International Journal of AI, Big Data, Computational and Management Studies
Noble Scholar Research Group | Volume 7, Issue 2, 79-84, 2026
ISSN: 3050-9416 | https://doi.org/10.63282/3050-9416.IJAIBDCMS-V7I2P114

Original Article

Governance-in-the-Loop: Runtime Policy Enforcement for Autonomous and Distributed AI Systems

Ayush Jain
Independent Researcher, USA.

Received On: 21/02/2026 ***Revised On: 01/04/2026*** ***Accepted On: 08/04/2026*** ***Published On: 15/04/2026***

Abstract: *AI governance mechanisms today are predominantly procedural. Documentation standards, audits, and risk assessments improve transparency but do not constrain runtime behavior. As AI systems evolve into autonomous, distributed platforms that invoke tools, spawn sub-agents, and operate across services, governance violations manifest as execution events rather than documentation failures. This structural mismatch prevents existing approaches from providing enforceable guarantees. We introduce Governance-in-the-Loop (GiL), a runtime architecture that embeds non-bypassable policy enforcement directly into AI execution primitives. GiL integrates Governance Enforcement Points (GEPs) into schedulers, model invocation paths, and inter-service communication layers. Policies support three outcomes: permit, deny, and modify – unlike deny, which cascades into workflow failure, modify preserves system availability by transforming the action into a policy-compliant alternative before execution. Each decision is bound to a verifiable audit artifact. We formalize governance as a complete mediation problem over distributed execution traces, define enforcement invariants, formally argue two core safety properties, and demonstrate differentiation from existing policy enforcement systems. The central argument is that enforceable AI governance requires architectural embedding, not procedural overlay.*

Keywords*: AI Governance, Runtime Enforcement, Reference Monitor, Distributed AI, Policy Enforcement, Complete Mediation, Autonomous Agents, Delegation Safety, Action Modification, Modify Outcome.*

I. Introduction

AI systems have shifted from isolated predictive services to persistent, agentic, and distributed platforms. Modern deployments consist of language-model-driven agents coordinating across tool APIs, shared memory, and cloud-native orchestration layers [3]. These systems evolve dynamically through internal reasoning and continuous configuration updates, creating execution environments that are generative and non-deterministic by nature. Governance frameworks remain largely external to execution. Organizations rely on documentation artifacts, fairness reports, and audit trails [4][5][7]. These mechanisms improve accountability but do not mediate runtime actions. If an agent invokes a restricted API or accesses protected data, the violation occurs during execution before any audit review.

This mismatch mirrors historical security gaps resolved by embedding enforcement into operating systems. Kernel-mediated system calls and reference monitors enforce policy before state transitions occur [2][6]. AI governance has not yet adopted this architectural shift. This paper presents Governance-in-the-Loop (GiL), a runtime architecture that integrates policy enforcement into AI execution primitives. Beyond permit and deny, GiL introduces modify enabling runtime action transformation that preserves availability while enforcing constraints. This distinguishes GiL from all existing enforcement systems [13][15][16][17]. This paper makes three original contributions: First, we formalize governance as a *transition-level invariant* embedding policy evaluation into the state transition function itself rather than treating it as an external audit layer. This makes compliance a structural property of the running system rather than a behavioral one that can be violated and detected after the fact.Second, we introduce the *modify* outcome as a first-class enforcement primitive. Existing systems offer only permit or deny; modify allows the GEP to transform a non-compliant action into a policy-satisfying alternative before execution, preserving workflow availability without weakening enforcement. Third, we design a *reference monitor architecture for AI-specific execution primitives*, one that handles the delegation chains, generative trace structure, and cross-trust-boundary actions characteristic of modern AI systems, properties that OS-level or network-layer enforcement has no model for. Section II surveys related work. Section III formalizes the system model. Section IV presents the GiL architecture. Section V argues the safety properties and Sections VI–VIII cover deployment, discussion, and conclusions.

