VALLIGNUS

Vallignus Runtime Governance for Autonomous AI Systems

VALLIGNUS.COM

Table of Contents

1 Executive Summary

Autonomous AI agents are rapidly moving beyond passive assistance and into direct operational roles. These systems can browse the internet, invoke tools, write and modify files, interact with APIs, and execute multi-step tasks without continuous human supervision. In doing so, they blur the traditional boundary between software and operator. Once deployed, an agent is no longer a short-lived computation but a persistent actor within an environment, capable of making decisions that have real technical, financial, and security consequences. As adoption accelerates, particularly in enterprise and government contexts, these systems are increasingly trusted with privileges that were previously reserved for human users or tightly scoped services.

Existing safety and security frameworks are not designed for this shift. Most current controls operate at the model level through prompt constraints, policy filters, or alignment techniques that attempt to influence how the system reasons. In practice, autonomous systems do not operate in static conditions. Goals evolve, context changes, tools fail, and partial information accumulates across time. When an agent is permitted to act repeatedly, small errors can compound into systemic failures. Traditional cybersecurity tools, meanwhile, treat the agent as just another application, offering little visibility into whether its behavior remains consistent with its original purpose.

This creates a critical governance gap. There is no standardized mechanism to enforce what an agent is allowed to do at runtime, independent of its internal reasoning. Developers are often forced to rely on manual supervision, informal guardrails, or ad hoc restrictions embedded in application logic. These measures are fragile and do not scale, especially in long-running or unattended deployments. As a result, organizations are increasingly exposed to situations where an autonomous system retains access long after its task has changed, exceeded, or become unsafe.

This paper proposes runtime governance as a foundational control layer for autonomous AI systems. Rather than attempting to shape model behavior, runtime governance enforces deterministic constraints on execution itself. By placing enforceable boundaries between agents and the systems they interact with, organizations can preserve the benefits of autonomy while maintaining accountability, containment, and operational control.

2 The Emerging Risk of Autonomous Execution

The rapid deployment of AI agents has introduced a new class of operational risk that does not neatly fit within existing security or governance models. Unlike traditional software, autonomous agents are not limited to predefined execution paths. They interpret objectives, select tools dynamically, and operate across extended time horizons. In many environments, agents retain access to file systems, credentials, network resources, and internal services while continuously adapting their behavior based on evolving context. This creates a scenario in which authority is granted once but exercised repeatedly, often without meaningful reassessment of whether that authority remains appropriate.

In practice, failures rarely occur through a single catastrophic action. Instead, risk accumulates through compounding behaviors: repeated retries, escalating tool usage, silent permission drift, and uncontrolled interaction with external systems. An agent may begin with a benign objective but encounter ambiguous states that cause it to explore alternative strategies, trigger unexpected integrations, or persist in loops that generate unintended side effects. Because these actions are technically valid from the system’s perspective, they frequently bypass traditional safeguards. Logging may record what occurred, but it does not prevent continuation. Alerting may surface anomalies, but only after impact has already begun.

For example, an agent tasked with ‘summarizing internal incident reports’ may retain access to sensitive repositories long after the task completes. If its context is later perturbed or its objective drifts, that persistent access becomes a pathway to unintended disclosure or destructive actions. Under runtime governance, execution authority can be time-bounded and scope-limited so that access expires automatically unless explicitly renewed.

This shift transforms the core problem from intent alignment to execution control. The central question is no longer whether an agent understands what it should do, but whether it should be allowed to continue doing anything at all.

2.1 Autonomous Agents as Synthetic Insider Threats

Autonomous agents effectively operate as persistent, credentialed actors within internal systems. Unlike traditional software processes, they are capable of initiating novel actions, selecting tools dynamically, and executing decisions over extended time horizons without direct human oversight.

In this context, an autonomous agent functions as a synthetic insider. It possesses legitimate access, operates under valid credentials, and performs actions that may appear operationally normal when evaluated in isolation. When an agent's context is compromised through prompt injection, data poisoning, or objective drift, its behavior can diverge from organizational intent while remaining technically authorized.

