Sovereign Agentic Loops: Decoupling AI Reasoning from Execution in Real-World Systems

Jun He
OpenKedge.io
junhe@openkedge.io

Deying Yu
OpenKedge.io
deying@openkedge.io

Abstract

Large language model (LLM) agents increasingly issue API calls that mutate real systems, yet many current architectures pass stochastic model outputs directly to execution layers. We argue that this coupling creates a safety risk because model correctness, context awareness, and alignment cannot be assumed at execution time. We introduce **Sovereign Agentic Loops (SAL)**, a control-plane architecture in which models emit structured intents with justifications, and the control plane validates those intents against true system state and policy before execution. SAL combines an obfuscation membrane, which limits model access to identity-sensitive state, with a cryptographically linked Evidence Chain for auditability and replay. We formalize SAL and show that, under the stated assumptions, it provides policy-bounded execution, identity isolation, and deterministic replay. In an OpenKedge prototype for cloud infrastructure, SAL blocks 93% of unsafe intents at the policy layer, rejects the remaining 7% via consistency checks, prevents unsafe executions in our benchmark, and adds 12.4 ms median latency.

1 Introduction

Large language models (LLMs) have accelerated interest in autonomous *agentic systems* that reason, plan, and interact with external environments. By integrating LLMs with APIs that perform real-world actions—ranging from software deployment to infrastructure operations—these systems are moving from prototypes toward production use.

Despite this progress, many systems still rely on an architectural assumption that deserves closer scrutiny: model outputs are treated as executable commands. In these frameworks, an LLM observes a system state, generates an action payload, and that payload is dispatched with little or no mediation. This tightly coupled pipeline assumes that the reasoning process is sufficiently correct, context-aware, and aligned with system invariants at the moment of execution [1].

In practice, that assumption is unreliable. Unlike deterministic software components, frontier generative models are stochastic and often operate under partial observability or obfuscated context. Their failures are not limited to incorrect text generation; they can also produce unsafe real-world mutations. For example, an agent might issue a syntactically valid command such as `TerminateInstances` against a critical database node because it inferred the wrong operational context. Post-hoc safeguards such as logging or coarse permission boundaries may record or limit such behavior, but they do not by themselves verify whether the proposed action is appropriate under the true system state.

In classical control theory, unstable systems are not made safe by assuming ideal controller behavior; they are governed through feedback structures that constrain their effects [2, 3]. Motivated by this perspective, we argue that stochastic reasoning should be bounded at the execution boundary rather than trusted to act directly on external systems.

Preprint.

We identify *direct execution authority* as the core architectural problem. To address it, we formalize the **Decoupling Principle**: reasoning models should yield *verifiable intent* rather than *direct execution authority*. This reframes model outputs as proposals that require validation before they can affect system state.

We operationalize this principle through the **Sovereign Agentic Loop (SAL)**, a minimal-trust control-plane architecture that separates semantic reasoning from execution through a deterministic mediation pipeline. Under SAL, model generations are intercepted and evaluated as intent proposals against pre-execution policy constraints.

To connect the formal framework to a deployable system, we implement SAL within the **OpenKedge** [4] control plane. This implementation uses an *obfuscation membrane* to limit model access to identity-sensitive information and a cryptographically linked *Evidence Chain* to record states, intents, and executed actions.

Our core contributions are as follows:

- We identify the coupling of probabilistic reasoning with direct execution authority as an architectural vulnerability in contemporary agentic systems.
- We formalize the *Decoupling Principle* and introduce the **Sovereign Agentic Loop (SAL)**, a control-plane framework that evaluates model outputs as intents before execution.
- We define an *obfuscation membrane* that isolates reasoning processes from identity-sensitive state while preserving the structural information needed for decision-making.
- We present an *Evidence Chain* ledger that records intents, evaluations, and executed actions to support auditability and deterministic replay.
- We evaluate the architecture in the **OpenKedge** prototype, showing that unsafe actions can be blocked before execution with 12.4 ms median overhead in our benchmark.

Together, these contributions support a design approach in which agent safety is enforced by the execution architecture rather than delegated solely to model behavior.

2 Related Work

The problem of safely integrating autonomous agents with real-world systems intersects several research areas, including agentic tool use, runtime verification, access control, and control theory. We position Sovereign Agentic Loops (SAL) as a unifying control-plane abstraction that addresses limitations across these domains.

2.1 Agentic Tool Use and LLM-Based Systems

Recent advances in large language models (LLMs), including systems developed by OpenAI [5] and Anthropic [6], have enabled agents to perform tool use and multi-step reasoning. Frameworks such as ReAct [7], Toolformer [8], and function-calling APIs allow models to generate structured actions that interact with external systems. More recently, there has been a push towards formalized execution verification for LLM agents in production environments [9], though many implementations remain constrained to read-only observability.

A recent systematic audit—the 2025 AI Agent Index [10]—documents the technical and safety features of 30 deployed agentic systems, revealing a *transparency asymmetry*: while developers readily disclose capabilities, only four of 30 agents provide agent-specific safety evaluations, and 25 disclose no internal safety results whatsoever. This gap motivates structural execution governance.

These approaches focus primarily on improving the *reasoning capabilities* of the model or the *interface* between models and tools. However, they typically assume that once an action is generated, it can be executed directly, subject only to syntactic validation or lightweight guardrails [1].

In contrast, SAL introduces a structural decoupling between reasoning and execution. Rather than treating model outputs as executable commands, SAL treats them as *intents* that must be validated within a sovereign control plane. This shifts the problem from tool orchestration to execution governance.

2.2 Guardrails and Runtime Validation

A growing body of work focuses on guardrails for LLMs, including prompt constraints, output filtering, rule-based validation layers [11, 12], and late-stage runtime monitors [1]. These techniques aim to prevent unsafe outputs by constraining model behavior at inference time.

