The Governed Process Runtime (GPR): A Unified Architecture for Zone III Enterprise Agentic Transformation

Eigenvector Research

Eigenvector Research Institute

Author Note

This paper is a design-science contribution (Hevner et al., 2004). All architectural claims are grounded in the Eigenvector Research corpus and the external academic literature cited. No empirical data was fabricated. Correspondence concerning this article should be addressed to Eigenvector Research Institute.

Abstract

Enterprise AI has reached a point at which model capability is no longer the central architectural constraint. The harder problem is how to automate consequential process work that unfolds across time, approvals, exceptions, evidence obligations, semantic ambiguity, and institutional accountability. This paper defines the **Governed Process Runtime (GPR)** as a new architectural class designed specifically for this regime, designated **Zone III**. The GPR converges the conceptual requirements of the Enterprise Intelligence Platform (EIP) with the operational mechanics of the Friedmann-Gleichung Machine (FGM), the Ontological Compliance Gateway (OCG), the Sovereign Verification Engine (SVE), and the Governance Runtime Assurance Framework (GRAF) into a single, unified, formally specified system. The paper argues that current enterprise AI systems fail not due to model limitations, but due to the absence of a runtime capable of enforcing governance, semantic correctness, and trajectory-aware control over long-running, high-risk processes. The core contribution is the rigorous specification of **42 runtime primitives**—including the Execution Envelope, CaseState, PolicyPath, ContextContract, AgentIntent, ObjectiveFunction, GoalState, AutonomyLevel, DecisionBoundary, AgentExecutionTrace, GoalDeviation, StrategyConstraint, and AutonomyEscalationTrigger—that transition enterprise AI from a conversational paradigm to a governed agentic execution paradigm. The paper adopts a design-science research stance (Hevner et al., 2004) and provides a complete formal specification, conceptual architecture, implementation guidance, open-world evaluation framework, and economic governance model suitable for sovereign enterprise deployment.

Keywords: Governed Process Runtime, Zone III, Agentic AI, Enterprise AI Governance, Execution Envelope, PolicyPath, Trajectory-Aware Governance, Semantic Authority, Multi-Agent Systems, Roundtrip Value Governance

1. Introduction and the Agentic Execution Model

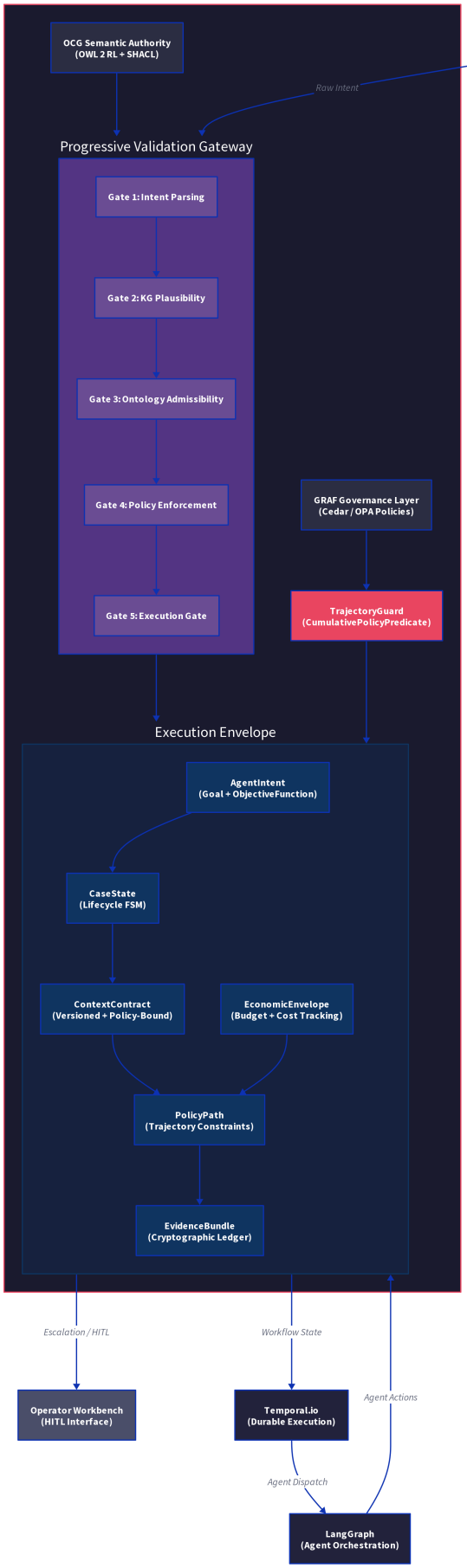
The most valuable work inside large enterprises is rarely a single prediction, a single answer, or a single conversational turn. It is a trajectory. Cases wait for approvals. Evidence arrives late. Policies conflict. Roles change. Exceptions surface. Human reviewers intervene. Systems of record must be updated without breaking authority boundaries. In that setting, the limiting factor is not intelligence in the abstract. It is institutionally bounded execution.

