Runtime Governance vs. Development Governance:

> Why Runtime Interception Is Not Decision Behavior Governance

Author: Spark Tsai
ORCID: https://orcid.org/0009-0006-8847-4703
Email: spark.tsai@gmail.com
Date: January 2026

Abstract

As large language models (LLMs) increasingly participate in organizational decision-making, contemporary AI governance practices have gravitated toward deployment-stage controls and runtime interception mechanisms. While these practices are operationally necessary, they are frequently misconstrued as sufficient indicators of effective governance.

This paper introduces a structural distinction between **Development Governance** and **Runtime Governance**, and further refines this distinction through the concepts of **Governance Existence** and **Governance Invocation**. We argue that runtime interception—absent pre-established decision constraints—cannot constitute governance over AI decision behavior.

We do not reject deployment or runtime governance. Rather, we clarify their governance scope and architectural limitations. Deployment and runtime mechanisms govern *operational conditions* and *execution environments*, but they do not govern the **decision formation process** of AI agents.

We define **Decision Behavior Governance (DBG)** as a governance paradigm that targets the structure by which AI agents form decisions *prior to execution*. By reframing governance as an ex-ante institutional condition rather than an ex-post reactive event, this paper explains why many existing governance approaches remain administratively valid yet behaviorally ineffective—particularly in probabilistic, black-box model environments.

Keywords

AI Governance, Decision Behavior Governance (DBG), Governance Existence, Governance Invocation, Development Governance, Runtime Governance, Decision Formation, ISO/IEC 42001

Introduction: Governance in the Age of Probabilistic Agents

Large language models do not execute decisions deterministically; they *sample* decisions from probabilistic distributions. This fundamental property disrupts traditional assumptions of software governance, which presume that behavior can be controlled through system configuration, deployment oversight, or runtime supervision.

In response, many organizations have expanded deployment-stage controls and runtime guardrails—monitoring outputs, intercepting sensitive responses, and enforcing policy filters. These mechanisms are

valuable. However, their increasing prominence has obscured a deeper governance question:

> **Was the AI agent's decision formed within an attributable governance structure, or merely filtered after the fact?**

This paper argues that conflating runtime interception with governance constitutes a categorical error.

Development Governance vs. Runtime Governance

Contemporary AI governance discourse frequently emphasizes **runtime governance** mechanisms, including output filtering, real-time monitoring, policy enforcement, and execution-time interception. These mechanisms are indispensable for managing operational risks in deployed systems. However, their prominence has led to a conceptual conflation between *governing execution* and *governing decision behavior*.

This section clarifies the structural distinction between **Development Governance** and **Runtime Governance**, not as competing approaches, but as governance mechanisms operating at fundamentally different stages of the decision lifecycle.

Runtime Governance

Runtime governance operates **after** a decision has been formed by an AI agent and entered the execution loop.

Its primary functions include:

- Observing generated behavior during execution
- Intercepting or suppressing outputs that violate predefined policies
- Enforcing environmental or contextual restrictions
- Recording logs for audit, monitoring, or incident response

Runtime governance is inherently **reactive** and **event-driven**. It responds to manifestations of behavior rather than shaping the internal structure by which decisions are formed.

From a probabilistic perspective, runtime governance acts as a **filter applied to sampled outputs**. The absence of a triggered interception indicates only that a specific execution path did not cross a detectable threshold. It does not constitute evidence that the underlying decision process was structurally governed.

Runtime governance therefore governs **outcomes and environments**, not the **decision formation process** itself.

Development Governance

Development governance operates **prior to execution**, at the stage where the decision space is defined, constrained, and institutionally authorized.

Its primary functions include:

- Defining which decision premises are admissible
- Establishing explicit, versioned constraints

- Structuring priority, override, and exclusion rules
- Determining the legitimacy conditions under which decisions may be formed

Development governance is **ex-ante**, **structural**, and **deterministic** in nature. It does not react to behavior; it defines the conditions under which behavior is allowed to exist.

In probabilistic AI systems, statistical uncertainty persists throughout the lifecycle. Consequently, governance cannot be relegated to runtime alone. Development governance exists to **bound the decision space before sampling occurs**, thereby reducing systemic risk that execution-stage controls—by their reactive nature—cannot reverse.

Development governance governs the **decision structure**, not the execution environment.

