RUNTIME ENFORCEMENT AND THE AI GOVERNANCE STACK

Embedding Execution Constraint Across the Governance Lifecycle

NOAH M. KENNEY

Founder & Principal Consultant, **Digital 520**

President & Chief Scientist, **Disruptive AI Lab**

President, **Ethical Tech Forum**

A COMPANION PAPER TO

Governing Intelligence: Law, Privacy, Security, and Compliance in the Age of Artificial Intelligence (2026)

Abstract

The AI Governance Stack, introduced in Governing Intelligence (2026), provides a five-layer framework for operationalizing governance across data, models, systems, monitoring, and audit. Since its publication, practitioners and researchers have raised a recurring question: where does runtime enforcement live in this model? As AI systems transition from advisory tools to autonomous agents capable of executing consequential actions, the ability to constrain behavior at the moment of execution has emerged as a central concern. This paper argues that runtime enforcement is not a missing sixth layer of the Stack, but a cross-cutting capability that must mature within each of the five existing layers. It examines the architectural, regulatory, and organizational dimensions of this challenge, and proposes a set of design principles for embedding enforcement into the governance lifecycle.

Keywords: AI governance, runtime enforcement, execution constraint, agentic AI, pre-execution authorization, governance architecture, AI Governance Stack

1. Introduction

The AI Governance Stack was designed to address a persistent gap in the field: the distance between governance principles and operational implementation. Where most frameworks describe what organizations should do, the Stack provides a structured model for how governance functions are distributed across five layers: Data Governance, Model Governance, System Integration, Control & Monitoring, and Audit & Evidence.

Since the publication of *Governing Intelligence*, a consistent theme has emerged in practitioner conversations: even with all five layers in place, a system can still execute actions that should not occur. A model can be validated, a system can be monitored, and an audit trail can be maintained, yet the system may still take an action at a moment when conditions no longer justify it. The question practitioners are asking is not whether the system is governed, but whether execution is still admissible right now.

This is a legitimate and important challenge. As AI systems evolve from tools that inform human decisions to agents that execute decisions autonomously, governance must account for the moment of action, not just the structures surrounding it. The purpose of this paper is to address that challenge directly: to explain why runtime enforcement is best understood as a cross-cutting capability rather than a discrete layer, and to propose design principles for embedding it across the Stack.

2. The Shift from Advisory to Agentic AI

Most existing governance frameworks, including the AI Governance Stack as originally presented, were designed in a context where AI systems primarily informed human decision-making. A credit scoring model produces a recommendation; a human reviews and acts on it. A content moderation system flags potential violations; a human reviewer makes the final call. In this paradigm, the human in the loop serves as the natural enforcement point. Governance structures ensure the system is built, trained, and monitored correctly, and the human provides the execution constraint.

The emergence of agentic AI systems fundamentally changes this dynamic. When AI agents can initiate multi-step workflows, interact with external systems, authorize transactions, or modify their own operational parameters, the human is no longer reliably positioned between the system and the consequence. The governance challenge shifts from ensuring that humans have the information they need to make good decisions, to ensuring that systems cannot take actions that exceed their authorization or violate their constraints.

This shift does not invalidate existing governance frameworks. It does, however, demand that they evolve. The question is how.

3. Why Runtime Enforcement Is Not a Sixth Layer

The most intuitive response to the enforcement gap is to propose a new layer. If the Stack has five layers and none of them fully address runtime constraint, perhaps it needs a sixth: an Execution Enforcement layer that sits between Control & Monitoring and the system's operational output. This framing is appealing in its simplicity, but it mischaracterizes the nature of the problem.

3.1 Enforcement Is Not Localized

Runtime enforcement cannot be meaningfully isolated into a single architectural layer because the conditions that determine whether an action is admissible span the entire Stack. Consider a credit decisioning agent that autonomously approves loan applications. The admissibility of any given approval depends on:

> **Data Governance (Layer 1):** Is the data the model is operating on still current? Has the applicant's financial profile changed since the data was ingested? Are the data sources still authorized for this use under applicable privacy regulations?
>
> **Model Governance (Layer 2):** Has the model drifted beyond acceptable performance thresholds? Has a regulatory change invalidated the features the model relies on? Is the model still operating within the population distribution it was validated against?
>
> **System Integration (Layer 3):** Are the downstream systems the agent will interact with (payment processors, notification services, credit bureaus) available and operating within expected

parameters? Are the API contracts still valid?

Control & Monitoring (Layer 4): Has a drift alert been triggered that should pause autonomous operation? Is the system currently operating under a degraded-mode policy due to an upstream incident? Has a human override been issued that the agent has not yet incorporated?

Audit & Evidence (Layer 5): Can the system produce a verifiable record of why this specific action was taken at this specific moment? Is the evidence chain sufficient to withstand regulatory scrutiny if the action is challenged?

Each of these questions must be answered affirmatively before an action should be permitted. No single layer owns the answer. This is why enforcement is fundamentally cross-cutting: it requires information, constraints, and verification mechanisms that span all five layers.