That challenge defines what this paper calls **Zone III**: the operating regime between rigid workflow and unconstrained autonomy. Zone III processes are structured enough to justify automation, but too semantically unstable, exception-sensitive, evidence-bearing, and governance-exposed for conventional BPM alone. At the same time, they are too consequential to entrust to loosely governed copilots or free-form agent stacks. This is where a large share of enterprise value remains trapped.

The current market often misdiagnoses the problem. It assumes that better prompting, larger models, or more agents will eventually dissolve enterprise complexity. That is the wrong frame. In consequential settings, the first-order questions are architectural: where policy is enforced, how meaning is stabilized, how evidence is preserved, how context survives interruption, how human override is encoded, and whether the deployment boundary remains enterprise-controlled. If those questions are unresolved, the system may be useful, but it is not yet trustworthy.

This paper introduces the **Governed Process Runtime (GPR)** as the architectural answer to that question. The GPR is not a product. It is a formal system specification—a blueprint for a class of runtime that can govern, execute, and audit Zone III enterprise processes under real-world constraints.

Figure 1. The Governed Process Runtime Kernel



Figure

Figure 1. The GPR Kernel Architecture

Figure 1. The GPR Kernel Architecture, showing the Execution Envelope, Progressive Validation Gateway, TrajectoryGuard, OCG Semantic Authority, and GRAF Governance Layer.

1.1 Agentic Transformation as a Core Concept

This paper explicitly formalizes **agentic transformation** as a core concept. Agentic transformation is defined as the controlled delegation of goal-oriented behavior to autonomous agents operating within policy-bound, context-admissible execution environments. This definition is not optional. It represents a fundamental shift from task automation to goal delegation. In task automation, the system executes a predefined sequence of steps. In agentic transformation, the system pursues a defined goal, adapting its strategy based on environmental feedback, while remaining strictly bounded by institutional governance (Kaptein et al., 2026; OpenAI, 2024).

The distinction matters because it changes the nature of the governance problem. When the system is executing a predefined sequence, governance can be applied at the sequence level—checking each step against a rulebook. When the system is pursuing a goal, governance must be applied at the trajectory level—evaluating whether the cumulative sequence of actions taken so far, and the proposed future trajectory, remain within institutional bounds. This is the core insight that motivates the GPR architecture.

1.2 The Formal Agentic Execution Model

To support agentic transformation, the GPR introduces a formal agentic execution model. Agents in the GPR are not abstract entities or simple LLM wrappers. They are goal-

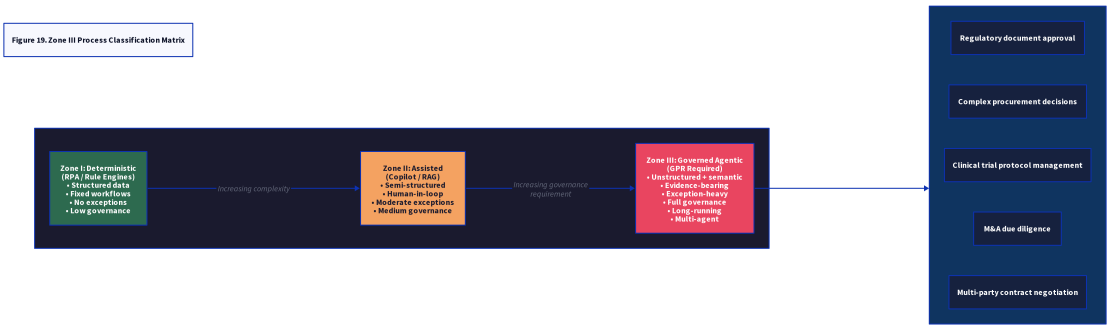
directed execution entities operating within Execution Envelopes under strict governance. The formal agentic execution model includes:

- **AgentIntent**: The formal specification of the goal the agent is authorized to pursue, including the target entity, authorized actions, and success criteria.
- **Objective Functions and Success Criteria**: The metrics by which the agent evaluates its progress toward the GoalState.
- **Agent Autonomy Boundaries**: The explicit limits on what the agent is permitted to do, encoded as AutonomyLevel and DecisionBoundary.
- **Allowed and Disallowed Strategies**: The predefined strategies the agent is permitted to employ and those explicitly forbidden, encoded as StrategyConstraints.

This model draws on foundational work in bounded autonomy for enterprise AI (Sohail & Haider, 2026; Saini, 2026) and extends it with formal runtime primitives that can be enforced at execution time.

2. The Zone III Classification

Not all enterprise processes belong in Zone III. The GPR is specifically designed for a well-defined class of processes that share a common set of characteristics. Understanding this classification is essential for understanding why the GPR architecture is necessary.



Figure

Figure 19. Zone III Process Classification Matrix

Figure 19. The Zone III Process Classification Matrix, distinguishing deterministic Zone I processes (RPA/rule engines), assisted Zone II processes (copilot/RAG), and governed agentic Zone III processes requiring the GPR.

Zone I processes are deterministic and fully structured. They involve structured data, fixed workflows, no exceptions, and low governance overhead. These are the domain of traditional RPA and rule-engine automation. The governance challenge is minimal because the process is fully specified in advance.