Recent work has advanced beyond static rule-based guardrails. AgentSpec [13] introduces a domain-specific language for defining customizable runtime safety rules—specifying triggers, conditions, and enforcement mechanisms that are evaluated during agent execution. Pro2Guard [14] extends this line of work from reactive to *proactive* enforcement, modeling agent behaviors as Discrete-Time Markov Chains (DTMCs) and using probabilistic reachability analysis to estimate the probability of reaching an unsafe state, enabling intervention before a violation materializes. Concurrently, the Open Agent Passport (OAP) [15] provides a deterministic pre-action authorization specification that synchronously intercepts tool calls, evaluates them against declarative policies, and produces cryptographically signed audit records—achieving a 0% adversarial success rate under restrictive policies with a median enforcement latency of 53 ms.

While effective for constraining specific agent behaviors, such approaches remain fundamentally *model-centric* or *action-centric*. They do not provide guarantees over the *effects* of actions once executed in external systems, nor do they establish a complete causal chain linking reasoning to execution outcomes.

SAL differs in that safety is enforced *prior to execution* through a deterministic evaluation function over real system state. The guarantees of SAL do not depend on the correctness or alignment of the underlying model, but on the enforcement properties of the control plane. While OAP shares SAL's pre-execution philosophy, SAL additionally formalizes the *intent abstraction*, information-theoretic isolation via the obfuscation membrane, and cryptographic evidence chains as first-class primitives of the control plane.

2.3 Runtime Verification and Formal Methods

Runtime verification and formal methods provide mechanisms for checking system properties during execution [16, 17]. Techniques such as temporal logic monitoring, invariant enforcement, and model checking have been widely applied to distributed and safety-critical systems.

Recent work by Doshi et al. [18] applies System-Theoretic Process Analysis (STPA) to LLM-based agents, systematically identifying hazards in agent workflows and deriving formal safety requirements for tool sequences and data flows. Their approach enhances the Model Context Protocol (MCP) with capability, confidentiality, and trust-level labels to enforce these requirements at runtime, moving from ad hoc reliability fixes toward formal safety contracts.

These approaches typically assume a known system model and operate over observable system traces. In agentic systems, however, the decision-making process is externalized to a stochastic model, and execution may be triggered without a formally verifiable decision boundary.

SAL complements runtime verification by introducing a *pre-execution decision boundary*. Instead of verifying correctness after or during execution, SAL enforces policy compliance before any state mutation occurs. The Evidence Chain also provides a complete causal trace that enables deterministic replay, extending traditional trace-based verification. Unlike STPA-derived safety contracts that require design-time hazard enumeration, SAL enforces safety invariants dynamically through sovereign policy evaluation over live system state.

2.4 Access Control and Policy Systems

Access control mechanisms such as Role-Based Access Control (RBAC) [19], Attribute-Based Access Control (ABAC) [20], and modern policy engines such as Open Policy Agent (OPA) [21] and Cedar [22] provide structured ways to regulate system access.

Capability governance extends static access control to agentic settings. Sidik and Rokach [23] introduce Aethelgard, a framework that moves beyond static container-based sandboxing toward learned, adaptive capability governance. Aethelgard uses reinforcement learning to enforce minimum viable capability sets per task, dynamically scoping tool awareness to mitigate the *capability*

overprovisioning problem—where agents are routinely exposed to all available tools regardless of task requirements.

These systems operate on the assumption that the caller identity and request context are trustworthy representations of intent. Unlike traditional policy engines, SAL does not rely on trusted caller identity or static authorization rules. Instead, it derives execution authority dynamically from validated intent, real system state, and justification consistency within the control plane. In agentic systems, however, the caller (i.e., the agent) may generate actions based on incomplete or incorrect reasoning.

SAL extends access control by shifting from *identity-based authorization* to *intent-based authorization*. Execution authority is not statically assigned but dynamically derived from validated intent, context, and policy evaluation. While Aethelgard's learned capability scoping reduces the attack surface available to the agent, SAL adds a separate constraint: even within the permitted capability set, every proposed action must pass deterministic policy evaluation before any state mutation occurs.

2.5 Multi-Agent Coordination and Failure Taxonomies

The scaling of agentic systems beyond single-agent deployments introduces emergent coordination risks that compound the safety challenges addressed by SAL. Cemri et al. [24] introduce MAST, a taxonomy categorizing multi-agent LLM system failures into 14 distinct modes across specification issues, inter-agent misalignment, and task verification failures—analyzed over 1,600 execution traces. Their findings demonstrate that uncoordinated multi-agent systems can amplify errors by up to $17\times$, while centralized architectures with validation bottlenecks contain amplification to approximately $4.4\times$.

Hierarchical oversight architectures have been proposed to address these failure modes. Kim et al. [25] present Tiered Agentic Oversight (TAO), a multi-agent framework that routes tasks to specialized agents based on complexity and risk, enabling higher tiers to oversee and correct lower-tier reasoning. Google Research [26] provides a quantitative foundation for multi-agent scaling through an evaluation of 180 agent configurations, demonstrating that multi-agent coordination can boost performance by up to 80.9% on parallelizable tasks while degrading performance by 39–70% on sequential reasoning tasks.

SAL is orthogonal to and composable with multi-agent coordination architectures. While MAST taxonomizes failures and TAO introduces hierarchical oversight, neither provides a *pre-execution invariant enforcement* layer that structurally prevents unsafe mutations regardless of the coordination topology. SAL can serve as the execution-boundary control plane for any agent within a multi-agent system, ensuring that the error amplification documented by Cemri et al. cannot propagate through actual infrastructure mutations.

2.6 Control Theory and Feedback Systems

Control theory models systems as feedback loops that regulate behavior based on observed state [3]. Classical control systems assume deterministic dynamics and well-defined control inputs.

Agentic systems violate these assumptions due to stochastic reasoning and partial observability. SAL can be interpreted as a control-theoretic extension in which the controller operates over *intent proposals* rather than direct control signals.

By introducing a closed-loop structure with evaluation, execution, and recording stages, SAL reestablishes controllability in systems where the decision-making process is otherwise opaque.