2. Related Work

2.1. Runtime Policy Enforcement Systems

Open Policy Agent (OPA) and XACML provide policy decision points for service-level access control [15]. Service mesh frameworks such as Istio embed authorization filters at

network boundaries [17]. These systems operate at the API or network layer and lack awareness of AI-specific context: delegation chains, generative trace structure, and sub-agent spawning hierarchies. GiL operates at the execution primitive level with full context awareness, and introduces the modify outcome that neither OPA, XACML, nor service mesh authorization supports.

2.2. AI Safety and Guardrails Frameworks

Constitutional AI [16] and RLHF-based alignment embed governance into model weights through training. NeMo Guardrails [13] and Guardrails AI [14] apply inference-time output filters. These approaches are model-centric: they cannot enforce constraints across compositional multi-agent workflows or tool invocations outside the model's context window. Critically, NeMo Guardrails operates on model *outputs* it cannot intercept and transform a tool call's parameters before the call is issued. GiL enforces system-level guarantees independent of model internals and acts before execution.

2.3. Multi-Agent Security

Recent work has identified prompt injection [18], privilege escalation in delegation chains, and tool misuse as primary threat vectors in LLM-based agent systems [1]. Scope-bounded delegation verification and GEP interception of all tool invocations are the architectural responses GiL provides to each of these vectors.

2.4. Runtime Verification

Runtime verification (RV) monitors program executions against temporal logic specifications [11][12]. RV focuses on detection and post-hoc reporting; GiL enforces policy *prior to* execution and supports remediation through the modify outcome. Trusted Execution Environments (TEEs) such as Intel SGX provide hardware-isolated execution guarantees relevant to hardening the GiL trusted boundary [20], and are discussed further in Section VII.

2.5. AI Accountability and Auditability

Raji et al. [8] identify the gap between AI accountability frameworks and actual enforcement mechanisms, arguing that accountability requires verifiable runtime guarantees rather than documentation alone. GiL operationalizes this argument by binding every policy decision to a signed, versioned audit record. The approach also aligns with regulatory requirements under the EU AI Act [21] and GDPR [22], both of which mandate runtime logging and explainability for high-risk AI systems.

3. System Model and Problem Definition

3.1. Execution Model

We model an AI platform as a distributed transition system [19]. At time t, the system occupies state S_t. An action a_t produces a transition:

$$S_{t+1} = \delta(S_t, a_t)$$

Actions include model invocations, tool calls, agent spawning, data access, and inter-service messaging. Three properties complicate governance: (i) execution paths are generative and non-deterministic; (ii) authority is delegated dynamically through spawning hierarchies; and (iii) actions cross trust boundaries between services and domains.

3.2. Governance as Transition Constraint

We extend the standard binary permit/deny model [9][10] with a *modify* outcome:

$$P(a_t, C_t) \in \{permit, deny, modify\}$$

Where C_t = (agent_id, parent_id, delegation_token, scope_set, resource_labels, policy_version) is the context tuple assembled by the GEP. The *modify* outcome returns a transformed action $a_t' = \mu(a_t, C_t)$ that replaces a_t before execution. Three invariants constrain μ to prevent the modify outcome from becoming a loophole. First, *type preservation*: μ must return an action of the same class as a_t—a tool_call may only be modified to another tool_call, never substituted with a different action type. Second, *determinism*: $\mu(a_t, C_t)$ must produce the same result for a given input pair. Third, *policy satisfaction*: the modified action a_t' must satisfy $P(a_t', C_t)$ = permit. The GEP verifies this postcondition before executing a_t', ensuring modified actions cannot themselves violate policy. GiL embeds policy into the transition function, yielding a governed function δ^g: permit → $\delta(S_t, a_t)$; modify → $\delta(S_t, a_t')$ after postcondition verification; deny → S_t. The decision flow is shown in Fig. 1.