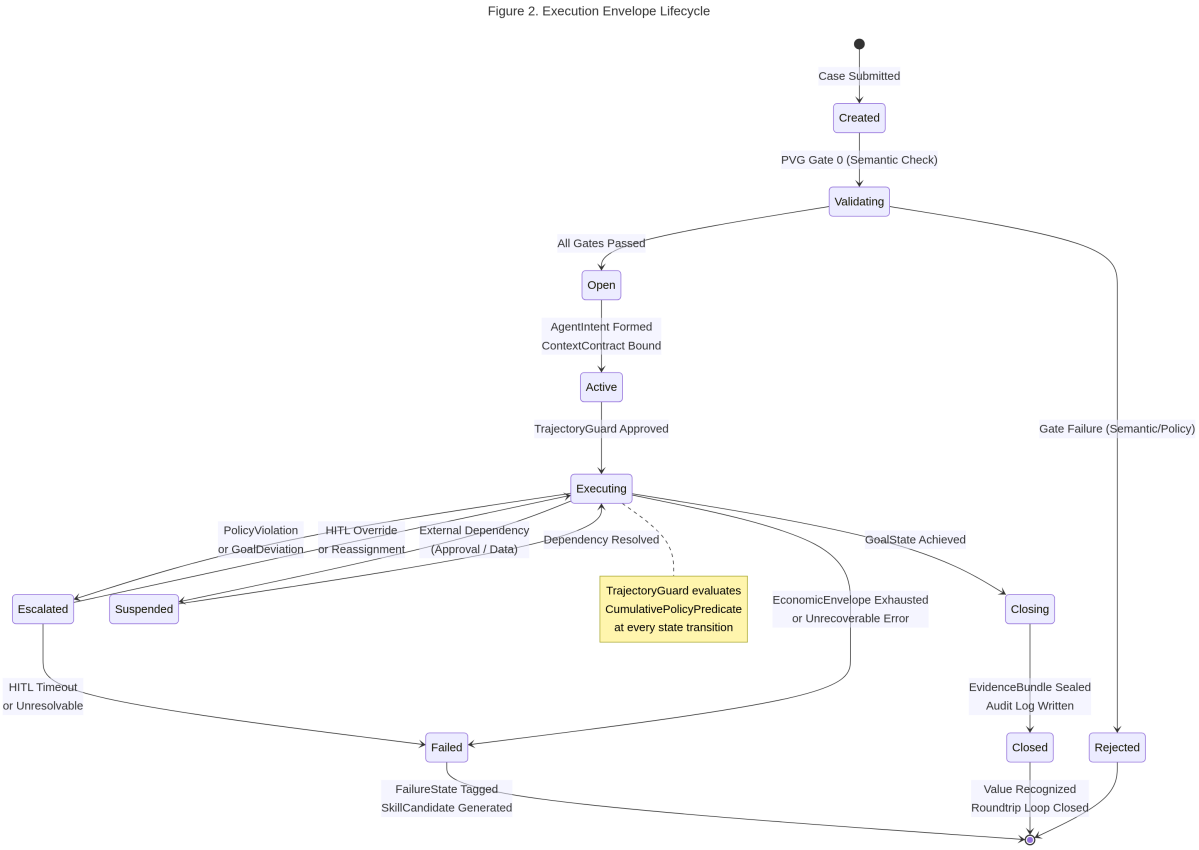
Zone II processes are semi-structured and human-assisted. They involve semi-structured data, human-in-the-loop workflows, moderate exceptions, and medium governance overhead. These are the domain of copilot systems and RAG-augmented assistants. The governance challenge is moderate because the human remains in the loop for consequential decisions.

Zone III processes are the focus of this paper. They are characterized by unstructured and semantically complex data, evidence-bearing obligations, exception-heavy execution, long-running timelines, multi-agent coordination requirements, and full governance exposure. Examples include regulated document approval, complex procurement decisions, clinical trial protocol management, M&A due diligence, and multi-party contract negotiation. These

processes cannot be handled by Zone I or Zone II architectures without unacceptable governance risk.

3. The Execution Envelope

The core architectural innovation of the GPR is the introduction of the **Execution Envelope** as the fundamental unit of computation.



Figure

Figure 2. Execution Envelope Lifecycle

Figure 2. The Execution Envelope Lifecycle, showing all states from Created through Closed or Failed, with the TrajectoryGuard evaluating the CumulativePolicyPredicate at every state transition.

In standard agentic systems, the unit of computation is the LLM call or the agent step. Governance, if present, is applied externally—as a pre-flight check or a post-hoc audit. In the GPR, an agent cannot simply "act." Every action must occur within an Execution Envelope.

The Execution Envelope is a policy-bound, stateful container for agentic execution. It is not a wrapper; it is the execution context itself. It explicitly includes:

- **CaseState**: The business lifecycle position of the case, encoded as a typed finite state machine.
- **ContextContract**: The versioned, policy-bound context admissible for the current step, including a cryptographic provenance chain.
- **PolicyPath**: The allowed trajectories of agent behavior, including path constraints, strategy constraints, and forbidden trajectories.
- **EvidenceBundle**: The cryptographically signed record of all inputs, decisions, and outputs, forming an immutable audit trail.
- **AgentIntent**: The formal goal specification driving the current execution.
- **EconomicEnvelope**: The budget constraints governing the execution, including token budgets, tool call limits, and intervention cost tracking.

