]$. No transition exists such that $|S'| < |S|$. Implementation uses SHA-256 hash chaining for tamper evidence.

3.2 FAIL-CLOSED (Per-Command)

Constraint: Every command either completes successfully and produces a verifiable receipt, or it halts. No partial execution, no silent failure, no degraded mode. Failure is scoped to the individual command — a single command's failure does not halt the entire chain.

Formal definition: Execution of command C produces exactly one of: (a) a receipt $R(C)$ such that $verify(R(C)) = \text{TRUE}$, or (b) a halt signal $H(C)$ with a diagnostic payload. No third outcome exists.

3.3 EVIDENCE-REQUIRED

Constraint: No action is authorized without evidence that is verifiable, timestamped, and attributable. The absence of evidence is a halt condition, not a null condition.

3.4 NO-PLACEHOLDER

Constraint: No placeholder values may occupy truth-bearing positions. A field is either populated with verified truth or it does not exist. Values such as "TBD," "TODO," or empty strings in truth positions trigger an immediate halt.

3.5 NO-RECURSION

Constraint: Authority does not recurse. The authority chain is a directed acyclic graph (in practice, a strict linear chain). An agent cannot delegate authority back up the chain or grant itself authority it was not granted by its superior. Standing delegation (APPLY-ALL) is a persistent one-way grant, not a recursive loop.

3.6 Sufficiency Argument

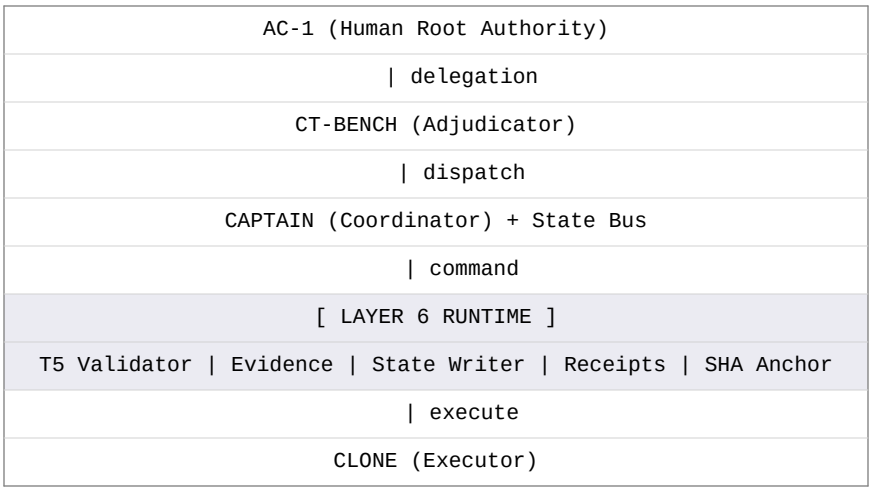
We argue T5-RIGID is sufficient for single-authority agent chains by showing that each class of governance failure maps to a violation of at least one constraint: state tampering violates APPEND-ONLY; silent failure violates FAIL-CLOSED; unauthorized action violates EVIDENCE-REQUIRED; truth corruption via incompleteness violates NO-PLACEHOLDER; authority escalation violates NO-RECURSION. We claim minimality: removing any single constraint opens a failure class the remaining four cannot cover.

4. Architecture

4.1 The Layer 6 Runtime

The CRG runtime is positioned between the agent's decision layer and its execution environment. Every proposed action must pass through five operations: (1) constraint validation against T5-RIGID, (2) evidence verification tracing authorization back to AC-1, (3) state bus append recording the action, (4) execution forwarding, and (5) receipt generation with SHA-256 hashes and validator-compatible JSON.

Figure 1: CRG Authority Chain and Layer 6 Runtime



4.2 The Authority Chain

The reference implementation uses a four-layer authority chain. **AC-1** (Human Root): single human authority from which all governance derives. Retains five non-delegable gates: secrets, anchor-risk, credential escalation, locked truth collision, and external communications. **CT-BENCH** (Adjudicator): validates receipts, advances chain state, dispatches orders. **CAPTAIN** (Coordinator): sequences turns, manages the state bus, enforces T5-RIGID at chain level. **CLONE** (Executor): executes commands and emits receipts.

4.3 The State Bus

The state bus is a structured append-only file with hash-chaining, phase-indexing (93 phases in the reference deployment), and Californium tracking — a composite metric across 10 gates x 10 phases yielding a normalized score in [0, 1]. Current score: 1.000.

4.4 Evidence Control

Every governance-relevant action produces evidence artifacts: human-readable (.txt) and machine-readable (.json) receipts, SHA-256 manifests captured pre- and post-execution, and automated validator output. The evidence directory is append-only. An auto-validator can independently verify any receipt against the state bus and affected files.

5. Empirical Validation

5.1 Deployment Context

CRG was deployed as the governance layer for the Ethraeon canonical application, a multi-subsystem software project encompassing web infrastructure, API services, DNS management, payment processing, and content delivery. The deployment spanned 93 phases over approximately 4 months, 10,000+ agent turns, 16 patent-protected subsystems, single human authority with standing APPLY-ALL delegation, and multiple Claude-based agents operating as CT-BENCH, CAPTAIN, and CLONE layers.