2.7 Data Privacy and Information-Theoretic Isolation

Techniques such as differential privacy [27], secure enclaves, and data anonymization aim to limit information leakage when processing sensitive data. Recent explorations into sovereign AI infrastructure [28] highlight the necessity of strict data boundaries when utilizing foreign foundation models.

SAL incorporates an information-theoretic perspective through the obfuscation membrane, which ensures that the reasoning model operates on a transformed state with no mutual information with

identity-sensitive components. Unlike traditional privacy techniques that focus on data release, SAL applies isolation to the *decision-making interface* itself.

2.8 Summary

Existing approaches address individual aspects of the problem—reasoning, validation, access control, or privacy—but do not provide a unified framework for governing autonomous execution. Recent advances in proactive guardrails [14], runtime enforcement DSLs [13], system-theoretic safety analysis [18], and pre-action authorization [15] represent important progress, yet each addresses a single facet of the problem without establishing an end-to-end execution governance architecture.

Anchored natively in the OpenKedge foundation, this work extends the OpenKedge control plane to contribute an abstraction that:

- Decouples reasoning from execution authority
- Enforces safety as a pre-execution constraint
- Provides deterministic replay through evidence chains
- Establishes information-theoretic isolation between reasoning and infrastructure state

SAL is designed to be complementary to advances in model capability, runtime enforcement, multi-agent coordination, and system verification.

3 System Model and Theoretical Foundations

3.1 System Topology and State Representation

As illustrated in Figure 1, integrating foreign frontier models with sovereign infrastructure introduces a tension: the sovereign system encapsulates sensitive, localized context that must be protected, while the foreign reasoning agent requires this context to formulate effective decisions.

We formalize the operating environment of a sovereign agentic system, as depicted in Figure 2, through a bisected state-space representation. This consists of the **true state space** $\mathcal{S}$, which contains identity-sensitive attributes, and the **obfuscated state space** $\hat{\mathcal{S}}$, a synthesized projection that preserves the structural topology required for higher-order reasoning while stripping identifying contexts.

We mathematically model any given state $s \in \mathcal{S}$ via a structural decomposition:

$$s = (s_{id}, s_{struct}) \tag{1}$$

where s_{id} encompasses the identity-sensitive components (e.g., exact IP addresses, node IDs), and s_{struct} denotes the structural properties.

Obfuscation Membrane. To enforce information-theoretic isolation, we introduce the **obfuscation membrane**, formalized as a projection mapping Π:

$$\Pi : \mathcal{S} \to \hat{\mathcal{S}} \tag{2}$$

Treating the system state as a random variable $S = (S_{id}, S_{struct})$ and defining $\hat{S} = \Pi(S)$, this projection must satisfy the strict isolation constraint:

$$I(S_{id}; \hat{S}) = 0 \tag{3}$$

where $I(\cdot;\cdot)$ denotes Shannon mutual information. This constraint implies that the identifying substate is not recoverable from the obfuscated representation under the model assumptions. In practice, this isolation holds provided that auxiliary side channels do not reintroduce identity information.

Concurrently, the obfuscation must optimize for structural preservation to maintain reasoning efficacy:

$$\max_{\Pi} \; I(S_{struct}; \hat{S}) \tag{4}$$

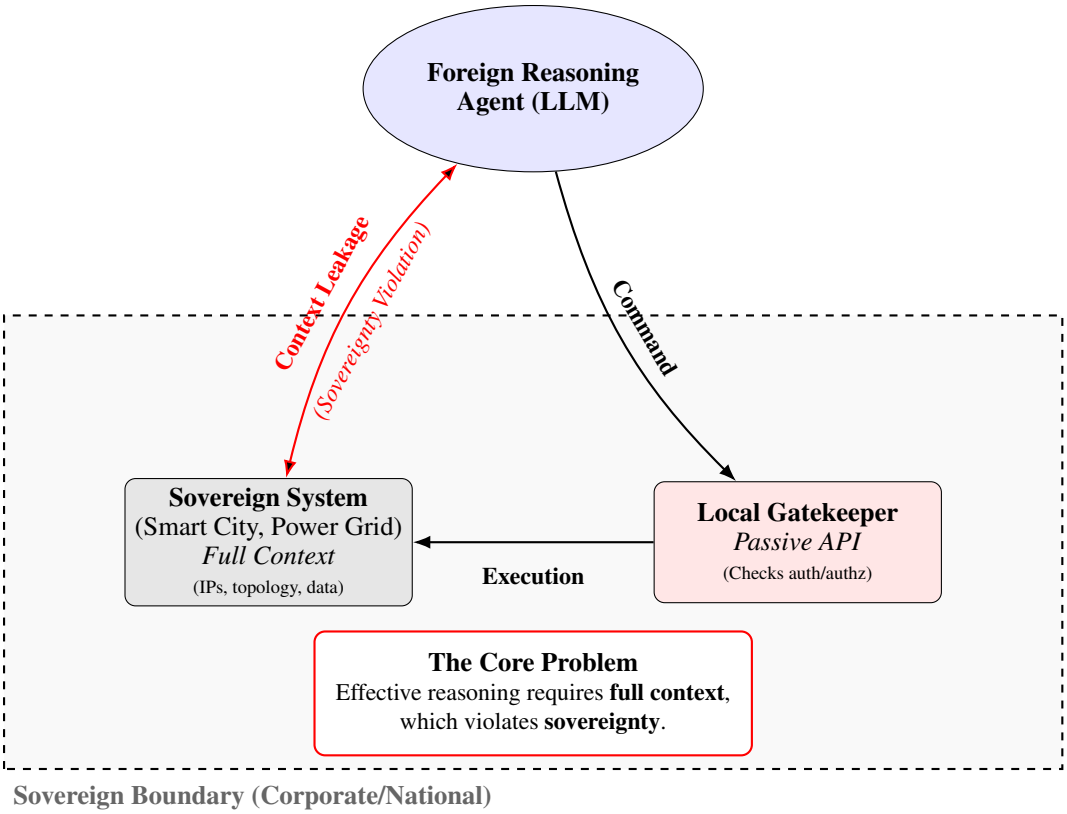

Figure 1: System topology illustrating the sovereignty conflict. A foreign reasoning agent requires full context from the sovereign system for effective decision-making, which leads to context leakage and a violation of data sovereignty.