Agents operate *inside* the envelope, not outside it. The envelope enforces the boundaries of autonomy. If an agent attempts an action outside the envelope's constraints, the action is blocked, and a governed failure state is triggered. This design is consistent with the principle of typed action contracts for bounded autonomy (Sohail & Haider, 2026) and the cryptographic runtime governance approach of the AEGIS architecture (Mazzocchetti, 2026).

4. PolicyPath and Trajectory-Aware Governance

Action-level policy enforcement cannot detect violations that emerge across a sequence of actions. A Separation of Duties (SoD) violation, for example, requires that the same agent not perform both the drafting and the approval of a document. No single action-level check can detect this; it requires evaluating the cumulative sequence.

The GPR extends the concept of **PolicyPath** from "allowed sequences of actions" to "allowed trajectories of agent behavior toward defined goals." This extension includes:

- **Path Constraints**: Rules governing the sequence of states a case must pass through.
- **Strategy Constraints**: Rules governing the methods an agent may use to achieve a goal.
- **Deviation Handling**: Formal mechanisms for managing situations where an agent deviates from the expected path.
- **Optimization Boundaries**: Limits on how aggressively an agent can optimize for its objective function.
- **Forbidden Trajectories**: Explicitly defined sequences of actions that are never permitted, regardless of the agent's goal.

This introduces **trajectory-aware governance** as a core runtime mechanism. The system evaluates not just the current action, but the entire history of the case and the proposed future trajectory, ensuring that the cumulative behavior remains compliant. This approach is directly supported by recent work on runtime governance policies on paths (Kaptein et al., 2026) and Metric First-Order Temporal Logic (MFOTL) for trajectory compliance verification.

5. The Case Lifecycle State Machine

The GPR formalizes the business lifecycle of every case as a typed state machine. The formal states are:

Table 1

Summary of data

State	Description	Trigger
Open	Case created, AgentIntent formed	User submission or upstream agent
Active	PVG gates passed, agent executing	All validation gates cleared
Suspended	Awaiting external dependency	Approval required, data unavailable
Escalated	Policy violation or goal deviation	TrajectoryGuard or HITL trigger
GoalDeviation	Strategy constraint or trajectory violation	StrategyConstraint violated
Closing	GoalState achieved, sealing evidence	Agent reports success
Closed	EvidenceBundle sealed, audit log written	Evidence integrity verified
Failed	Unrecoverable error or budget exhausted	EconomicEnvelope depleted

Every state transition is governed by a **TransitionGuard** that evaluates the admissibility of the transition against the current PolicyPath and ContextContract. The **TrajectoryGuard** evaluates the cumulative sequence of transitions against the CumulativePolicyPredicate, ensuring that the overall trajectory remains compliant.

6. The Nine New Agentification Primitives

To fully formalize the agentic execution model, the GPR introduces nine new runtime primitives in addition to the 33 primitives identified in the foundational gap analysis. Each is defined as a runtime object with a specific structure, lifecycle, interfaces, and failure modes.

Table 2

Summary of data

Primitive	Type	Description
AgentIntent	Goal Specification	Formal machine-readable specification of the goal an agent is authorized to pursue
ObjectiveFunction	Evaluation Metric	Mathematical or logical formula the agent uses to evaluate its progress
GoalState	Terminal Condition	Formal definition of the conditions under which the AgentIntent is considered achieved
AutonomyLevel	Dynamic Classification	Classification of the agent's current freedom to act (Level 1–5)
DecisionBoundary	Authority Demarcation	Explicit demarcation between autonomous decisions and those requiring HITL
AgentExecutionTrace	Audit Record	Detailed immutable record of the agent's internal reasoning, tool invocations, and state changes
GoalDeviation	Failure State	Formal state triggered when the agent's trajectory diverges significantly from the expected path
StrategyConstraint	Policy Rule	Policy rule restricting the methods an agent can use to achieve its goal
AutonomyEscalationTrigger	Governance Mechanism	Predefined condition that automatically reduces the agent's AutonomyLevel and triggers HITL review

The AutonomyLevel classification (Levels 1–5) draws on established frameworks for AI autonomy governance (Kalia, 2026) and extends them with runtime enforcement mechanisms. Level 1 represents fully supervised execution (every action requires human approval), while Level 5 represents fully autonomous execution within the Execution Envelope (no human approval required unless an AutonomyEscalationTrigger fires).

7. The Progressive Validation Gateway (PVG)

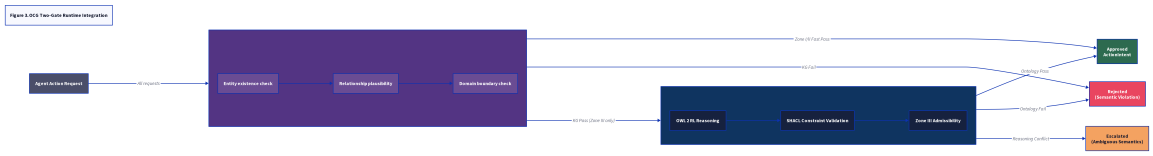
The Progressive Validation Gateway is the entry point for all agentic execution in the GPR. It implements a five-gate validation pipeline that ensures every action is semantically admissible, policy-compliant, and economically bounded before execution begins.