Traditional security controls such as data loss prevention, network monitoring, and access logging are poorly suited to this threat model. These systems are designed to detect anomalous external behavior, not misuse of valid internal authority. As a result, an autonomous agent may exfiltrate data, misuse credentials, or propagate unintended actions while appearing indistinguishable from a legitimate automated workflow.

This shifts the core risk from external compromise to misuse of granted authority. Effective governance must therefore focus not on detecting malicious intent, but on constraining what any autonomous system is permitted to execute, regardless of how or why the action was generated.

3 Runtime Governance as a Control Primitive

Existing approaches to AI governance primarily focus on shaping model behavior prior to execution. In this paper, runtime governance refers to an execution-level control layer that authorizes, constrains, or denies agent actions in real time, independent of model decision logic or intent formation. Techniques such as prompt constraints, policy filters, and alignment strategies are designed to influence how an agent reasons about a task. While these controls may affect intent formation, they offer limited protection once an autonomous system begins operating in a live environment. At runtime, decisions are no longer theoretical. The agent is actively invoking tools, accessing systems, modifying state, and interacting with external services. Governance mechanisms that exist only at the reasoning layer cannot enforce meaningful boundaries over these actions once execution has begun.

This limitation becomes increasingly pronounced as agents operate over extended time horizons. Autonomous systems do not execute a single isolated instruction, but rather a sequence of evolving decisions shaped by partial information, intermediate outcomes, and changing environmental context. Over time, small deviations can compound into behavior that no longer aligns with the original objective. An agent may continue acting in ways that are technically valid yet operationally unsafe, without triggering traditional guardrails that were designed for static or short-lived interactions.

Observability alone is insufficient to address this gap. While logging and monitoring provide visibility into what an agent has done, they do not prevent undesired actions from occurring in the first place. Alerts often surface only after impact has already taken place, when data has been accessed, credentials have been exercised, or external systems have been affected. In highly privileged environments, post hoc visibility does not constitute meaningful control. Effective governance requires the ability to intervene before execution proceeds, not merely to analyze outcomes afterward.

Runtime governance reframes control around the moment where authority is exercised. Rather than attempting to predict or influence reasoning, runtime governance enforces deterministic constraints on execution itself. By defining what actions are permitted, under what conditions, and for how long, organizations can establish enforceable boundaries that persist regardless of how an agent’s internal reasoning evolves. In this model, autonomy is not eliminated but bounded. Execution becomes a governed process that preserves bounded execution and operational assurance.

Runtime governance does not replace alignment techniques, secure development practices, or application-level controls. It complements them by enforcing authority at the execution boundary, where model-level and workflow-level safeguards no longer provide reliable control.

4 Core Principles of Runtime Governance

Effective runtime governance requires a shift from abstract policy definitions to enforceable operational constraints. Rather than attempting to govern intent or influence reasoning, governance must be anchored in the concrete moments where authority is exercised. This requires clearly defined principles that translate high-level risk tolerance into deterministic execution boundaries. These principles do not replace existing security or compliance frameworks but extend them into the runtime layer where autonomous behavior occurs.

At its foundation, runtime governance must treat execution as a privileged act rather than a default capability. Every action performed by an autonomous system represents a transfer of authority from the organization to the agent. Without explicit constraints, that authority persists indefinitely, even as objectives, environments, and risk conditions change. Governance, therefore, must focus on limiting not only what an agent can do, but when, under what conditions, and for how long that authority remains valid.

The first principle is **explicit authorization**. Autonomous systems should not inherit broad or implicit permissions by default. Instead, authority must be granted deliberately, scoped to specific actions, resources, and contexts. This ensures that access is purposeful rather than residual and prevents agents from accumulating capabilities simply through continued operation. Explicit authorization transforms autonomy from open-ended execution into conditional permission.

The second principle is **contextual constraint**. Actions that may be acceptable under one set of conditions may become unsafe under another. Runtime governance must incorporate environmental signals such as task state, system health, elapsed time, or external dependency changes. By tying execution rights to live context rather than static configuration, organizations can prevent agents from continuing operations when underlying assumptions are no longer valid.

The third principle is **temporal limitation**. Authority should decay unless it is actively renewed. Long-running agents frequently outlive the intent that initiated them, leading to permission drift and unintended persistence. Time-bounded execution ensures that autonomy remains aligned with current objectives and forces deliberate reauthorization as conditions evolve. In this model, continued operation becomes an explicit choice rather than an accident of uptime.