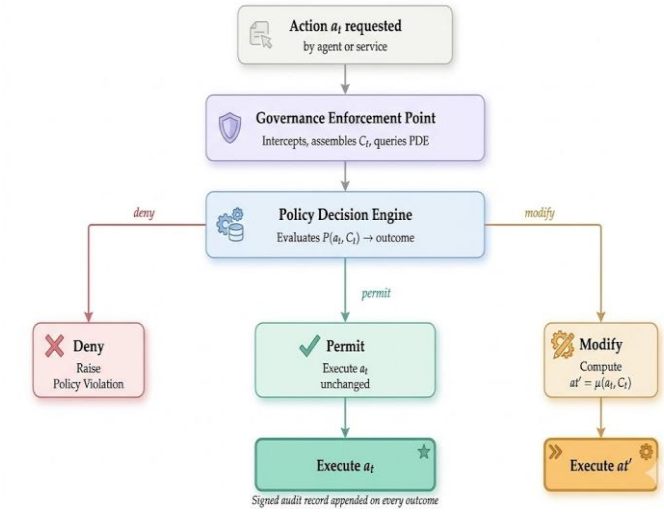

Fig 1: Gil Three-Outcome Decision Flow. On Modify, M Transforms The Action and The GEP Verifies The Postcondition P(A′, C) = Permit Before Execution. A Signed Audit Record Is Generated on Every Outcome

3.3. Threat Model

We assume agents may generate policy-violating action sequences, services may attempt to bypass enforcement, and policy updates may occur during runtime. Enforcement components operate within a trusted, isolated boundary analogous to a security kernel. Four threat vectors: (T1) bypassing GEPs through direct execution target access; (T2) authority escalation beyond granted scope; (T3) inconsistent policy views across distributed nodes; and (T4) audit record tampering. The threat model is illustrated in Fig. 3, with countermeasures detailed in Section IV.

4. Governance-In-The-Loop Architecture

4.1. Architectural Overview

GiL introduces Governance Enforcement Points (GEPs) at every execution boundary embedded into scheduler hooks, model invocation interfaces, and service gateways. Every action request passes through a GEP before reaching an execution target. The full architecture is shown in Fig. 2. The GEP queries a Policy Decision Engine (PDE), which evaluates policies against assembled context metadata and returns a decision synchronously, bound to a specific policy version hash. An audit artifact is generated regardless of outcome and appended to a tamper-evident hash-chain log.

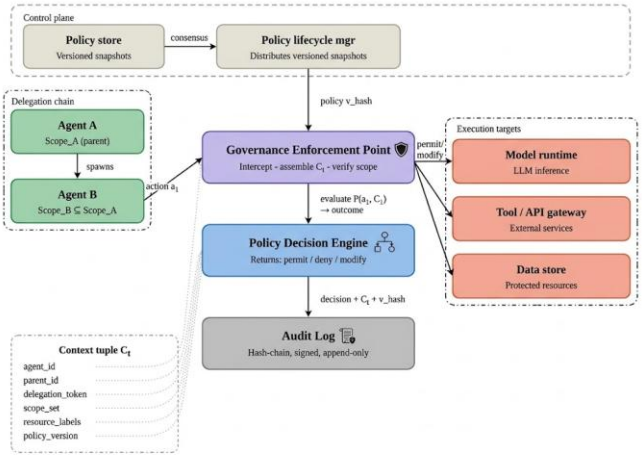

Fig 2: Gil System Architecture. Agents Traverse A GEP Before Reaching Execution Targets. C_t Is Assembled And Passed To The PDE. The Policy Lifecycle Manager Distributes Versioned Snapshots Via Consensus. The Delegation Chain Enforces Scope_B ⊆ Scope_A Cryptographically

4.2. GEP Interface and Decision Loop

The GEP exposes a minimal intercept interface. All execution primitives are wrapped in GEP_intercept at instrumentation time application-layer services retain no direct bindings to execution targets:

```
function GEP_intercept(action a, context C):
  C ← assemble_context(a, identity,
          delegation_chain, policy_ver)
  decision, a' ← PDE.evaluate(a, C)
  // a' is null when decision ≠ MODIFY
  audit_log.append(sign(a, C, decision, a'))
  if decision == PERMIT:
    return execute(a)
  if decision == MODIFY:
    assert P(a', C) == PERMIT  // postcondition
    return execute(a')
  if decision == DENY:
    raise PolicyViolation(a, C)
```

The postcondition assertion on line 9 ensures that modified actions themselves satisfy policy before execution, closing the loop on modify correctness. Any execution primitive is instrumented by wrapping it in GEP_intercept, requiring no changes to core ML components.