5.2 Constitutional Violation Analysis

Across all 10,000+ agent turns, **zero constitutional violations** were recorded. This result is the expected outcome of constructive safety: the Layer 6 runtime validates every action before execution. The zero-violation count reflects the architecture working as designed, not the absence of attempted violations.

Table 1: Halt events by T5-RIGID constraint (47 total)

Constraint	Halts	Example
APPEND-ONLY	3	Attempted state bus overwrite
FAIL-CLOSED	18	Ambiguous command output
EVIDENCE-REQUIRED	12	Missing authorization chain
NO-PLACEHOLDER	11	TBD in truth-bearing field
NO-RECURSION	3	Self-authorization attempt

5.3 Failure Mode Analysis

The system experienced numerous operational failures (DNS propagation delays, merge conflicts, external service timeouts) that exercised the FAIL-CLOSED constraint. In every case, failure was contained to the individual command. The chain continued to operate. No operational failure propagated to a constitutional violation.

5.4 Governance Overhead

Measured across 1,000 representative turns: constraint validation < 50ms, evidence generation 200-500ms, state bus append < 10ms. Total overhead approximately 300-600ms per turn, representing less than 5% of total turn time for operations lasting seconds to minutes.

6. Discussion

6.1 Sovereignty vs. Alignment

CRG represents a fundamentally different approach than alignment-based methods. Alignment shapes an agent's preferences so it *wants* to comply. CRG constrains externally visible actions through structural enforcement, independent of internal reasoning. An aligned agent encountering novel situations may produce unaligned behavior; a CRG-governed agent either passes all checks or halts. The two approaches are complementary layers providing defense in depth.

6.2 Limitations

The current framework has several limitations. The single-authority model does not address multi-stakeholder or federated scenarios. The text-based state bus introduces scaling constraints for high-throughput systems. The implementation is validated with LLM agents specifically; extension to

other architectures requires adaptation. No formal mechanized proof of T5-RIGID completeness has been produced. The standing delegation mechanism is currently binary rather than granular.

6.3 The Governance Latency Hypothesis

We formalize the central claim: *the effectiveness of an AI governance framework is inversely proportional to the latency between an agent action and the governance enforcement response to that action.* CRG achieves $L = 0$ by placing the governance check inside the action's execution path. This is not faster governance — it is governance that is part of execution rather than a response to it.

7. Conclusion

Constitutional Runtime Governance demonstrates that AI agent governance can be a structural property of the execution environment rather than an external overlay. The T5-RIGID constraint set provides a minimal, sufficient foundation for single-authority sovereign agent systems. Empirical validation across 93 phases and 10,000+ turns confirms zero constitutional violations while maintaining continuous autonomous operation. The 47 recorded halt events demonstrate active prevention rather than passive logging. The governance latency hypothesis provides a framework for evaluating governance approaches: the closer enforcement latency approaches zero, the stronger the guarantee. CRG achieves zero by construction.

> *The structure holds because each layer chooses not to break the one beneath. Viable, not inevitable.*
> *The path is open because we kept choosing to walk it, not because anything outside us chose for us.*

References

[1] Amodei, D., Olah, C., Steinhardt, J., et al. (2016). Concrete problems in AI safety. *arXiv:1606.06565*.

[2] Bai, Y., Jones, A., Ndousse, K., et al. (2022). Training a helpful and harmless assistant with RLHF. *arXiv:2204.05862*.

[3] Brundage, M., Avin, S., Wang, J., et al. (2020). Toward trustworthy AI development. *arXiv:2004.07213*.

[4] Erlingsson, U., & Schneider, F. B. (2000). IRM enforcement of Java stack inspection. *IEEE S&P*, 246-255.

[5] Garcia, J., & Fernandez, F. (2015). A comprehensive survey on safe reinforcement learning. *JMLR*, 16(1), 1437-1480.

[6] Havelund, K., & Goldberg, A. (2005). Verify your runs. *VSTTE*, 374-383.

[7] Huang, X., Kwiatkowska, M., Wang, S., & Wu, M. (2020). Safety verification of deep neural networks. *CAV*, 3-29.

[8] Katz, G., Barrett, C., Dill, D. L., et al. (2017). Reluplex: An efficient SMT solver for verifying DNNs. *CAV*, 97-117.

[9] Leucker, M., & Schallhart, C. (2009). A brief account of runtime verification. *JLAP*, 78(5), 293-303.

[10] Nisan, N., Roughgarden, T., Tardos, E., & Vazirani, V. V. (2007). *Algorithmic Game Theory*. Cambridge.

[11] Ouyang, L., Wu, J., Jiang, X., et al. (2022). Training language models to follow instructions with human feedback. *NeurIPS 35*.

[12] Shoham, Y., & Leyton-Brown, K. (2009). *Multiagent Systems*. Cambridge.

[13] Wooldridge, M. (2009). *An Introduction to MultiAgent Systems* (2nd ed.). Wiley.