The five gates are:

1. **Gate 1: Intent Parsing** — The raw user or agent input is parsed into a formal AgentIntent. Malformed or ambiguous intents are rejected at this gate.
2. **Gate 2: Knowledge Graph Plausibility** — The AgentIntent is checked against the enterprise knowledge graph (Neo4j) for entity existence and relationship plausibility. This is a fast check (target latency: <20ms) designed to catch obvious semantic errors.
3. **Gate 3: Ontology Admissibility** — For Zone III processes, the AgentIntent is validated against the formal enterprise ontology (OWL 2 RL, Apache Jena). This is a slower but more rigorous check (target latency: <200ms) that ensures the intent is consistent with the enterprise's formal reasoning model.
4. **Gate 4: Policy Enforcement** — The AgentIntent is evaluated against the applicable PolicyPath using the GRAF policy engine (Cedar/OPA). This check enforces action-level policy constraints.
5. **Gate 5: Execution Gate** — The Execution Envelope is instantiated with the validated AgentIntent, ContextContract, PolicyPath, and EconomicEnvelope. The agent is authorized to begin execution.

8. OCG Integration and Semantic Authority

The GPR integrates the Ontological Compliance Gateway (OCG) as its semantic authority. The OCG provides the two-gate runtime integration described in Gates 2 and 3 of the PVG, but its role extends beyond the initial validation.



Figure

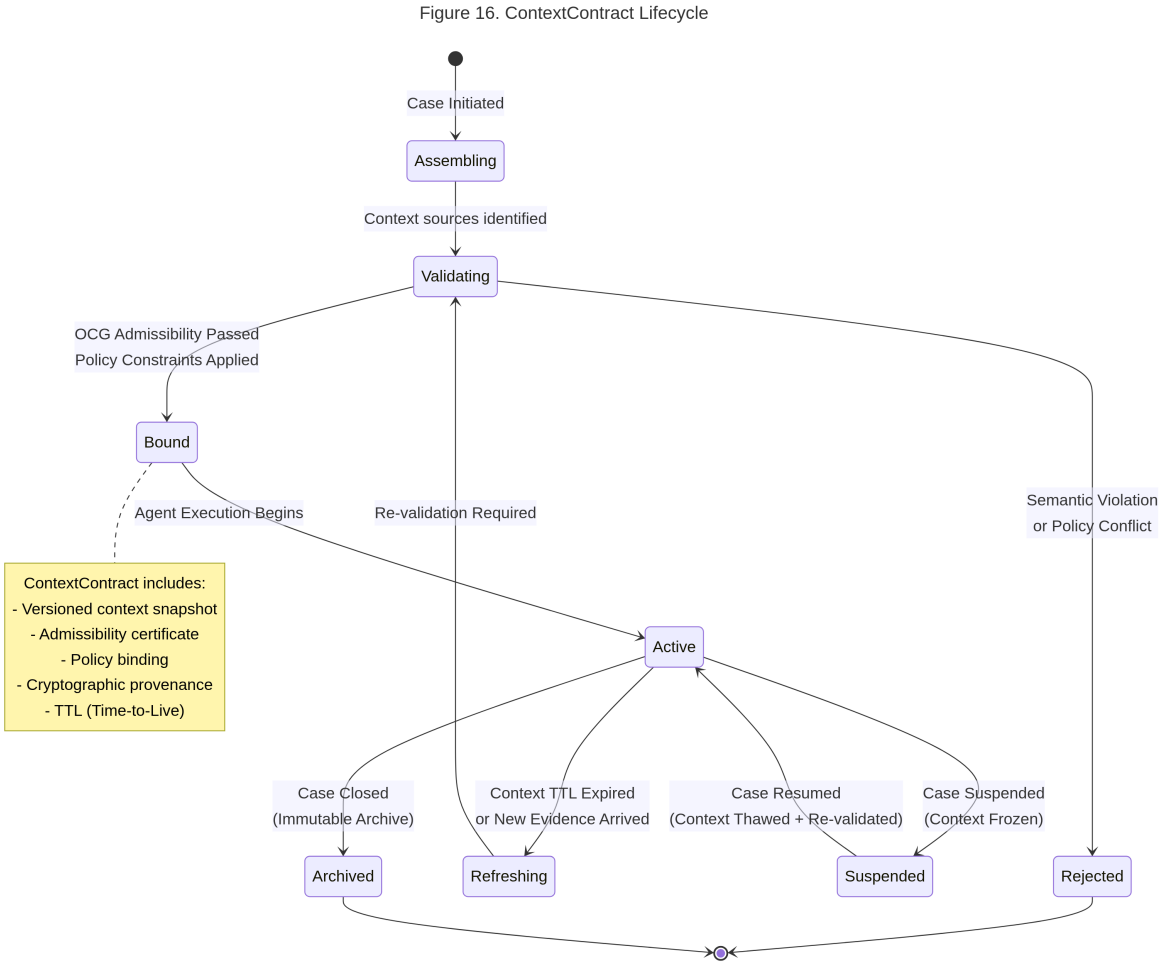
Figure 3. OCG Two-Gate Runtime Integration

Figure 3. The OCG Two-Gate Runtime Integration, showing the fast Knowledge Graph check and the formal Ontology Admissibility check, with approval, rejection, and escalation paths.