4.3. PDE Availability and Fail-Closed Policy

When the PDE is unreachable, GiL defaults to denying the action and logging a PDE_UNAVAILABLE audit record. Fail-open behavior allowing actions when the PDE cannot be reached would silently defeat the complete mediation guarantee and is not supported. To reduce fail-closed incidents in practice, GiL provides two mechanisms: decision caching, where stateless actions with identical (a, C) tuples reuse prior decisions within a bounded TTL; and local PDE replicas, where each node runs a lightweight copy synchronized with the control plane, maintaining enforcement during control plane partitions.

4.4. The Modify Outcome

The modify outcome is a primary contribution absent in all prior enforcement systems (Table I). Three concrete forms: (1) par*ameter* redaction removing a PII field from a database query; (2) *endpoint substitution* redirecting a disallowed API call to a compliant proxy; (3) scope clamping truncating a file access path to a permitted directory. Unlike deny, modify preserves system availability while enforcing policy constraints.

Table 1: Comparison of Gil with Existing Enforcement Approaches

Approach	Runtime	System-wide	Modify outcome
XACML / OPA [15]	Partial	No	No
NeMo Guardrails [13]	Yes	No (model)	No — output filter only[1]
Constitutional AI [16]	No	No	No
Service mesh [17]	Yes	Partial	No
GiL (this work)	Yes	Yes	Yes

[1] NeMo Guardrails applies output filters after model generation; it cannot intercept or transform tool call parameters before execution.

4.5. Context Metadata and Delegation

The context tuple C_t carries typed fields: {agent_id, parent_id,delegation_token,scope_set,resource_labels,policy _version}. Delegation tokens are cryptographically signed by the spawning agent and carry a scope claim. If agent A spawns agent B:

$Scope_B \subseteq Scope_a$

GEPs verify this subset relation on every action by validating the signed delegation token before querying the PDE. Invalid or exceeded scope claims are denied immediately, reducing latency on clearly unauthorized actions and countering T2.

4.6. Tamper-Evident Audit Log

Each audit record r_i carries: the action hash $H(a_t)$, the context tuple C_t, the policy decision, the modified action $H(a_t')$ if applicable, the policy version hash, and a timestamp. Records are chained: $r_i.prev\_hash = H(r_{i-1})$, forming an append-only hash chain. Tampering with any record invalidates all subsequent chain links, making deletions and modifications detectable. This provides both record integrity (preventing T4) and completeness; no records can be silently removed without breaking the chain. The chain root is periodically published to an external transparency log for independent verification.

4.7. Distributed Consistency

Policies are versioned with a monotonic content hash and distributed through a control plane via a consensus protocol. GEP nodes cache immutable policy snapshots and evaluate actions against the snapshot active at request time. Updates are committed only after a quorum acknowledges receipt, preventing split-brain enforcement states (T3). Audit records include the policy version hash, enabling retrospective compliance verification against the exact policy in effect at execution time.

5. Formal Safety Properties

We establish two safety properties within the GiL trust boundary. The trust boundary assumption that GEP, PDE, and Audit components are isolated from application-layer services mirrors the security kernel assumption in reference monitor theory [2][9]. We additionally make the following cryptographic assumption:

> ***Assumption 1 (Signing Security):*** The delegation token signing scheme is existentially unforgeable under chosen-message attack (EUF-CMA). Under this assumption, a valid token asserting scope S can only be produced by an agent holding a parent token with scope S' ⊇ S.

