LangGuard.AI

Agent Run-time Governance

Why Enterprises Must Adopt Deterministic Run-time Enforcement for Trust, Safety, and Compliance in Agentic AI

Executive Summary

Enterprises are entering a new phase of agent adoption. What began as experimentation with chat interfaces has evolved into agentic workflows that reason, plan, access enterprise data, and execute workflows autonomously. As agents are embedded into business workflows, applications increasingly behave autonomously, selecting tools, chaining actions, and operating across production systems without a human push behind any of it.

To support this shift, enterprises are deploying a new layer of AI infrastructure: model gateways, agent platforms, orchestration layers, and MCP servers. But while AI infrastructure is evolving rapidly, the governance response has not kept pace. Most enterprises have defaulted to one of two inadequate postures: blocking AI by default, which destroys productivity, or deploying detect-and-alert monitoring, which produces visibility without authority.

The AI governance gap is not a visibility problem. It is an enforcement problem.

Detect-and-Response (DR) approaches tell you what happened after an agent acted. Deterministic run-time enforcement stops the action before it reaches a production system.

One produces reports. The other produces trust.

Recent incidents have made this distinction operationally urgent. Agents with access to production systems such as databases, transactional platforms, customer records, code repositories have taken destructive actions that no guardrail, monitoring tool, nor LLM judge intercepted in time. For example, PocketOS incident where a production database was deleted in nine seconds by a maintenance agent is not an edge case. It is the pattern.

This white paper makes three arguments:

- AI Detect-and-Response (AIDR) approaches including shadow AI discovery, monitoring, LLM judges, and guardian agents cannot deliver trust, safety, nor compliance for agentic workflow adoption. They govern the wrong moment. Need run-time governance to confidently run agentic workflows in production.
- Deterministic run-time enforcement applied post-reasoning and pre-execution, at the action surface is the only governance model that works for agents. It governs what reasoning produces, making it immune to adversarial manipulation and capable of producing trusted governance evidence.

- The new AI infrastructure enterprises are already deploying including AI gateways, MCP gateways, agent harnesses should be leveraged for run-time enforcement points, not the traditional gateways on the edge. For example, LangGuard's Arbiter©, powered by the GRAIL context graph, operationalizes this run-time governance architecture.

An agent run-time governance that enforces deterministically at the action surface gives IT, data & AI teams, security, compliance, internal controls, and business leaders the trusted autonomy they need to accelerate AI adoption without accepting unacceptable risk.

1. What Is AI Runtime Governance?

An AI Run-time Governance is a centralized decision and enforcement layer that operates across AI infrastructure to provide visibility, policy enforcement, and auditability for AI systems without rewriting applications or intercepting execution by default.

A Run-time Governance layer enforces:

- **Who:** AI acts as (identity and credentials)
- **What:** AI can access (data, tools, models)
- **Why:** AI is acting (intent and task context)
- **How:** AI behavior aligns with enterprise policy
- **When:** AI actions are authorized versus escalated to human decision

Unlike point security products that inspect packets, prompts, or endpoints in isolation, an AI Run-time Governance enriches those signals with deep context about AI identity, intent, data access, and behavior, so that governance decisions are made against ground truth, not surface observations.

Critically, an AI Run-time Governance is not a monitoring layer. It is a run-time authority layer. A monitoring layer observes and reports. An authority layer evaluates and enforces - deterministically, at the moment an agent decides to act, before any production system is touched.

2. Why Enterprises Need an AI Run-time Governance

Enterprise AI introduces a structural fragmentation problem that cannot be solved by existing point security and IT tools.

- **AI is fragmented:** across models, agent platforms, embedded AI features, and external services, each with different execution paths and behaviors at runtime.
- **Data is fragmented:** across data lakes, SaaS applications, APIs, MCP servers, and user environments, pulled dynamically at runtime by agents reasoning about what they need.
- **Governance is anchored in the past:** Legacy Systems of Record (IAM, CMDB, SIEM, DLP, endpoint security) were built for human-driven workflows. They have no concept of an agent session, an action sequence, or a post-reasoning enforcement point.