The final principle is **intervention capability**. Governance must allow systems to halt, restrict, or revoke execution before harm occurs. Observability alone is insufficient when response follows impact. Runtime governance requires the ability to intervene synchronously, preventing actions from completing when they violate defined constraints. This transforms control from reactive oversight into proactive enforcement.

Together, these principles establish a governance framework that operates at the point of execution rather than the point of design. By enforcing authorization, context, time, and intervention directly within runtime, organizations can preserve the benefits of autonomous operation while maintaining governed autonomy and execution safety.

5 Runtime Governance Architecture

Runtime governance introduces a distinct architectural control layer within autonomous AI systems. Rather than operating at the level of model reasoning or application logic, this layer is positioned directly between an agent and the systems it is authorized to access. Its role is not to influence what an agent intends to do, but to govern what it is permitted to execute in real time.

In a typical autonomous stack, intent originates from a user or mission definition and is interpreted by a model. That model's outputs are operationalized through an agent framework, which coordinates tool usage, system access, and task sequencing. Once execution begins, the agent interacts directly with files, APIs, credentials, network resources, and external services. At this stage, most existing governance mechanisms are no longer active. Authority flows downstream without continuous enforcement, and decisions are executed immediately once produced.

Runtime governance intervenes at this execution boundary. By residing between the agent and its available capabilities, the governance layer evaluates each attempted action before it occurs. This evaluation is independent of model decision logic and does not rely on semantic interpretation of intent. Instead, it enforces deterministic rules governing what actions are allowed, under what conditions, and for what duration. Execution is permitted only when those conditions are satisfied.

This architectural separation is critical. Governance logic embedded within application code or agent prompts remains tightly coupled to specific workflows and cannot persist as agent behavior evolves. In contrast, a dedicated runtime layer maintains authority regardless of how goals change, tools are selected, or reasoning paths evolve. Control is preserved even when an agent's internal state diverges from its original objective.

Vallignus implements this architecture as an execution-layer control system that operates independently of agent reasoning, enabling governance policies to be enforced consistently across models, frameworks, and long-running autonomous workflows.

5.1 Logic-Layer Governance vs Infrastructure-Layer Isolation

Traditional containment mechanisms such as containers, virtual machines, and operating system sandboxes operate at the infrastructure layer. These controls restrict access to resources, defining what files, networks, or system calls a process may access. While effective for isolating workloads, they do not govern the logic of execution itself.

Autonomous agents introduce a different class of risk. The critical failure mode is not unauthorized access to a file descriptor, but the repeated or inappropriate exercise of otherwise valid capabilities. An agent operating inside a container may remain fully compliant with infrastructure constraints while still engaging in unsafe behavior, such as excessive retries, uncontrolled external communication, or execution beyond the scope of its original objective.

Runtime governance operates at the logic layer rather than the infrastructure layer. Instead of limiting which resources exist, it governs when, why, and under what conditions authority may be exercised. This enables constraints such as limiting execution duration, preventing mission drift, restricting action frequency, or revoking authority when contextual assumptions change.

Infrastructure isolation defines where an agent may operate. Runtime governance defines how it is allowed to operate. Both are complementary, but only logic-layer governance can enforce bounded autonomy in long-running autonomous systems.