Together, these constraints isolate identifying variables while preserving the spatial geometry needed for remote agentic reasoning.

To reverse this mapping within a secured boundary, we define a privileged reconstruction function Π^{-1}, accessible exclusively within the control plane:

$$\Pi^{-1} : \hat{\mathcal{S}} \times \mathcal{K} \to \mathcal{S} \tag{5}$$

where $\mathcal{K}$ represents a sovereign mapping key or cryptographic lookup structure.

3.2 The Sovereign Agentic Loop Formulation

Agentic Reasoning Function. Let M denote a foreign reasoning model (e.g., a frontier LLM). The model strictly receives obfuscated inputs to generate formal intent and natural language justification:

$$M(\hat{s}, G) \to (i, J) \tag{6}$$

where $\hat{s} \in \hat{\mathcal{S}}$ is the obfuscated state context, G is the goal, $i \in \mathcal{I}$ is the structured intent, and J acts as an explanatory reasoning trace. Consequently, the model's conditional generation carries zero mutual information regarding local infrastructure identity:

$$I\big(S_{id}; M(\hat{S}, G)\big) = 0 \tag{7}$$

In practice, this isolation holds under the assumption that external signals do not reintroduce identity information.

Unstable Reasoning Model. We formally define the agent M as a stochastic, *unstable decision process*. For any given state-goal pair, the probability distribution over generated intents is formally

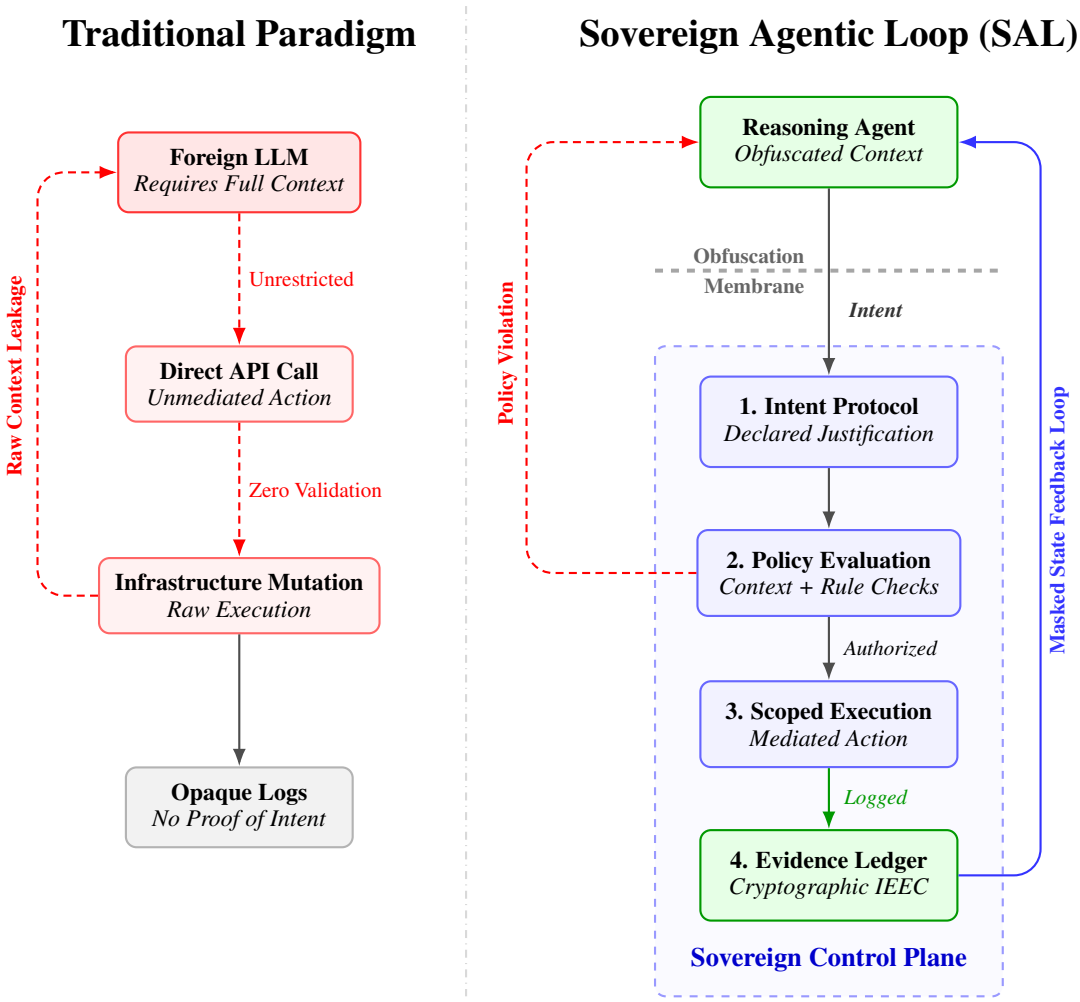

Figure 2: Comparison between traditional agent execution and the Sovereign Agentic Loop (SAL). In the traditional model (left), actions are directly executed with no validation, resulting in opaque and non-replayable behavior. In SAL (right), all actions cross the Obfuscation Membrane into a sovereign control plane, enforcing policy validation, execution scoping, and cryptographic evidence recording.

non-stationary:

$$\exists \hat{s}, G,\ i_1 \neq i_2 \quad \text{such that} \quad P\big(M(\hat{s}, G) = (i_1, J_1)\big) > 0 \ \wedge \ P\big(M(\hat{s}, G) = (i_2, J_2)\big) > 0 \tag{8}$$

This non-determinism means that identical or near-identical scenarios can yield conflicting intents. This motivates treating model outputs as proposals that require downstream validation rather than as commands that can be executed directly.