2.1 The Rise of New AI Infrastructure

To operationalize AI at scale, enterprises are deploying a new layer of AI infrastructure, including:

- Multiple foundation and fine-tuned models
- Agent frameworks and orchestration platforms
- AI gateways for model access, routing, and cost control
- MCP gateways exposing enterprise tools and data to agents
- Embedded AI across analytics, collaboration, and SaaS platforms

This infrastructure is dynamic and autonomous by design. Agents select models, tools, and data at runtime. They do not follow pre-defined scripts. They reason toward goals and that reasoning can produce actions no guardrails written in advance could fully anticipate.

The same infrastructure that makes agents powerful (AI gateways, MCP gateways, harnesses) are also the most natural enforcement points for agent action enforcement. LangGuard's recommendation is to use this infrastructure as it was intended: not just as a routing layer, but as the enforcement boundary where every agent action is evaluated before it proceeds.

2.2 Enterprise Policy and the Governance Gap

While AI infrastructure is evolving rapidly, enterprise policy remains anchored in legacy Systems of Record:

- Identity and access management systems
- Service catalogs and CMDBs
- Cloud and endpoint security platforms
- Data classification and governance tools
- Incident response and audit systems

These systems were designed to govern human-driven workflows. They evaluate permissions, not action sequences. They record what a user was authorized to access, not what an agent decided to do with that access. Without a unifying enforcement layer, the seams between AI, data, identity, and policy are left ungoverned.

3. Why Detect-and-Alert Is Not Governance

The default enterprise response to AI risk has been to extend the conventional security frameworks: deploy shadow AI discovery, add LLM output monitoring, route agent traffic through a CASB or DLP proxy, and build alerting pipelines to SIEM. This posture has intuitive appeal as it is familiar, it requires no changes to agent execution, and it produces dashboards.

It does not produce governance. And for agentic AI, the gap between visibility and run-time authority is the gap between knowing what happened and being able to prevent it.

> *“A monitoring layer tells you an agent deleted the database. A deterministic run-time enforcement stops the agent before it does.”*
>
> **— LangGuard**

3.1 The Structural Failures of Detect-and-Alert

Why Detect-and-Alert Approaches Cannot Govern Agentic AI	
Shadow AI Discovery	Identifies unsanctioned AI tools and usage patterns. Does not evaluate what those tools do when agents invoke them at runtime. Discovery is inventory. Inventory is not enforcement.
Post-hoc Monitoring	Detects violations after execution. The database is already deleted. The customer data is already exfiltrated. The transaction is already committed. Monitoring is forensic. Authority is pre-execution. These are not substitutes.
LLM Output Monitoring	Evaluates model outputs for policy violations. Does not intercept the actions those outputs instruct agents to take. An agent that produces a clean, flagged output and then executes a destructive API call is invisible to output monitoring.
CASB / DLP Proxies	Govern data movement at the network layer. Do not evaluate agent action sequences, SoD violations, or the combination of individually permissible actions that produces an unauthorized outcome. Context-blind by design.
SIEM Alerting	Aggregates signals after the fact. Mean time to detection is measured in minutes to hours. Agent actions that cause damage are measured in seconds. The asymmetry is structural, not configurational.
Acceptable Use Policies	Define expected behavior. Do not enforce it. An agent does not read policy documents. It reasons past them.

4. Why Deterministic Run-time Enforcement Is the Only Model That Works

The case for post-reasoning, pre-execution enforcement rests on a single structural insight: agents are non-deterministic in their reasoning and deterministic in their actions. You cannot constrain the reasoning without crippling the agent. You can and must govern what the reasoning produces.

4.1 The Reasoning Step Must Be Free

The value of agentic AI is precisely its ability to reason autonomously toward a goal to select tools, evaluate options, handle obstacles, and form a plan without explicit human instruction at every step. Any governance approach that intervenes inside the reasoning process degrades this value.

There are three common interventions that do exactly this:

Why Mid-Reasoning Interventions Undermine Agent Governance	
System Prompt Constraints	Shape the inputs to reasoning. The agent acknowledges them, reasons around obstacles, and — as PocketOS demonstrated — violates every stated rule when its goal demands it. Advisory input is not enforcement.
LLM Judges	Use a second language model to evaluate the primary agent's reasoning or outputs. They introduce latency, burn additional tokens on every turn, add a second non-deterministic layer (the judge itself can be manipulated), and still do not intercept the action before it reaches a production system. Expensive and insufficient.
Guardian Agents	Deploy a monitoring agent alongside the primary agent. Same problems as LLM judges, compounded: an additional reasoning layer that can itself be manipulated, that consumes significant compute, and that governs the wrong moment. Two non-deterministic agents do not produce one deterministic outcome.