The OCG's role in the GPR includes:

- **Ontology-grounded execution**: Agents must justify their actions against the enterprise ontology. Actions that cannot be grounded in the ontology are blocked.
- **Semantic admissibility pipeline**: All inputs and outputs are validated against the ontology at each step of the execution.
- **Ontology compiler pipeline**: Business rules are translated into formal ontology constraints by the OCG's compiler, ensuring that governance policies are semantically grounded.
- **Semantic drift detection lifecycle**: The OCG continuously monitors the ontology for inconsistencies introduced by new data or changing business rules, triggering a SemanticCoherenceMonitor alert when drift is detected.

The ContextContract Lifecycle is tightly coupled with the OCG's semantic validation pipeline.



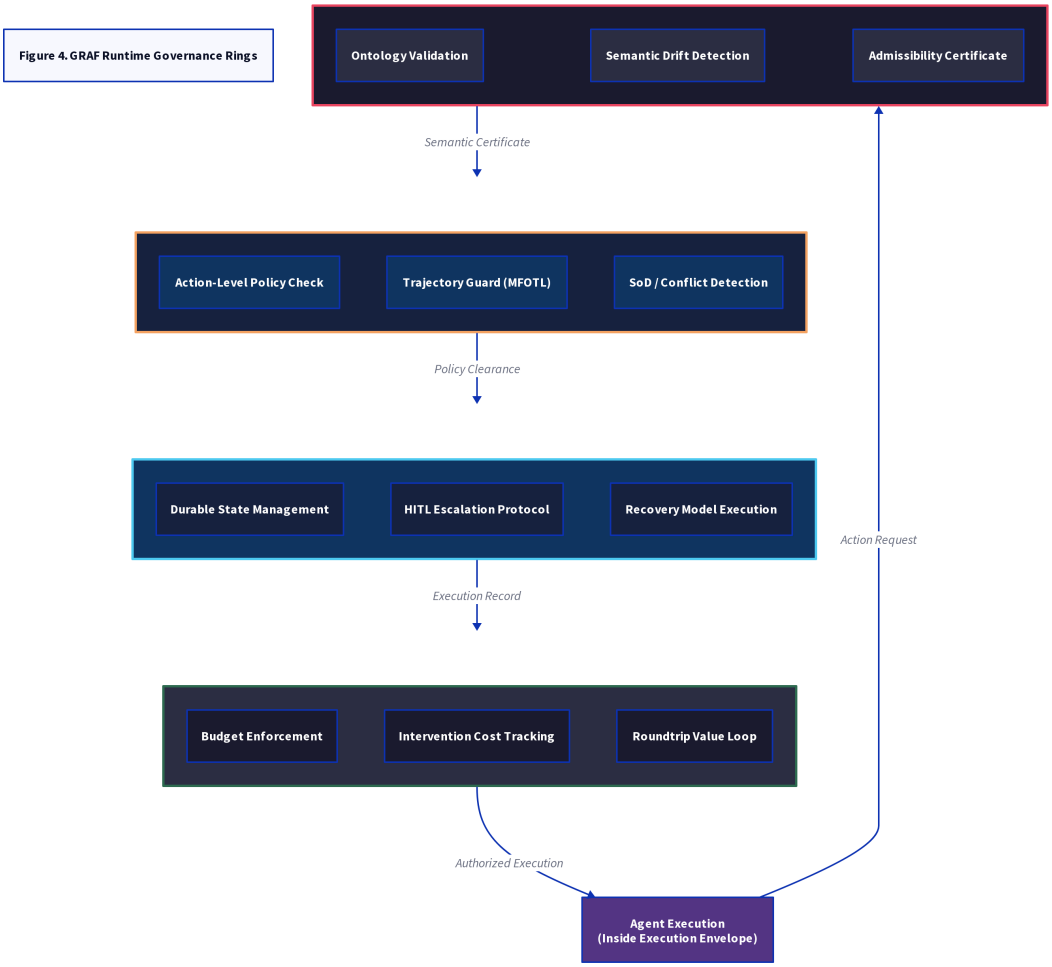
Figure

Figure 16. ContextContract Lifecycle

Figure 16. The ContextContract Lifecycle, showing how context is assembled, validated by the OCG, bound to the Execution Envelope, and archived on case closure.

9. GRAF Governance Rings

The Governance Runtime Assurance Framework (GRAF) provides the policy enforcement layer of the GPR. GRAF implements a four-ring governance model that ensures every aspect of agentic execution is governed.



Figure

Figure 4. GRAF Runtime Governance Rings

Figure 4. The GRAF Runtime Governance Rings, showing the four concentric rings of governance: Semantic Admissibility, Policy Enforcement, Execution Governance, and Economic Governance.