Sovereign Evaluation Function. To govern the stochastic generated intents, we define a deterministic control-plane evaluation function:

$$E : \mathcal{I} \times \mathcal{S} \times \mathcal{P} \to \{0, 1\} \tag{9}$$

where $\mathcal{P}$ represents the set of invariant sovereign policies. We logically decompose E to validate both rule-based policy compliance and contextual reasoning alignment:

$$E(i, s, \mathcal{P}) = E_{\text{policy}}(i, s, \mathcal{P}) \wedge E_{\text{consistency}}(i, J, \hat{s}) \tag{10}$$

Here, $E_{\text{policy}} : \mathcal{I} \times \mathcal{S} \times \mathcal{P} \to \{0, 1\}$ validates compliance against system constraints using the full state s, while $E_{\text{consistency}} : \mathcal{I} \times \mathcal{J} \times \hat{\mathcal{S}} \to \{0, 1\}$ checks whether the proposed intent is consistent with its justification and the perceived context $\hat{s}$.

Execution Semantics. Once an intent is authorized, the system executes it via a strictly mediated execution operator $\mathcal{X}$:

$$\mathcal{X} : \mathcal{I} \times \mathcal{S} \times \mathcal{P} \to \mathcal{A} \cup \{\perp\} \tag{11}$$

where physical execution is explicitly conditioned on the evaluation ruling:

$$\mathcal{X}(i, s, \mathcal{P}) = \begin{cases} a_{\text{exec}} & \text{if } E(i, s, \mathcal{P}) = 1 \\ \perp & \text{otherwise} \end{cases} \tag{12}$$

where the action $a_{\text{exec}} = \Pi^{-1}(i)$ uses the privileged reconstruction mapping $\mathcal{K}$ to translate the abstract intent into a concrete mutation in the true infrastructure state space.

Evidence Chain. To support auditability, we formalize the *Evidence Chain* $\mathcal{C}$ as a cryptographically linked execution ledger. At step t, the chain records:

$$\mathcal{C}_t = \langle i_t, J_t, E_t, a_{\text{exec},t}, H(c_t), H(\mathcal{C}_{t-1}) \rangle \tag{13}$$

where $a_{\text{exec},t} = \Pi^{-1}(i_t)$ (if authorized and executed), $c_t \subseteq s_t$ captures the precise context snapshot used during evaluation, and $H(\cdot)$ denotes a collision-resistant cryptographic hash function.

3.3 Formal Safety Guarantees

Assumptions. We adopt the following assumptions for the SAL model:

- **A1 (Deterministic Evaluation):** The evaluation function E produces a consistent decision for a given input $(i, s, \mathcal{P})$.
- **A2 (Deterministic Execution):** The execution operator $\mathcal{X}$ deterministically maps an approved intent to a concrete action.
- **A3 (Closed Execution):** All state mutations are mediated exclusively by $\mathcal{X}$.
- **A4 (Context Completeness):** The context c_t contains all information required to evaluate the correctness and safety of an intent.

In practice, the closed execution assumption (A3) is enforced by restricting all mutation-capable credentials to the control plane, ensuring that no external component can invoke execution APIs directly.

Lemma 1 (Non-Invertibility Without Sovereign Key). *Let $\Pi : \mathcal{S} \to \hat{\mathcal{S}}$ be the obfuscation membrane satisfying $I(S_{id}; \hat{S}) = 0$. Then, without access to the privileged mapping key $\mathcal{K}$ (or an equivalent reconstruction oracle), there exists no deterministic estimator f such that:*

$$f(\hat{S}) = S_{id} \tag{14}$$

with an accuracy exceeding that of uninformed random guessing over the prior marginal distribution of S_{id}.

Proof Sketch. Given the obfuscation constraint $I(S_{id}; \hat{S}) = 0$, the obfuscated state $\hat{S}$ shares no mutual information with the identifying attributes S_{id}. Consequently, an estimator $f(\hat{S})$ cannot improve on the marginal prior $P(S_{id})$ using only $\hat{S}$. Structural inversion therefore requires auxiliary side-channel information not contained in $\hat{S}$. In SAL, this information is confined to the privileged state-cache $\mathcal{K}$ within the control plane.

These guarantees hold strictly under the closed execution assumption (A3) and do not extend to systems with external execution pathways. □

Theorem 1 (Execution-Bound Sovereignty). *Assume a system satisfies the topological definitions established in Section 3 and operates under assumptions A1–A4. Then, for every actualized execution $a_{\text{exec},t}$, the SAL architecture structurally guarantees:*

1. **Policy-Bounded Execution:**

$$\forall a_{\text{exec},t},\ \exists i_t \text{ such that } E(i_t, s_t, \mathcal{P}) = 1 \tag{15}$$

2. **Identity Isolation:**

$$I(A_{\text{exec},t}; S_{id} \mid \hat{S}_t) = 0 \tag{16}$$

3. **Deterministic Cryptographic Replay:**

$$\text{Replay}(\mathcal{C}_t) = a_{\text{exec},t} \tag{17}$$

Proof Sketch. First, by the closed execution assumption (A3), the execution operator $\mathcal{X}$ mediates all possible state mutations. As formalized in the execution semantics, $\mathcal{X}$ yields an actionable mutation if and only if $E(i_t, s_t, \mathcal{P}) = 1$, which bounds execution to policy-approved intents.

Second, identity isolation follows from the obfuscation projection $I(S_{id}; \hat{S}_t) = 0$. Since the agent M computes intent using $\hat{S}_t$ and G, the resulting i_t is generated without direct access to identifying state. The mapping from i_t to $a_{\text{exec},t}$ occurs within the control plane through Π^{-1}.

Third, deterministic replay follows from assumptions A1 and A2. Because the Evidence Chain ledger $\mathcal{C}_t$ records i_t, c_t, E_t, and the chained hash state $H(\mathcal{C}_{t-1})$, the information required to reconstruct the executed mutation remains available for replay and audit.

Consequently, the architecture guarantees policy-bounded execution, identity isolation, and deterministic replayability, completing the proof. □

Summary. Theorem 1 shows that, under the stated assumptions, safety properties can be enforced *prior* to physical execution by the control plane rather than by relying on model behavior alone. This formulation establishes a closed-loop pipeline from obfuscated observation to intent generation, evaluation, and execution, forming the basis of execution-bound safety guarantees.