The reasoning step is where the agent does its job. The governance layer should not be inside it. It should be after it — at the action surface, evaluating what the reasoning decided to do, against the full context of what the agent has already done.

4.2 Three Properties Only Post-Reasoning Enforcement Can Satisfy

1. It governs the output of reasoning, not the reasoning itself

Pre-reasoning controls shape inputs. They cannot govern what the agent decides to do. Arbiter© operates on the output — the action the agent has decided to take — evaluated against the full context GRAIL holds for that agent and session. This is the only control layer that is not upstream of the decision, and therefore the only one that cannot be bypassed by reasoning.

2. It is immune to adversarial instruction attacks

The Lethal Trifecta: private data access, untrusted content exposure, and external communication capability co-occurring in a single agent session is catastrophically exploitable. A malicious instruction embedded anywhere in the data an agent reads can redirect its reasoning entirely, past every system prompt, every guardrail, every safety rule.

LangGuard Arbiter© enforces at the post-reasoning, pre-execution layer. GRAIL's session context makes the Lethal Trifecta pattern detectable regardless of how the agent's reasoning was manipulated to produce it. No adversarial instruction changes what LangGuard Arbiter© evaluates: the action produced and the full context in which it was produced.

3. It is the only layer that produces trusted governance evidence

Governance is not just prevention, it is proof. In the human era, accountability was personal: a named individual took a named action. Agents are ephemeral. The agent that deleted the PocketOS database no longer exists in any recoverable form. GRAIL holds the immutable, execution-time record of what that agent did across every system it touched. Arbiter© produces the policy ledger on top of that record — documenting what was authorized, by whom, against what compliance intent, at the moment of enforcement.

Post-hoc logs reconstruct what happened. GRAIL and Arbiter© record it as it happens. Auditors and boards will eventually know the difference and regulators will eventually require the latter.

5. Recommended Run-time Governance Architecture

The following diagram illustrates a recommended governance architecture for enterprise agentic AI. The key structural principle: the reasoning step is a free zone. Run-time Governance operates before it (pre-reasoning controls) and after it (post-reasoning enforcement). The enforcement outcome is always deterministic.

FIGURE 1: Run-time Governance Architecture

PRE-REASONING CONTROLS	Input screening · Identity & credential verification · Tool authorization (SCOPE-MCP risk tier: LOW / MEDIUM / HIGH / CRITICAL) · Budget & token controls · Context injection policy · Lethal Trifecta pre-check
▼ Agent receives task ▼	
REASONING (FREE ZONE)	The agent reasons freely toward its goal. No LLM judge. No guardian agent. No token-burning interception. The agent selects tools, forms a plan, and decides to act. This is where agent value is created.
▼ Agent issues action ▼	
POST-REASONING ENFORCEMENT	Arbiter© intercepts every action before it reaches a production system. Evaluated deterministically against GRAIL context graph (full session history, SoD flags, action sequence). Policy-driven, not heuristic.
Deterministic Outcome	✓ **ALLOW** × **BLOCK** ⚠ **ESCALATE → Human**
▼ Every decision recorded in GRAIL Policy Ledger ▼	

ENFORCEMENT INFRASTRUCTURE	AI Gateways (Databricks Unity AI Gateway · OpenRouter · LiteLLM) · MCP Gateways · Agent Harnesses (Claude Code · Cursor · Hermes) · Collaboration platforms (Slack · Jira) all operating as run-time policy enforcement points

5.1 Pre-Reasoning Controls

Before an agent begins reasoning toward a goal, Agent Run-time Governance applies a set of deterministic controls at the point of entry including the AI gateway or MCP gateway:

- **Input screening:** Evaluate the task, context, and any injected content for known adversarial patterns before they reach the reasoning layer.
- **Identity and credential verification:** Confirm the agent's identity, its delegated authority, and the credentials it carries into the session.
- **Tool authorization:** SCOPE-MCP pre-classifies 250+ MCP servers by risk tier (LOW, MEDIUM, HIGH, CRITICAL). Agents are only permitted to invoke tools within their authorized scope before reasoning begins.
- **Budget and token controls:** Define the operational envelope such as token limits, API call budgets, session duration, before the agent incurs any cost or exposure.
- **Context injection policy:** Control what context, data, and history the agent carries into the reasoning step.