The four rings are:

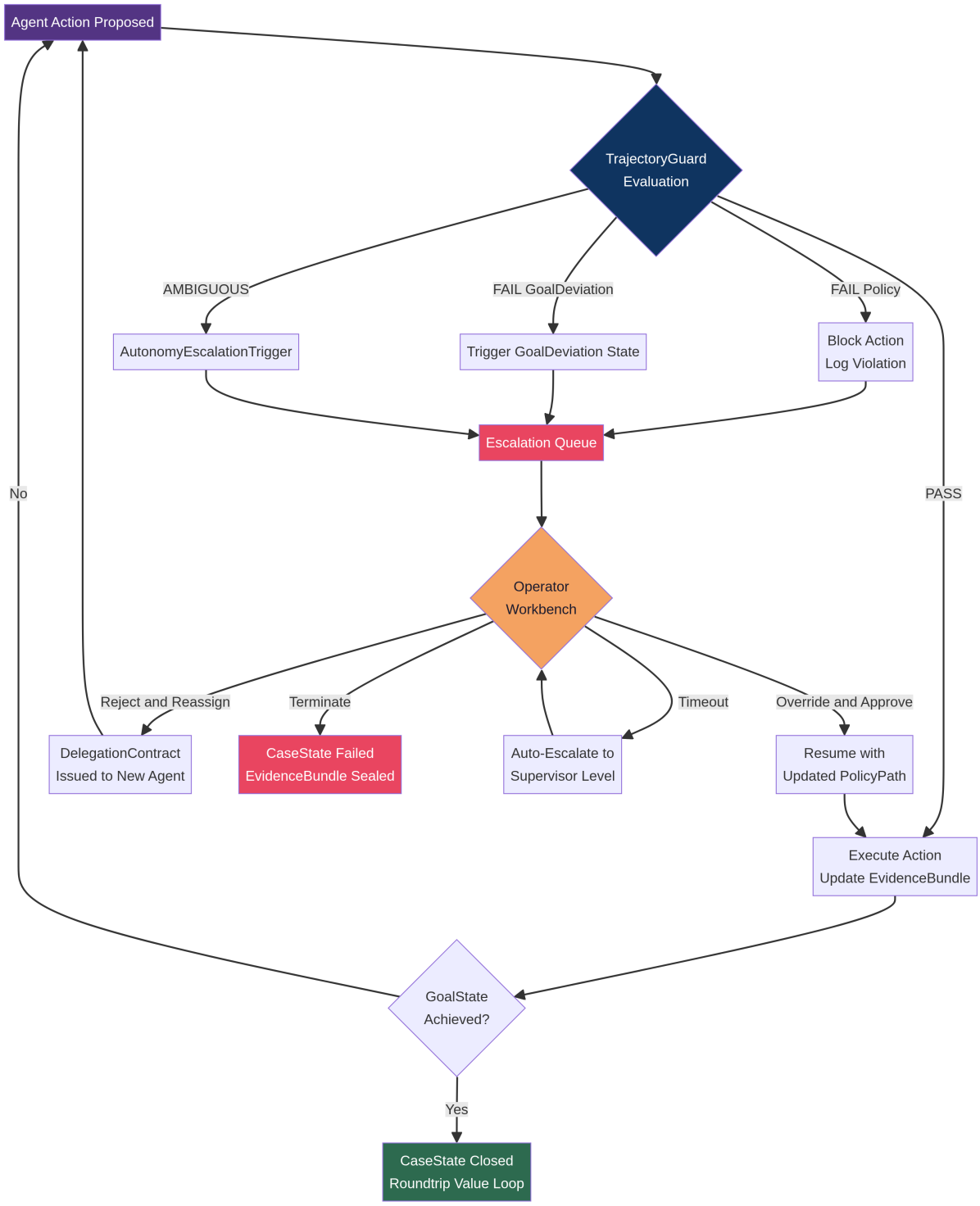
1. **Ring 1: Semantic Admissibility** — Provided by the OCG and the enterprise ontology (OWL 2 RL). Ensures that all agent actions are semantically valid.

2. **Ring 2: Policy Enforcement** — Provided by the Cedar/OPA policy engine and the TrajectoryGuard (MFOTL). Ensures that all agent actions comply with institutional policies, including trajectory-level constraints.
3. **Ring 3: Execution Governance** — Provided by Temporal.io and LangGraph. Ensures that execution is durable, recoverable, and subject to HITL escalation.
4. **Ring 4: Economic Governance** — Provided by the EconomicEnvelope. Ensures that execution remains within budget and that the cost of governance is tracked and optimized.

10. HITL Escalation Protocol

The Human-in-the-Loop (HITL) escalation protocol is a first-class architectural component of the GPR, not an afterthought. The GPR treats human intervention as a governed state transition, not an exception.