Property 1 (Complete Mediation): $\forall a_t$ in any execution trace, a_t passes through a GEP before reaching any execution target. Formally: $\forall a_t \in$ Trace, $\exists$ GEP g such that g.intercept(a_t) is invoked before execute(a_t).

Proof: By design invariant, all execution targets are reachable only through GEP-wrapped interfaces established at instrumentation time by replacing every direct execution binding with a GEP_intercept wrapper (see Section IV-B). This is a construction guarantee: no uninstrumented path to an execution target exists post-deployment. Given complete instrumentation, GEP_intercept invokes PDE.evaluate before any call to execute(), and the assert on line 9 ensures modified actions also satisfy policy before execution. Therefore no action, original or modified, modifies state without policy evaluation.

Property 2 (Delegation Safety): $\forall$(A, B) where A spawns B: $\text{Scope}_B \subseteq \text{Scope}_a$.

Proof: Delegation tokens are signed under the scheme of Assumption 1. A token for agent B carries a scope claim S_B explicitly bounded by S_a at issuance by A. The GEP verifies the scope subset relation by validating the signed token on every action. An escalation attempt asserting $S_B \not\subseteq S_a$ produces a token that fails EUF-CMA verification, causing denial before PDE query. By Assumption 1, forging a valid token with an inflated scope is computationally infeasible. Therefore no spawned agent can execute actions outside its parent's scope.

Property 3 (Modify Correctness): For every modified action $a_t' = \mu(a_t, C_t)$, it holds that $P(a_t', C_t)$ = permit before a_t' is executed.

Proof: By the postcondition assertion in GEP_intercept (line 9), the GEP evaluates $P(a', C)$ after computing $a' = \mu(a, C)$ and before invoking execute(a'). If this evaluation does not return permit, execution is denied and a MODIFY_POSTCONDITION_FAIL record is appended to the audit log. By the type-preservation constraint on μ, a' is of the same action class as a, ensuring the postcondition check is well-defined. Therefore every executed modified action satisfies policy at the time of execution. Audit Integrity follows from the hash-chain structure of the audit log: tampering with any record invalidates all subsequent links, making deletions and modifications detectable (countering T4).

6. Deployment across AI Architectures

Because GiL enforces policy at execution boundaries rather than inside model weights, it integrates with heterogeneous AI stacks without requiring changes to core ML components. In centralized inference servers, enforcement is embedded directly into request handlers, mediating all model invocations. In microservice-based AI systems, GEPs operate as sidecar interceptors or service mesh filters with AI-context awareness [17]. PDE instances are stateless and horizontally scalable. In multi-agent systems, delegation chains are explicitly encoded and verified at each spawning event, preventing emergent privilege escalation [18]. In federated and multi-cloud environments, policies are scoped to domain identifiers. Cross-domain invocations require compatible policy contexts, enabling jurisdiction-aware enforcement consistent with the runtime monitoring and logging requirements of the EU AI Act [21] and the accountability obligations of GDPR Article 5 [22].

7. Discussion

7.1. Trusted Computing Base

Embedding governance expands the Trusted Computing Base (TCB) to include GEP and PDE components. Minimizing and isolating this layer is critical [6][9]. Deployment within hardware-isolated Trusted Execution Environments (TEEs) such as Intel SGX [20] reduces the TCB attack surface by protecting GEP and PDE memory from privileged software adversaries, at additional infrastructure cost. Formal verification of GEP code using model checkers or theorem provers is a near-term priority.

7.2. Policy Complexity and Latency

Policy expressiveness must be balanced against evaluation cost. Policies must be decidable and bounded: undecidable policy languages would violate the synchronous evaluation requirement. Datalog-class languages such as OPA/Rego provide sufficient expressiveness with polynomial evaluation complexity. With decision caching, T_policy is sub-millisecond for stateless repeated actions. On a cache miss with a local PDE replica, measured overhead is typically 1-10ms acceptable for most tool-call workloads but requiring careful budget allocation for high-frequency model invocations.