3.2 The Analogy to Security

A useful parallel exists in cybersecurity. Organizations do not treat security as a single layer in their technology stack. Security is embedded in the network layer (firewalls, segmentation), the application layer (input validation, authentication), the data layer (encryption, access controls), and the operational layer (monitoring, incident response). Attempting to consolidate all security functions into a single layer would create a brittle architecture that fails at the boundaries.

The same principle applies to runtime enforcement in AI governance. Enforcement must be distributed across the architecture, with each layer contributing the constraints it is best positioned to evaluate. The challenge is not to build a new layer, but to mature the enforcement capabilities within each existing one.

4. Enforcement Capabilities Across the Stack

If runtime enforcement is a cross-cutting capability, what does it look like in practice within each layer? This section examines the enforcement mechanisms that each layer of the Stack must develop as AI systems become more autonomous.

4.1 Data Governance: Freshness and Authorization Gates

In an advisory context, data governance focuses on collection, quality, consent, and lineage. In an agentic context, data governance must additionally ensure that the data an agent is acting on remains valid at the moment of execution. This introduces the concept of data freshness gates: mechanisms that verify data currency before permitting consequential actions. If a model is making a credit decision based on financial data that was ingested 90 days ago, and the regulatory requirement specifies 30-day currency, the data layer must be capable of blocking the action, not merely logging the staleness for later review.

Similarly, data authorization must become dynamic. Privacy regulations like the GDPR require that data processing be tied to specific, documented purposes. When an agent repurposes data across contexts (for example, using customer service interaction data to inform a credit decision), the data governance layer must enforce purpose limitation at runtime, not rely on a static data use agreement reviewed during system design.

4.2 Model Governance: Conditional Validity

Model governance traditionally focuses on validation at the point of deployment: bias testing, fairness metrics, performance benchmarks, and explainability assessments. Runtime enforcement extends this to conditional validity: the ongoing question of whether the model's outputs remain trustworthy under current conditions.

This includes population drift detection (is the model encountering inputs that differ meaningfully from its training distribution?), concept drift detection (has the relationship between inputs and outcomes changed?), and regulatory drift detection (has a new regulation or guidance changed the acceptable use of certain model features?). Each of these conditions can render a previously validated model unreliable in ways that periodic revalidation cycles will not catch. The enforcement mechanism here is not a human reviewing a quarterly report, but an automated gate that constrains or pauses model-driven actions when validity conditions are no longer met.

4.3 System Integration: Architectural Constraint

The System Integration layer is where enforcement has the most natural architectural home, and where the most work remains to be done. In current practice, system integration focuses on how models are deployed into production environments: API design, access controls, rate limiting, and interoperability. For agentic systems, this layer must also define the execution boundary: the set of actions the system is authorized to take, under what conditions, with what constraints, and with what escalation paths.

Emerging patterns include circuit-breaker architectures (where systems automatically halt operations when error rates exceed thresholds), policy-as-code engines (where governance rules are expressed as executable policies evaluated at runtime), and capability-based authorization (where agents are granted specific, scoped permissions rather than broad access). These patterns are not yet standardized, and their implementation varies significantly across organizations and use cases. This is by design: the Stack's principles-based approach recognizes that prescriptive enforcement architectures will differ based on system architecture, risk profile, regulatory context, and operational requirements.

4.4 Control and Monitoring: From Detection to Constraint

The Control & Monitoring layer, as originally designed, emphasizes tiered human oversight: preventive controls (before deployment), concurrent controls (during operation), and post-hoc controls (after incidents).

The enforcement evolution of this layer involves tightening the feedback loop between detection and response.

In an advisory system, detecting drift or anomalous behavior triggers an alert that a human can investigate on a reasonable timeline. In an agentic system, the time between detection and consequence may be measured in milliseconds. This requires automated escalation policies that can constrain system behavior in real time: reducing the scope of permitted actions, requiring human approval for actions above certain thresholds, or pausing autonomous operation entirely until the anomaly is resolved.

The key distinction is between monitoring as observation and monitoring as governance. Observation produces information. Governance produces constraint. The Control & Monitoring layer must evolve to do both.

4.5 Audit and Evidence: Runtime Evidence Generation

The Audit & Evidence layer is traditionally concerned with demonstrating compliance after the fact: maintaining logs, producing reports, and supporting investigations. Runtime enforcement extends this layer's role in two important ways.

First, evidence must be generated at the moment of execution, not reconstructed later. When an agent takes a consequential action, the audit trail must capture not only what action was taken, but the complete decision context: what data was available, what model produced the output, what constraints were evaluated, and what authorization was granted. This is the difference between forensic evidence (what happened?) and compliance evidence (was the action justified at the moment it occurred?).

Second, the evidence layer must be capable of supporting real-time compliance verification. Under regulations like the EU AI Act, high-risk AI systems must demonstrate ongoing compliance, not merely compliance at the point of deployment. Immutable, timestamped evidence generated at execution time is the foundation of this capability.

5. Design Principles for Embedded Enforcement

Based on the analysis above, this paper proposes six design principles for embedding runtime enforcement into the AI Governance Stack. These principles are intentionally framed at the governance level rather than the implementation level, consistent with the Stack's principles-based approach.