4 System Design and Implementation

We implement the Sovereign Agentic Loop (SAL) in a prototype system called **OpenKedge** [4], designed to integrate external reasoning agents with cloud infrastructure environments under control-plane mediation. OpenKedge intercepts, translates, evaluates, and governs agent-initiated mutations before they reach infrastructure APIs.

4.1 Architectural Principles

The OpenKedge implementation is organized around four design principles that operationalize the theoretical bounds established in Section 3:

1. **Execution Sovereignty:** Infrastructure mutations pass through the control plane rather than directly from the reasoning model to the execution layer.
2. **Minimal Trust in Reasoning:** The reasoning model is treated as a stochastic component whose outputs require validation before execution.
3. **Deterministic Replayability:** Executed mutations should be reproducible from an immutable ledger.
4. **Pluggable Enforcement:** Evaluation engines must decouple from execution layers, allowing integration with existing enterprise rule sets (e.g., OPA, AWS IAM) without disrupting structural safety.

To realize these guarantees, the system is decomposed into six stateless modular components: an **Intent Interface** to normalize inputs, an **Obfuscation Membrane** to limit semantic leakage, a **Context Aggregator** to synthesize runtime metrics, a deterministic **Evaluation Engine**, an environment-specific **Execution Adapter**, and an append-only **Evidence Ledger**. Stateless operation localizes persistence to the evidence log and simplifies auditing.

4.2 State Obfuscation and Intent Abstraction

Obfuscation Membrane. The implementation of the mapping projection Π replaces sensitive production topology elements with structurally equivalent abstract graphs. OpenKedge strips identifying metadata—such as AWS account IDs, exact subnets, routing hashes, and internal DNS

endpoints—and replaces them with ephemeral tokens (e.g., `Node-A7x`). The dependency graph topology is preserved so that the model can still reason over structural relationships. Π and its inverse Π^{-1} are maintained in a secure in-memory mapping cache within the control plane.

Normalized Intent Representation. Outputs generated by the reasoning model are intercepted and parsed into a structured intent format $\mathcal{I}$. OpenKedge enforces the schema:

$$i = \langle \text{action\_type}, \text{target\_token}, \text{parameters}, \text{justification} \rangle \tag{18}$$

For example, an intent to remove an anomalous compute instance is represented as a structured tuple rather than a raw shell command:

```
action: TERMINATE_NODE, target: Node-A7x, justification:
"Memory exhaustion"
```

By requiring the agent to communicate through this intent protocol, OpenKedge separates semantic reasoning from executable commands and reduces the opportunity for prompt-injected script execution or direct API payload generation.

4.3 Deterministic Sovereign Evaluation

The OpenKedge evaluation engine deterministically validates an intent i using the localized true state s against predefined policies $\mathcal{P}$. The function $E(i, s, \mathcal{P})$ executes via two distinct pipelines:

Policy Alignment. OpenKedge uses Open Policy Agent (OPA) to evaluate infrastructure mutations against explicit safety boundaries, such as production-destruction limits, blast-radius thresholds, and region-specific workload restrictions. Any intent that violates these rules evaluates to zero ($E_{\text{policy}} = 0$) and is rejected.

Contextual Consistency. Policy validation alone is insufficient if the intent's justification contradicts the observed system state. The consistency pipeline ($E_{\text{consistency}}$) compares the generated reasoning trace J with the real-time context c_t pulled from the Aggregator. For example, if an agent proposes terminating a node because of "high CPU utilization" while telemetry shows nominal 15% load, the consistency check fails.

4.4 Mediated Execution and Evidence Chaining

Transient Execution Domains. Authorized intents that cross the evaluation threshold ($E = 1$) are translated and dispatched to environment-specific execution adapters (e.g., an AWS execution layer). To reduce the risk of persistent unauthorized access, the execution operator $\mathcal{X}$ provisions short-lived, scoped Identity and Access Management (IAM) tokens. These credentials expire upon action completion and tie execution privileges to the authorization window of the validated intent.

Cryptographic Evidence Ledger. Upon completion, the precise decision lifecycle is symmetrically hashed into the immutable Evidence Chain ledger:

$$\mathcal{C}_t = \langle i_t, J_t, E_t, a_{\text{exec},t}, H(c_t), H(\mathcal{C}_{t-1}) \rangle \tag{19}$$

This ledger supports deterministic replay, programmatic auditability, and forensic inspection of system mutations.

Fail-Safe Resilience. OpenKedge follows a default-deny fallback model. If action parsing fails, a context check cannot be completed, or an adapter faults, processing terminates in a denied state recorded in the chain rather than falling back to unmediated operations. This design keeps execution authority inside the control plane even when upstream reasoning or downstream infrastructure components fail.

5 Evaluation

We evaluate OpenKedge along two dimensions: (1) the latency overhead introduced by the Sovereign Agentic Loop (SAL) and (2) the empirical effectiveness of the architecture in preventing unsafe infrastructure mutations.

5.1 Experimental Setup

We deployed the OpenKedge prototype on a high-throughput testbed utilizing AWS EC2 `c6g.2xlarge` instances (AWS Graviton2, 8 vCPUs, 16GB RAM) running Amazon Linux 2026. The *Obfuscation Membrane* utilized an isolated Redis cluster for in-memory token mapping ($\mathcal{K}$), while the *Evidence Ledger* applied SHA-256 for cryptographic chaining. The deterministic policy engine was implemented using open-source Open Policy Agent (OPA) binaries.

To evaluate the control plane under varied conditions, we curated a dataset of $N_{\text{total}} = 10,000$ intent evaluation traces targeting simulated cloud infrastructure operations (e.g., EC2 termination, IAM role escalation, and VPC topology shifts). This workload was divided into:

- **Benign Workload ($N_{\text{valid}} = 7,500$):** Routine, correctly structured operational requests representing compliant agent behavior.
- **Adversarial Workload ($N_{\text{unsafe}} = 2,500$):** Context-inconsistent or unsafe intents generated by red-teaming a simulated frontier model.