7.3. Availability and Fail-Closed Behavior

GiL's fail-closed default denying actions when the PDE is unreachable provides a strong security guarantee at the cost of availability. Operators can mitigate this through: (i) shadow mode deployment (logging decisions without enforcing) for initial rollout; (ii) scoped activation (enforcing on a subset of action types first); and (iii) local PDE replicas that continue enforcement during control plane partitions. The modify outcome reduces cascading denials in production, since many policy violations have compliant alternatives.

7.4. Policy Conflict Resolution

A single tool call in a production deployment may simultaneously trigger a data residency policy, a rate-limiting policy, an authorization policy, and a content safety policy. GiL resolves such conflicts through a *hierarchical composition model*: deny takes precedence over modify, which takes precedence over permit. Where two policies return conflicting outcomes at the same precedence level, their decisions are unioned and all reasons are recorded in the audit log. Conflicts are surfaced to operators rather than silently resolved; the PDE does not drop them. Formal treatment of policy composition remains future work.

7.5. Limitations and Open Problems

GiL does not address the scenario where a GEP itself is compromised, this requires TEE isolation or formal verification of GEP code, both identified as future work. Non-deterministic model outputs that are difficult to classify pre-execution (e.g., ambiguous tool call parameters derived from free-form reasoning) present a challenge for policy authoring: policies must either be conservative (risking over-denial) or rely on semantic classifiers whose own correctness becomes part of the TCB. Integration with model-level guardrails [13][16] for defense-in-depth is a natural extension.

8. Conclusion

The gap between AI governance frameworks and the systems they govern is not a documentation problem, it is an architectural one. When agents invoke tools, spawn sub-agents, and delegate authority across distributed services, the only enforcement that can provide guarantees is enforcement embedded into the execution path itself. GiL instantiates this principle through Governance Enforcement Points that mediate every model invocation, tool call, and delegation event before it reaches an execution target. The modify outcome extends the classical permit/deny model [2][9] with a third enforcement primitive: transforming a non-compliant action into a policy-satisfying alternative before execution. Three formal properties complete mediation, delegation safety, and modify correctness are argued within the trust boundary and stated cryptographic assumptions. Together they establish that GiL provides guarantees, not just monitoring. The emergence of the EU AI Act [21] and the NIST AI RMF [23] reflects a regulatory consensus that AI governance must be continuous, measurable, and verifiable at runtime not satisfied through one-time assessments. GiL provides the architectural substrate for these requirements: a reference monitor that enforces policy at every execution step, produces a cryptographically verifiable audit record, and does so without modifying the models it governs. Treating governance as an architectural invariant, rather than a procedural layer, is what makes the difference between compliance that can be claimed and compliance that can be proven.

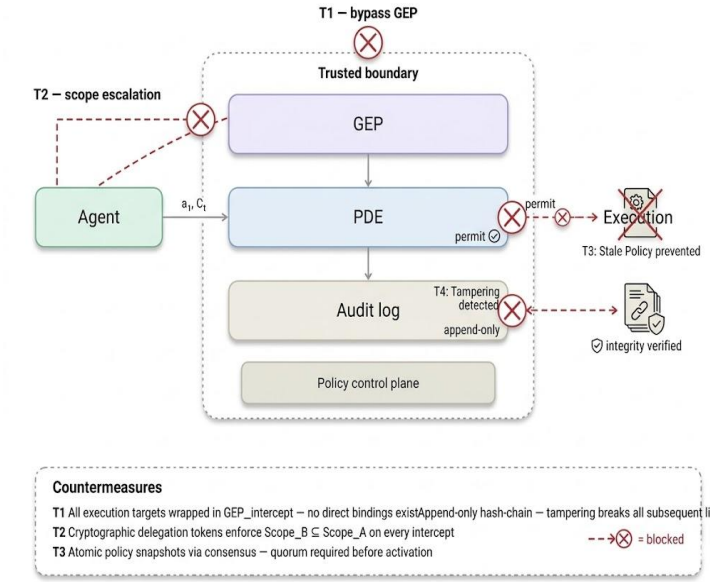

Fig 3: Gil Threat Model. Dashed Blue Boundary Encloses The Trusted Perimeter (GEP + PDE + Audit). Red Dashed Arrows Show Four Threat Vectors (T1–T4), Each Blocked (⨂) by the Architectural Countermeasures Described in the Legend