Governance Existence vs. Governance Invocation

The distinction between development and runtime governance can be further formalized through two governance concepts: **Governance Existence** and **Governance Invocation**.

Governance Existence

Governance Existence refers to the institutional fact that a decision space is *pre-bounded* by defined, versioned, and authoritative constraints *before* execution occurs.

- **Nature:** Ex-ante, structural, deterministic
- **Function:** Defines the admissible decision space
- **Analogy:** Architectural boundaries or safety rails that exist regardless of whether they are tested

Governance exists once these boundaries are formally established—even if a particular decision instance never triggers enforcement.

Governance Invocation

Governance Invocation refers to reactive interventions during execution, such as output filtering, runtime blocking, or policy-triggered interception.

- **Nature:** Ex-post, event-driven, probabilistic
- **Function:** Responds to detected behaviors
- **Analogy:** Law enforcement responding to observed violations

Invocation presupposes existence; invocation alone does not establish governance.

The Probabilistic Limit of Runtime-Centric Governance

LLM behavior is generated through probabilistic sampling. Runtime interception therefore constitutes a probabilistic filter acting upon a probabilistic generator.

In such a system:

- “No interception occurred” is a **statistical outcome**, not a governance guarantee.
- Absence of violation cannot be equated with presence of control.

From an engineering or legal standpoint, The application of a stochastic filter upon a stochastic generator does not yield a deterministic guarantee. does not establish determinism, accountability, or attributable governance.

Interception is a form of **event suppression**, not **behavior correction**. Once a decision is formed, runtime governance cannot alter the system's internal decision structure that produced it.

Decision Behavior Governance (DBG): A Distinct Governance Target

What DBG Governs

Decision Behavior Governance (DBG) governs the *formation* of decisions—not their outputs, environments, or side effects.

Specifically, DBG targets:

- Which premises may be used
- Which constraints apply
- How priorities and overrides are structured
- How decision legitimacy is established *before execution*

DBG does not optimize reasoning quality, nor does it police runtime behavior. It defines the **institutional conditions under which decision-making is allowed to occur**.

How DBG Differs from Existing Governance Approaches

Model Governance
Governs model capability at the pre-training stage; it is not organization-specific.

Deployment Governance
Governs configuration and exposure prior to execution; it cannot alter internal decision logic.

Runtime Governance
Governs observed behavior after decision formation; it is reactive and probabilistic in nature.

Decision Behavior Governance (DBG)
Governs decision formation *ex-ante*; it is structural and attributable.

DBG introduces governance *before* the execution loop begins.

Software Engineering Analogy: Shift-Left Governance

In software engineering, modern practice does not replace static typing or compile-time checks with production log analysis.

Runtime errors are evidence of failure—not substitutes for correctness.

Similarly, DBG argues that governance must be **shifted left** to the decision formation stage. Runtime logs may record incidents, but they do not constitute governance existence.

Implications for ISO/IEC 42001

Under ISO/IEC 42001, organizations are required to demonstrate accountable AI governance. This paper proposes a clarification:

- Runtime interception logs demonstrate **governance invocation**
- Only ex-ante decision constraints demonstrate **governance existence**

High-quality compliance requires evidence that every execution instance is traceable to a pre-defined governance boundary—not merely that violations were sometimes blocked.

From Statistical Outcomes to Institutional Facts

"Runtime interception did not trigger" is a **statistical fact**.
"Decision behavior occurred within a defined governance regime" is an **institutional fact**.

Only the latter supports accountability, auditability, and legal attribution.

DBG provides the missing structural layer required to transform AI governance from outcome monitoring into institutional control.

Conclusion

Development governance and runtime governance are both necessary components of AI system governance—but they govern different things. Development governance establishes **governance existence** by defining ex-ante decision boundaries, while runtime governance performs **governance invocation** by reacting to specific execution events.

Runtime governance without development governance reduces governance to probabilistic damage control. **Conversely, development governance without runtime governance lacks operational enforcement. Effective AI governance requires both, properly situated, clearly distinguished, and institutionally aligned. They do not replace one another; they complete different parts of the governance problem.**

Decision Behavior Governance defines the conditions under which AI decisions are allowed to exist. In probabilistic AI systems, governance cannot be an afterthought. It must be an architectural precondition.