> **Principle 1: Enforcement must be distributed, not centralized.**
> No single component or layer should bear sole responsibility for runtime constraint. Each layer must contribute the enforcement checks it is best positioned to evaluate.

Principle 2: Authorization must be continuous, not point-in-time.
System authorization at deployment is necessary but insufficient. The conditions that justified deployment must be continuously verified throughout operation, and violations must trigger automated constraint.

Principle 3: Evidence must be contemporaneous, not reconstructed.
The justification for any consequential action must be captured at the moment of execution, creating an immutable record that supports both internal governance and external regulatory review.

Principle 4: Constraints must be proportional to consequences.
The rigor of runtime enforcement should scale with the potential impact of the action. Low-risk actions may proceed with logging alone; high-risk actions should require multi-factor authorization gates and real-time evidence generation.

Principle 5: Enforcement mechanisms must be context-specific.
Prescriptive, one-size-fits-all enforcement architectures will not serve the diversity of AI applications, industries, and regulatory environments. Organizations must implement enforcement patterns appropriate to their specific risk profiles, system architectures, and regulatory obligations.

Principle 6: Governance must degrade gracefully.
When enforcement mechanisms fail or become unavailable, systems must default to more restrictive behavior, not less. The failure mode of a governance system should be constraint, not permission.

6. Regulatory Alignment

The shift toward runtime enforcement is not occurring in a regulatory vacuum. Several emerging regulatory frameworks either explicitly require or strongly imply continuous governance capabilities that align with the cross-cutting enforcement model described in this paper.

The EU AI Act requires providers of high-risk AI systems to implement risk management systems that operate throughout the system's lifecycle (Article 9), technical documentation that reflects the system's current state (Article 11), and post-market monitoring systems that actively collect and analyze data on system performance (Article 72). These requirements contemplate governance that is continuous and embedded, not periodic and external.

The NIST AI Risk Management Framework similarly emphasizes the GOVERN and MAP functions as ongoing processes that inform the MEASURE and MANAGE functions throughout an AI system's lifecycle. The framework's emphasis on organizational context, risk tolerance, and continuous monitoring aligns naturally with a distributed enforcement model.

A critical challenge remains, however: there is not yet a unifying federal AI policy in the United States that would standardize enforcement requirements across industries. In the absence of such policy, organizations are creating internal governance frameworks that vary widely in their treatment of runtime enforcement. This paper's principles-based approach is designed to accommodate that diversity while pointing toward a common architectural direction.

7. The Path Forward

The transition from governance-as-guidance to governance-as-enforcement is not a problem that will be solved by a single architectural innovation or regulatory mandate. It will emerge through the convergence of several developments:

First, the maturation of policy-as-code tooling that allows governance rules to be expressed in machine-executable formats, evaluated at runtime, and audited automatically. This is the technical foundation that makes distributed enforcement feasible at scale.

Second, the development of industry-specific enforcement standards that translate general governance principles into concrete requirements for particular sectors. Financial services, healthcare, and critical infrastructure will likely lead this development due to their existing regulatory density.

Third, the evolution of regulatory frameworks from compliance-at-deployment to compliance-in-operation. The EU AI Act's post-market monitoring requirements represent an early step in this direction, and future regulatory guidance is likely to become more specific about runtime governance expectations.

Fourth, organizational maturity in governance infrastructure. Most organizations are still building basic monitoring and audit capabilities. Runtime enforcement requires a foundation of instrumentation, observability, and governance tooling that many organizations have not yet established. This is a practical reality that any enforcement framework must acknowledge.

8. Conclusion

The question of where runtime enforcement lives in the AI Governance Stack reflects a genuine and important evolution in the field. As AI systems become more autonomous, the ability to constrain behavior at the moment of execution becomes essential, not as a supplement to governance, but as a core governance function.

This paper has argued that runtime enforcement is best understood as a cross-cutting capability rather than a discrete layer. Enforcement depends on data currency, model validity, system architecture, monitoring responsiveness, and evidence generation. No single layer can evaluate all of these conditions independently. The six design principles proposed here provide a framework for organizations to begin embedding enforcement across their governance architecture in a way that is proportional to risk, appropriate to context, and aligned with emerging regulatory expectations.

The AI Governance Stack was designed to bridge the gap between governance theory and operational practice. Runtime enforcement is the next bridge to build: from governance as structure to governance as active constraint. The field is not yet there uniformly, but the direction is clear, and the foundations are already in place.

About the Author

Noah M. Kenney is the Founder and Principal Consultant at Digital 520, where he advises organizations on AI governance, privacy, and security implementation. He is the author of *Governing Intelligence: Law, Privacy, Security, and Compliance in the Age of Artificial Intelligence* (2026), a freely available textbook covering the legal, technical, and organizational dimensions of AI governance. He holds a Master of Engineering degree and the CIPM certification from the International Association of Privacy Professionals.

Citation: Kenney, N.M. (2026). Runtime Enforcement and the AI Governance Stack: Embedding Execution Constraint Across the Governance Lifecycle. Digital 520.

Contact: digital520.com