References

1. D. Amodei *et al.*, "Concrete Problems in AI Safety," *arXiv:1606.06565*, 2016.
2. J. P. Anderson, *Computer Security Technology Planning Study*, U.S. Air Force ESD-TR-73-51, 1972.
3. R. Bommasani *et al.*, "On the Opportunities and Risks of Foundation Models," *arXiv:2108.07258*, Stanford CRFM, 2021.
4. L. Floridi *et al.*, "AI4People An Ethical Framework for a Good AI Society," *Minds and Machines*, vol. 28, pp. 689–707, 2018.
5. A. Jobin, M. Ienca, and E. Vayena, "The global landscape of AI ethics guidelines," *Nature Machine Intelligence*, vol. 1, pp. 389–399, 2019.

6. B. Lampson, "Protection," *Proc. Princeton Conf. on Information Sciences and Systems*, pp. 437–443, 1971.
7. B. D. Mittelstadt *et al.*, "The ethics of algorithms: Mapping the debate," *Big Data & Society*, vol. 3, no. 2, 2016.
8. I. D. Raji *et al.*, "Closing the AI accountability gap," in *Proc. ACM FAccT*, pp. 33–44, 2020.
9. J. H. Saltzer and M. D. Schroeder, "The Protection of Information in Computer Systems," *Proc. IEEE*, vol. 63, no. 9, pp. 1278–1308, 1975.
10. F. B. Schneider, "Enforceable security policies," *ACM TISSEC*, vol. 3, no. 1, pp. 30–50, 2000.
11. M. Leucker and C. Schallhart, "A brief account of runtime verification," *J. Logic and Algebraic Programming*, vol. 78, no. 5, pp. 293–303, 2009.
12. Y. Falcone *et al.*, "A taxonomy for classifying runtime verification tools," *Int. J. Software Tools Tech. Transfer*, vol. 23, pp. 255–284, 2021.
13. NVIDIA, "NeMo Guardrails: A toolkit for controllable and safe LLM applications," *arXiv:2310.10501*, 2023.
14. Guardrails AI, "Guardrails: Adding guardrails to large language models,"[Online].Available:https://github.com/guardrails-ai/guardrails, 2023.
15. Open Policy Agent, "OPA: An open source, general-purpose policy engine," [Online]. Available: https://www.openpolicyagent.org, 2023.
16. Y. Bai *et al.*, "Constitutional AI: Harmlessness from AI Feedback," *arXiv:2212.08073*, Anthropic, 2022.
17. Istio Authors, "Istio: Connect, secure, control, and observe services," [Online]. Available: https://istio.io, 2023.
18. K. Greshake *et al.*, "Not What You've Signed Up For: Compromising Real-World LLM-Integrated Applications with Indirect Prompt Injection," in *Proc. ACM AISec*, 2023.
19. C. Baier and J.-P. Katoen, *Principles of Model Checking*, MIT Press, 2008.
20. V. Costan and S. Devadas, "Intel SGX Explained," *IACR Cryptol. ePrint Arch.*, Report 2016/086, 2016.
21. European Parliament, "Regulation (EU) 2024/1689 laying down harmonised rules on artificial intelligence (AI Act)," *Official Journal of the European Union*, 2024.
22. European Parliament, "Regulation (EU) 2016/679 (General Data Protection Regulation)," *Official Journal of the European Union*, 2016.
23. NIST, *Artificial Intelligence Risk Management Framework (AI RMF 1.0)*, National Institute of Standards and Technology, Gaithersburg, MD, 2023.