These controls reduce the attack surface before reasoning begins. They do not govern reasoning itself. They are necessary but not sufficient.

5.2 The Reasoning Step: Free Zone

Once pre-reasoning controls are satisfied, the agent reasons freely. This is intentional and essential. The value of agentic AI is its ability to reason autonomously. LLM judges and guardian agents that interpose on the reasoning step consume tokens without producing deterministic outcomes. LangGuard does not intervene in the reasoning step.

5.3 Post-Reasoning Enforcement: Example: LangGuard Arbiter©

Every action the agent decides to take — every tool call, API invocation, database query, file write, or external communication — passes through Arbiter© before it reaches any production system. Arbiter© evaluates the action against the full session context held in GRAIL. The outcome is always one of three deterministic states:

ALLOW	BLOCK	ESCALATE
Action is within authorized scope. No policy conflict. Session context is clean. Action proceeds and is recorded in the policy ledger.	Action is classified CRITICAL or DESTRUCTIVE or violates policy. Stopped before reaching any target system.	Action is HIGH risk, SoD flag present, Lethal Trifecta active, or exception threshold crossed. Session paused.

	Reason recorded. No rollback needed.	Human notified with full context.

Every decision (ALLOW, BLOCK, or ESCALATE) is recorded in the GRAIL policy ledger as it happens. This is not a log reconstructed after execution. It is an execution-time evidentiary record, produced automatically as a byproduct of enforcement.

6. AI Gateways and MCP Gateways as Enforcement Points

The most important architectural recommendation in this white paper is this: the AI infrastructure enterprises are already deploying AI infrastructure that should serve as the enforcement boundary for agent governance.

AI gateways like Databricks Unity AI Gateway, OpenRouter, LiteLLM already sit in the critical path between agent harnesses and the models and tools they invoke. MCP gateways already mediate access between agents and the enterprise systems they act upon. These are not just routing layers. They are natural enforcement points — the precise moment in the execution stack where every agent action passes through a governed boundary before reaching a production system.

LangGuard's Architectural Recommendation

Deploy Arbiter© as the enforcement layer within AI gateways and MCP gateways — not as a monitoring sidecar, not as a proxy after the fact, but as the enforcement engine embedded in the critical path.

This approach requires no changes to agent harness code. It does not intercept or degrade reasoning. It operates at the natural boundary where agent actions become system actions — the only moment where deterministic enforcement is both possible and sufficient.

GRAIL, powered by Databricks Lakebase and integrated with Unity Catalog and Unity AI Gateway, maintains the context graph that makes Arbiter©'s enforcement authoritative. Every enforcement points feeds GRAIL. GRAIL feeds every enforcement decision. The result is a closed-loop governance system that sees everything, acts deterministically, and produces a complete evidentiary record.

6.1 Supported Enforcement Points

LangGuard Arbiter© Deployment Surface	
AI Gateways	Databricks Unity AI Gateway · OpenRouter · LiteLLM — enforcement at the model invocation layer
MCP Gateways	Enterprise MCP servers exposing tools, data, and APIs — enforcement at the tool invocation layer

Agent Harnesses	Claude Code · Cursor · Hermes · internally built harnesses — enforcement at the action execution layer
Collaboration Platforms	Slack · Jira — enforcement for embedded agents activated by default within SaaS platforms

6.2 Why Gateway-Native Enforcement Matters

Adding explicit governance logic to individual agent harnesses or application code does not scale. Enterprises run dozens of harnesses, hundreds of agents, and thousands of sessions. Governance logic that lives in the application layer must be maintained, versioned, and audited in every harness independently. It will drift. It will have gaps. It will be bypassed.

Gateway-native enforcement is centralized, consistent, and harness-agnostic. Every agent that reaches a governed gateway is subject to the same enforcement model, regardless of which harness invoked it, which model it uses, or which tool it is trying to call. Policy is defined once and enforced everywhere. Evidence is captured in one place. Gaps are structurally impossible.