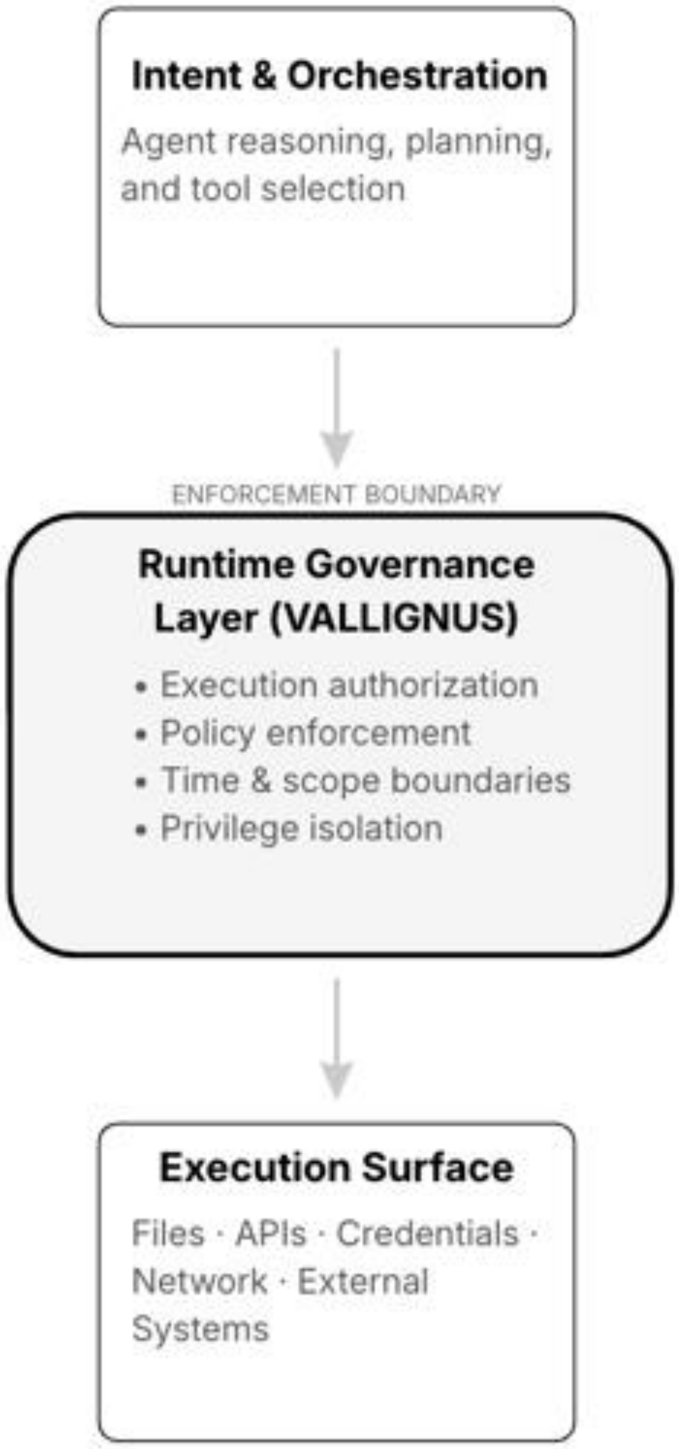
Intent & Orchestration
Agent reasoning, planning, and tool selection
ENFORCEMENT BOUNDARY
Runtime Governance Layer (VALLIGNUS)
• Execution authorization
• Policy enforcement
• Time & scope boundaries
• Privilege isolation
Execution Surface
Files · APIs · Credentials · Network · External Systems

Figure 1: Runtime Governance Control Architecture

6 Runtime Enforcement and Policy Execution

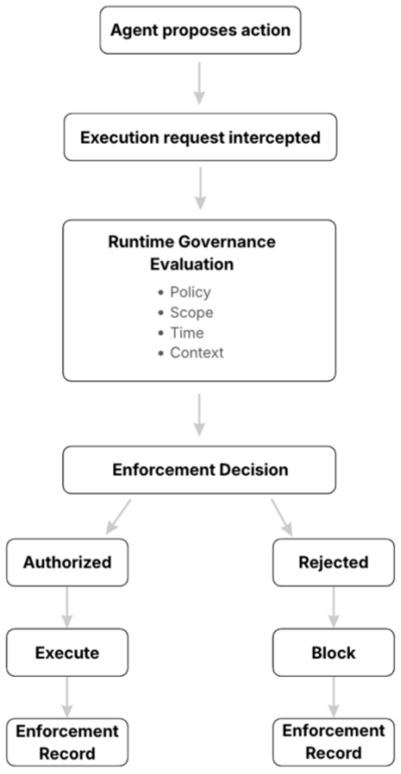

Figure 2: Runtime Execution Enforcement Flow

Runtime governance is effective only if enforcement occurs at the moment an action is attempted. Rather than evaluating outcomes after execution, the governance layer must intervene before authority is exercised. This requires a shift from observational controls to deterministic execution enforcement, where every attempted operation is evaluated against enforceable policy prior to being carried out.