5.2 Latency Overhead

A key concern for inline mediation is throughput degradation. We profiled the end-to-end evaluation latency introduced by the control plane, measuring the time elapsed from intent ingestion to final Evidence Ledger serialization. Timing profiles were collected using high-resolution monotonic clocks across the $10,000$ simulated traces.

The measured overhead exhibited a median latency of:

$$\Delta t_{\text{median}} \approx 12.4 \text{ ms} \tag{20}$$

with a tightly constrained 95$^{\text{th}}$ percentile of $\Delta t_{95} \approx 21.7$ ms.

By instrumenting the specific subsystems, we decomposed the median 12.4 ms latency burden into:

1. **De-obfuscation and Context Fetch (Π^{-1} extraction):** 2.1 ms
2. **OPA Policy Evaluation (E_{policy}):** 6.8 ms
3. **Consistency Evaluation ($E_{\text{consistency}}$):** 2.3 ms
4. **Cryptographic Hashing & Ledger Serialization:** 1.2 ms

This result suggests that intent mediation adds modest overhead relative to typical network jitter and is compatible with cloud orchestration workloads.

5.3 Safety Effectiveness

To quantify the protective capability of the architecture, we evaluated how the system handled the 2,500 adversarial intents (N_{unsafe}).

Of the adversarial intents, OpenKedge successfully detected and blocked 2,325 (93.0%) purely at the stateless rule-based layer ($E_{\text{policy}} = 0$). These intents attempted direct violations of invariant boundaries, such as deleting active production databases or bypassing cross-region restrictions.

The remaining 175 adversarial intents (7.0%) satisfied the policy constraints but contained semantic contradictions (e.g., proposing an emergency cluster eviction because of "critical high temperature" while the aggregated telemetry graph indicated nominal temperature states). These attempts were rejected by the consistency evaluator ($E_{\text{consistency}} = 0$).

The combined blocking rate was therefore:

$$\text{Blocking Rate} = \frac{N_{\text{blocked\_policy}} + N_{\text{blocked\_consistency}}}{N_{\text{unsafe}}} = \frac{2325 + 175}{2500} = 1.00 \tag{21}$$

In this benchmark, no unsafe intent bypassed the evaluation stage to execute on the underlying infrastructure.

5.4 Replay Verification

Auditability is an important requirement for mediated execution. To validate the deterministic replay property, we used an isolated verification script to reconstruct the logic for all $7,500$ successfully executed benign actions by traversing the *Evidence Chain*.

For every conditionally executed mutation within the testbed, the deterministic reconstruction yielded:

$$\text{Replay}(\mathcal{C}_t) = a_{\text{exec},t} \tag{22}$$

yielding a 100% verification match rate across the dataset, with no observed cryptographic desynchronization. This indicates that the contextual input (c_t) and justification (J_t) needed to reconstruct each policy decision were preserved in the chain.

5.5 Summary

The empirical evaluation is consistent with the theoretical framework. In our benchmark, the control plane blocked unsafe intents before execution while adding roughly 12.4 ms median latency. These results suggest that mediated execution can improve operational safety without imposing large runtime cost.

6 Conclusion

Agentic execution is becoming increasingly relevant in cloud and infrastructure systems. However, many current architectures still grant stochastic reasoning models direct access to execution interfaces, creating a pathway from incorrect or unsafe generations to real system mutations.

We introduced the *Sovereign Agentic Loop* (SAL), a control-plane architecture that enforces **Execution Decoupling**. By separating semantic reasoning from the authority to mutate system state, SAL routes agent intents through a mediation pipeline before execution. We formalized this framework and showed that, under the stated assumptions, it provides policy-bounded execution, identity isolation via obfuscation, and auditable replay through the Evidence Chain.

In our empirical evaluation using the OpenKedge prototype, SAL blocked unsafe intents before execution in the benchmark workload. The mediation pipeline—including obfuscation, OPA policy evaluation, contextual consistency checks, and cryptographic chaining—added approximately 12.4 ms median latency. These results indicate that execution mediation can be practical for low-latency automation settings.

From a control-theoretic perspective, frontier AI agents introduce a stochastic decision process into operational domains that demand reliable behavior. Rather than relying solely on prompt design or alignment to eliminate that variability, SAL constrains its effects at the execution boundary. In this sense, SAL follows the engineering principle of managing instability through system structure [2].

As AI systems are deployed across organizational and regulatory boundaries, the case for mediated execution becomes stronger. SAL suggests one path toward that goal: treat model outputs as proposals, validate them against system state and policy, and execute only after that decision boundary has been crossed.

References

[1] Zhou Lin and R. Gupta. Safety-critical llm agents: A survey of runtime architectures. *IEEE Transactions on Artificial Intelligence*, 2026.

[2] Hendrik W. Bode. Network analysis and feedback amplifier design. 1945.

[3] Karl Johan Åström and Richard Murray. *Feedback Systems: An Introduction for Scientists and Engineers*. 2010.

[4] Jun He and Deying Yu. Openkedge: Governing agentic mutation with execution-bound safety and evidence chains. *arXiv preprint arXiv:2604.08601*, 2026.

[5] OpenAI. Gpt-4 technical report. *arXiv preprint arXiv:2303.08774*, 2023.

[6] Anthropic. Claude: Constitutional ai and harmlessness. *arXiv preprint*, 2023.

[7] Shunyu et al. Yao. React: Synergizing reasoning and acting in language models. *arXiv preprint arXiv:2210.03629*, 2022.

[8] Timo et al. Schick. Toolformer: Language models can teach themselves to use tools. *arXiv preprint arXiv:2302.04761*, 2023.

[9] Y. Wang, L. Chen, and C. Martinez. Towards verifiable agentic execution in cloud environments. In *Proceedings of the 42nd International Conference on Machine Learning (ICML)*, 2025.