7. What an AI Run-time Governance Does

An AI Run-time Governance provides deterministic controls from agent onboarding through action enforcement:

- **Pre-reasoning:** Discover, screen, and on-board agents. Define Scopes an translate guardrails, tool authorizations, and budget controls for agent workflows
- **Post-reasoning:** Enforce ALLOW | BLOCK | ESCLATE TO HUMAN before the agent acts on the enterprise systems. Capture immutable audit records for compliance and internal controls

For example, LangGuard operationalizes this run-time authority enabling a common, open standard for policy definition that is separated from AI execution and extensible across AI infrastructure and enterprise systems. LangGuard operationalizes Open Policy Agent (OPA) specifically for AI gateways, agent harnesses, MCP servers, and integrations with existing Systems of Record.

8. Governing AI Without Killing Productivity

AI trust and safety are non-negotiable. As enterprises adopt agentic workflows, IT and Security face a new challenge: agents act autonomously, adapt dynamically, and operate across models, tools, and data. Controls that lack context about AI intent, identity, and authority default to blunt enforcement. Blocking an agent often disables the very workflow it was designed to automate and do so without explaining why or how to remediate.

This does not improve trust and safety. It erodes them and drives workarounds like shadow AI use, manual bypasses, and ungoverned automation that is worse than the sanctioned alternative.

8.1 What Boards and Executives Actually Need

Leadership is not debating whether to deploy agentic workflows. They expect IT and Security to:

- Ensure safe agent deployment in production – not just guardrail it
- Enforce excessive agency – not just detectable after the fact
- Demonstrate compliance with evidence – to auditors, regulators, and boards
- Govern control at scale – across dozens of harnesses, hundreds of agents, and thousands of sessions

Deterministic run-time enforcement gives leadership exactly this. Not a monitoring dashboard. A run-time authority layer that prevents incidents before they happen, produces evidence automatically, and scales to the full agentic deployment without requiring manual review of every session.

9. Example: LangGuard Run-time Governance Platform

LangGuard consists of three integrated components that operate in sequence: Governance Run-time AI Links (GRAIL) establishes the context foundation, SCOPE-MCP maps the action surface, and Arbiter© enforces authority at run-time.

LangGuard Platform Components	
GRAIL Data Fabric	Governance AI Run-time Links. The context graph that powers all LangGuard enforcement. Powered by Databricks Lakebase and integrated with Unity AI Gateway, Unity Catalog, and enterprise systems of record. GRAIL maintains the continuous, structured history of agent activity, every action, every session, every system touched. Arbiter© evaluates every enforcement decision against what GRAIL holds. Without GRAIL, enforcement is reactive. With GRAIL, it is deterministic.
SCOPE-MCP	Open-source action surface classification. Pre-classifies 250+ MCP servers by compliance and operational risk tier: LOW, MEDIUM, HIGH, CRITICAL. Dynamic classification for local and private MCP servers. The action surface mapped before a single agent session runs. Available at scope-mcp.langguard.ai.
Arbiter©	Intent-based policy generation and run-time enforcement. Screens agents, generates policies from compliance intent (SOX, GDPR, SoD, operational risk), and enforces decisions at every action (ALLOW, BLOCK, or ESCALATE) drawing on GRAIL's context graph. Deployed within AI gateways, MCP gateways, agent harnesses, and collaboration platforms.

Conclusion

The enterprise AI governance question for agentic AI is whether governance happens before or after the damage. Detect-and-Response answered that question for the last decade of cybersecurity. For agentic AI, where an agent can delete a production database in nine seconds, where adversarial instructions embedded in data can redirect reasoning past every guardrail, where LLM judges and guardian agents burn tokens without producing deterministic outcomes, the answer must be different.

Deterministic, run-time enforcement at the action surface is not a feature of a mature AI governance program. It is the foundation of one.

LangGuard's recommendation is to build that enforcement layer into the AI infrastructure enterprises are already deploying AI gateways, MCP gateways, agent harnesses. The enforcement point is already there. The question is whether it is being used.

About LangGuard

LangGuard is the run-time action authority layer for enterprise agentic AI. LangGuard.AI is built for the humans accountable for what agents do: AI builders, Forward Deployed Engineers, IT, Compliance teams, and Chief Data and AI Officers. LangGuard is headquartered in Austin, TX with offices in the Bay Area and Canada.

To learn more about LangGuard, visit https://www.langguard.ai or follow us on LinkedIn