In an autonomous system, execution requests originate when an agent attempts to perform an action such as invoking an API, accessing a file, using credentials, initiating a network call, or interacting with an external service. At this point, intent has already been translated into action. Runtime governance operates precisely at this boundary, intercepting execution requests before they reach the underlying system surface.

Each requested action is evaluated against a set of active runtime policies. These policies define explicit constraints including allowed actions, resource scope, execution context, time limits, escalation thresholds, and privilege boundaries. Enforcement is not probabilistic and does not rely on semantic interpretation of agent intent. Instead, policies are evaluated as deterministic rules applied uniformly across all execution attempts.

When a requested action satisfies all active constraints, execution is permitted and proceeds normally. When a request violates policy, execution is denied by default. This denial occurs synchronously and prevents the action from being initiated at the system level. As a result, unsafe or unauthorized behavior cannot occur, even if the agent's internal reasoning process determines that the action is valid or desirable.

Runtime enforcement also supports conditional and contextual evaluation. Policies may incorporate factors such as execution frequency, accumulated actions over time, environment classification, or privilege escalation state. This enables governance to account for compounding risk, where individual actions may appear safe in isolation but become unsafe when repeated or combined.

Importantly, enforcement decisions are independent of how an agent arrived at a given action. Retries, alternative reasoning paths, prompt reformulation, or model self-correction do not alter enforcement outcomes. Because governance is applied at execution rather than reasoning, agents cannot bypass constraints through adaptation or exploration. Authority remains fixed even as behavior evolves.

This enforcement model ensures that governance persists throughout the lifecycle of an autonomous system. Policies remain active across long-running sessions, unattended execution, and dynamic goal shifts. Control does not degrade over time, nor does it depend on continuous human supervision to remain effective.

By enforcing policy at the execution boundary, runtime governance establishes a reliable mechanism for maintaining operational safety, containment, and accountability. Autonomous systems retain the ability to reason, plan, and adapt, but execution remains subject to explicit authorization. In this model, autonomy is preserved, but authority is never implicit.

6.1 Common Runtime Failure Modes and Enforcement Responses

Runtime governance enables deterministic responses to common failure modes observed in autonomous systems. Rather than relying on detection after impact, enforcement occurs before execution proceeds. Examples include:

• **Infinite or excessive retry loops**
Execution is halted once predefined action or time thresholds are exceeded.

• **Unauthorized network egress**
Requests to unapproved destinations are denied prior to execution.

• **Privilege escalation attempts**
Actions exceeding the agent’s authorized scope are blocked and recorded.

• **Continued operation after task completion**
Authority expires automatically unless explicitly renewed.

• **Unexpected tool invocation or integration activation**
Execution is denied when outside approved policy conditions.

In each case, enforcement decisions are applied synchronously and recorded as part of the system’s governance audit trail. Unsafe actions are prevented by design rather than investigated after impact has occurred.

7 Auditability, Accountability, and Oversight

Autonomous systems introduce a fundamental challenge for organizational oversight. As agents operate independently across extended time horizons, traditional monitoring approaches become insufficient to establish accountability. Logging model outputs or recording execution traces after the fact does not provide meaningful assurance that authority was exercised appropriately at the time decisions were made. Effective governance requires the ability to demonstrate not only what occurred, but under what permissions, constraints, and conditions each action was allowed to proceed.

Runtime governance enables auditability by generating enforcement-level records at the point of execution. Each attempted action is evaluated against active governance policies before it occurs, producing an authoritative record of whether execution was permitted, denied, delayed, or escalated. These records reflect the actual boundaries applied to the system, independent of an agent's internal reasoning process. As a result, audit trails capture operational control rather than inferred intent, providing a verifiable account of how authority was exercised in practice.

This distinction is critical for accountability. In autonomous systems, failures often emerge not from a single catastrophic decision, but from a sequence of technically valid actions that collectively exceed acceptable risk. Post hoc inspection of reasoning or logs may explain what happened but cannot demonstrate that appropriate controls were in place beforehand. By contrast, runtime governance produces a continuous record of enforced limits, enabling organizations to reconstruct the exact governance state under which each action occurred.