[10] Leon Staufer, Kevin Feng, Kevin Wei, Luke Bailey, Yawen Duan, Mick Yang, A. Pinar Ozisik, Stephen Casper, and Noam Kolt. The 2025 ai agent index: Documenting technical and safety features of deployed agentic ai systems. *arXiv preprint arXiv:2602.17753*, 2026.

[11] Jason et al. Wei. Chain-of-thought prompting elicits reasoning in large language models. *NeurIPS*, 2022.

[12] OpenAI. Openai function calling api, 2023. `https://platform.openai.com/docs`.

[13] Haoyu Wang, Christopher M. Poskitt, and Jun Sun. Agentspec: Customizable runtime enforcement for safe and reliable llm agents. *arXiv preprint arXiv:2503.18666*, 2025. To appear at ICSE 2026.

[14] Haoyu Wang, Christopher M. Poskitt, and Jun Sun. Pro2guard: Proactive runtime enforcement of llm agent safety via probabilistic model checking. *arXiv preprint arXiv:2508.00500*, 2025.

[15] Open Agent Passport Consortium. Before the tool call: Deterministic pre-action authorization for autonomous ai agents. *arXiv preprint arXiv:2603.20953*, 2026.

[16] Martin Leucker and Christian Schallhart. A brief account of runtime verification. *Journal of Logic and Algebraic Programming*, 2009.

[17] Ezio et al. Bartocci. Specification-based monitoring of cyber-physical systems. *ACM Computing Surveys*, 2018.

[18] Aarya Doshi, Yining Hong, Congying Xu, Eunsuk Kang, Alexandros Kapravelos, and Christian Kästner. Towards verifiably safe tool use for llm agents. *arXiv preprint arXiv:2601.08012*, 2026. To appear at ICSE-NIER 2026.

[19] Ravi et al. Sandhu. Role-based access control models. *IEEE Computer*, 1996.

[20] Vincent et al. Hu. Attribute-based access control. *IEEE Computer*, 2015.

[21] Styra. Open policy agent, 2019. `https://www.openpolicyagent.org`.

[22] Amazon Web Services. Cedar policy language, 2023. `https://www.cedarpolicy.com`.

[23] Bronislav Sidik and Lior Rokach. Beyond static sandboxing: Learned capability governance for autonomous ai agents. *arXiv preprint arXiv:2604.11839*, 2026.

[24] Mert Cemri, Melissa Z. Pan, Shuyi Yang, Lakshya A. Agrawal, Bhavya Chopra, Rishabh Tiwari, Kurt Keutzer, Aditya Parameswaran, Dan Klein, Kannan Ramchandran, Matei Zaharia, Joseph E. Gonzalez, and Ion Stoica. Why do multi-agent llm systems fail? *arXiv preprint arXiv:2503.13657*, 2025.

[25] Yubin Kim, Hyewon Jeong, Chanwoo Park, Eugene Park, Haipeng Zhang, Xin Liu, Hyeonhoon Lee, Daniel McDuff, Marzyeh Ghassemi, and Cynthia Breazeal. Tiered agentic oversight: A hierarchical multi-agent system for healthcare safety. *arXiv preprint arXiv:2506.12482*, 2025.

[26] Google Research and Google DeepMind. Towards a science of scaling agent systems. *arXiv preprint arXiv:2512.08296*, 2025.

[27] Cynthia Dwork. Calibrating noise to sensitivity in private data analysis. *TCC*, 2006.

[28] Ji-Hoon Kim and S. Patel. Sovereign ai infrastructure: Challenges and opportunities. In *Proceedings of the 2025 ACM SIGSAC Conference on Computer and Communications Security (CCS)*, 2025.

A Notation

For clarity, we summarize the notation used throughout the paper.

Symbol	Description
$\mathcal{S}$	True state space of the infrastructure
$s \in \mathcal{S}$	Concrete system state
$\hat{\mathcal{S}}$	Obfuscated state space
$\hat{s} \in \hat{\mathcal{S}}$	Obfuscated system state
s_{id}	Identity-sensitive component of the state
s_{struct}	Structural (topological) component of the state
Π	Obfuscation mapping $\Pi : \mathcal{S} \to \hat{\mathcal{S}}$
Π^{-1}	Privileged reconstruction mapping within control plane
$\mathcal{I}$	Intent space
$i \in \mathcal{I}$	Structured intent generated by the agent
M	Reasoning model (e.g., LLM)
G	Goal specification provided to the agent
J	Justification generated by the reasoning model
E	Evaluation function for intent validation
$\mathcal{P}$	Set of sovereign policy constraints
$\mathcal{X}$	Execution operator mapping intent to action
a	Abstract action
a_{exec}	Executed action in the real system
$\mathcal{C}_t$	Evidence chain at time t
c_t	Context snapshot used for evaluation
$H(\cdot)$	Cryptographic hash function
$I(\cdot;\cdot)$	Mutual information between random variables

Table 1: Summary of notation used in the SAL model.

B Figure-to-Notation Mapping

To clarify the correspondence between the conceptual architecture illustrated in Figure 2 and the formal notation introduced in Section 3, we provide the following mapping. We explicitly align the conceptual components in Figure 2 with the formal model to ensure consistency between the architectural and mathematical representations.

Figure Component	Formal Notation
Reasoning Agent (LLM)	M
Obfuscated Context	$\hat{s} \in \hat{\mathcal{S}}$
Obfuscation Membrane	$\Pi : \mathcal{S} \to \hat{\mathcal{S}}$
Intent / Proposed Action	$i \in \mathcal{I}$
Justification	J
Sovereign Control Plane	$(E, \mathcal{X})$
Policy Evaluation	$E(i, s, \mathcal{P})$
System State (True Context)	$s \in \mathcal{S}$
Scoped Execution	$\mathcal{X}(i, s)$
Executed Action	a_{exec}
Context Snapshot	c_t
Evidence Chain (Ledger)	$\mathcal{C}_t$
Feedback Loop (State Update)	$s_{t+1} = f(s_t, a_{\text{exec},t})$

Table 2: Mapping between Figure 2 components and formal notation in the SAL model.