Oversight in this model shifts from continuous human supervision to structured review of bounded execution. Human operators are not required to monitor agents in real time. Instead, oversight is achieved through policy definition, enforcement review, and audit analysis. Governance decisions are made explicit, inspectable, and attributable, allowing organizations to assess whether autonomous behavior remained within authorized limits throughout an agent's operational lifecycle.

This approach also supports institutional requirements for transparency and compliance. Regulatory review, internal audits, and incident investigations increasingly require evidence of active control rather than retrospective explanation. Runtime governance provides a mechanism to demonstrate that autonomous systems operated within defined authority structures, that privilege escalation was constrained, and that unsafe or unauthorized actions were technically prevented, not merely discouraged.

By grounding accountability at the execution layer, runtime governance establishes a durable foundation for oversight in autonomous systems. Control becomes demonstrable, reviewable, and enforceable, enabling organizations to deploy increasingly capable agents while preserving institutional responsibility, audit integrity, and operational trust.

8 Implications for Enterprise and Government Systems

The adoption of autonomous systems within enterprise and government environments introduces constraints that differ fundamentally from consumer or experimental use cases. These environments operate under strict requirements for accountability, access control, auditability, and risk management. Autonomous agents deployed within such contexts must function within clearly defined authority boundaries, regardless of their internal reasoning capabilities or performance optimizations.

In high-trust environments, unrestricted autonomy is rarely acceptable. Systems must be capable of demonstrating not only functional effectiveness, but also enforceable governance. This includes the ability to limit execution scope, constrain privilege escalation, and ensure that automated actions always align with organizational policies. As autonomous agents assume greater operational responsibility, the absence of runtime control mechanisms becomes a limiting factor for deployment at scale.

Runtime governance enables organizations to reconcile autonomy with institutional responsibility. By introducing deterministic enforcement at the execution layer, enterprises and government entities can allow agents to operate independently while maintaining confidence that actions remain bounded by policy. This approach reduces reliance on continuous human supervision and mitigates the operational burden associated with long-running or unattended autonomous processes.

For regulated environments, runtime governance also supports alignment with existing risk and compliance frameworks. Rather than introducing entirely new governance structures, execution-level enforcement can integrate with established access control, identity management, and audit workflows. Autonomous systems become subject to the same expectations of authorization, traceability, and review that already govern human and service-based activity.

In government and defense contexts, these considerations are amplified. Systems frequently operate across sensitive data domains, classified environments, and mission-critical infrastructure. The ability to demonstrate enforceable control over autonomous execution is essential not only for internal governance, but for interagency coordination, oversight, and assurance. Runtime governance provides a mechanism for deploying autonomous capabilities without eroding established authority structures.

As autonomous systems continue to evolve, their adoption will increasingly depend on the presence of verifiable control rather than raw capability. Organizations that can demonstrate bounded autonomy will be positioned to deploy agents more broadly, while those relying solely on intent-based safeguards will face structural limitations. Runtime governance establishes the foundation upon which autonomous systems can be trusted, scaled, and integrated into institutional operations.

9 Conclusion: Toward Governed Autonomy

As autonomous systems become increasingly capable, the question of control becomes central to their responsible deployment. Advances in reasoning, planning, and tool use have expanded what agents can accomplish, but have also amplified the risks associated with unrestricted execution. Without enforceable boundaries, autonomy scales faster than governance, creating structural limitations for adoption in environments where accountability and assurance are mandatory.

This paper has argued that effective governance cannot reside solely within model behavior or application logic. Control must be exercised at the point where authority is enacted. By introducing runtime governance as an independent enforcement layer, organizations can establish deterministic limits on execution while preserving the adaptive capabilities that make autonomous systems valuable.

Runtime governance reframes autonomy as a bounded capability rather than an open-ended permission. Execution becomes conditional, auditable, and revocable by design. This approach enables institutions to move beyond reliance on intent-based safeguards and toward verifiable control mechanisms that persist across long-running and unattended operations.

As autonomous systems continue to mature, their successful integration into enterprise and government environments will depend less on what they can do and more on how reliably they can be governed. Establishing execution-level control provides a foundation for trust, scalability, and institutional adoption. Governed autonomy represents not a restriction on intelligent systems, but a prerequisite for their sustainable use.