Figure 7. HITL Escalation Logic and Intervention Protocol



Figure

Figure 7. HITL Escalation Logic

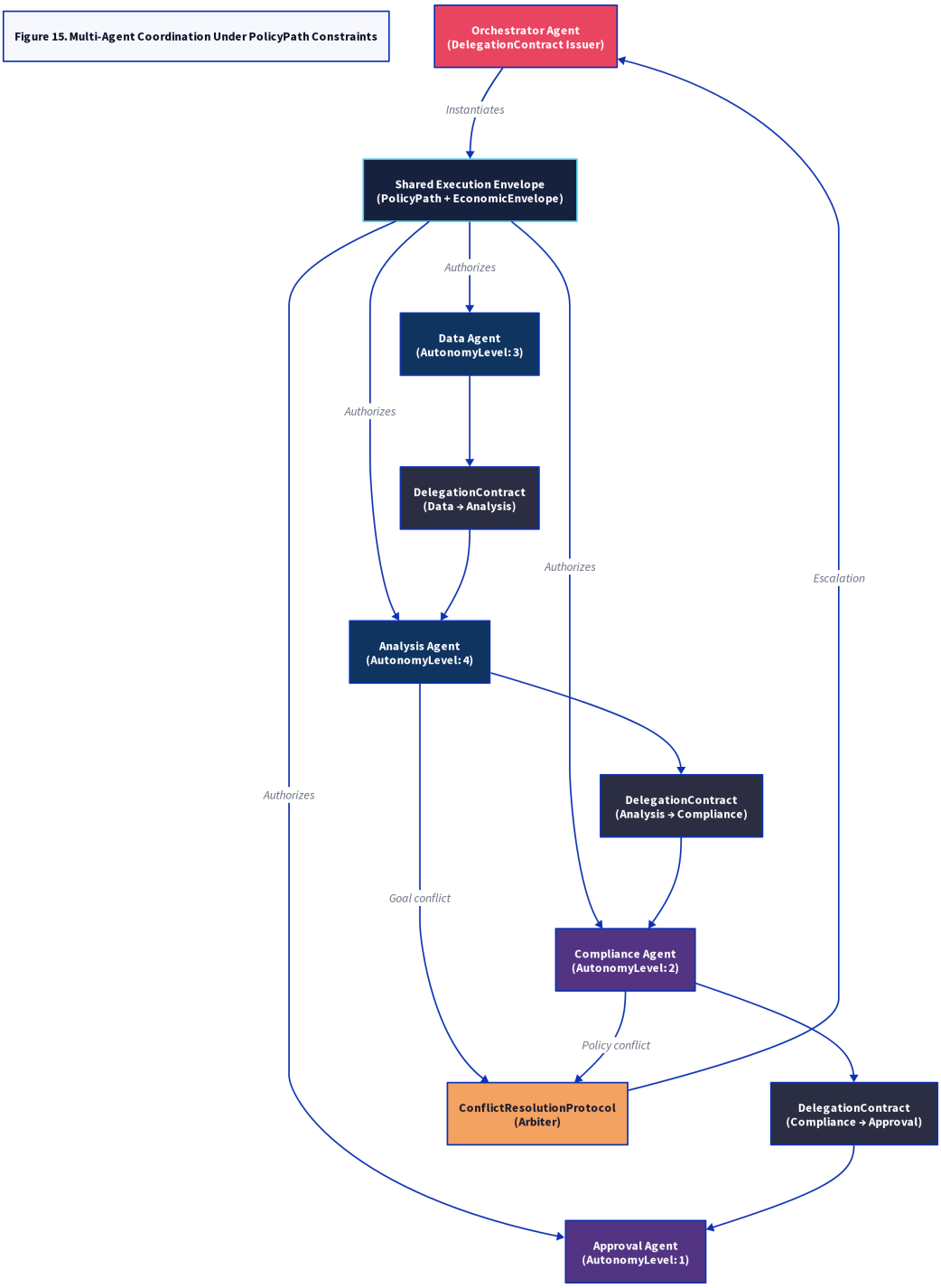
Figure 7. The HITL Escalation Logic and Intervention Protocol, showing the full decision tree from agent action proposal through TrajectoryGuard evaluation, escalation, operator decision, and resolution.

The escalation protocol includes:

- **Escalation triggers**: Policy violations, GoalDeviations, AutonomyEscalationTriggers, and ambiguous semantic states.
- **Escalation queue**: All escalations are queued in the Operator Workbench with full context, including the AgentExecutionTrace and the EvidenceBundle up to the point of escalation.
- **Operator decisions**: Override and approve (resume with updated PolicyPath), reject and reassign (issue a new DelegationContract), terminate (seal the EvidenceBundle and mark the case as Failed), or auto-escalate to a supervisor level on timeout.
- **Intervention burden tracking**: The cost of each HITL intervention is recorded in the EconomicEnvelope and factored into the Roundtrip Value Governance Loop.

11. Multi-Agent Coordination

Multi-agent coordination is not a secondary feature of the GPR; it is a core architectural component. Zone III processes frequently require the collaboration of multiple specialized agents.



Figure

Figure 15. Multi-Agent Coordination Topology

Figure 15. Multi-Agent Coordination Under PolicyPath Constraints, showing the Orchestrator Agent, DelegationContracts, and the ConflictResolutionProtocol.

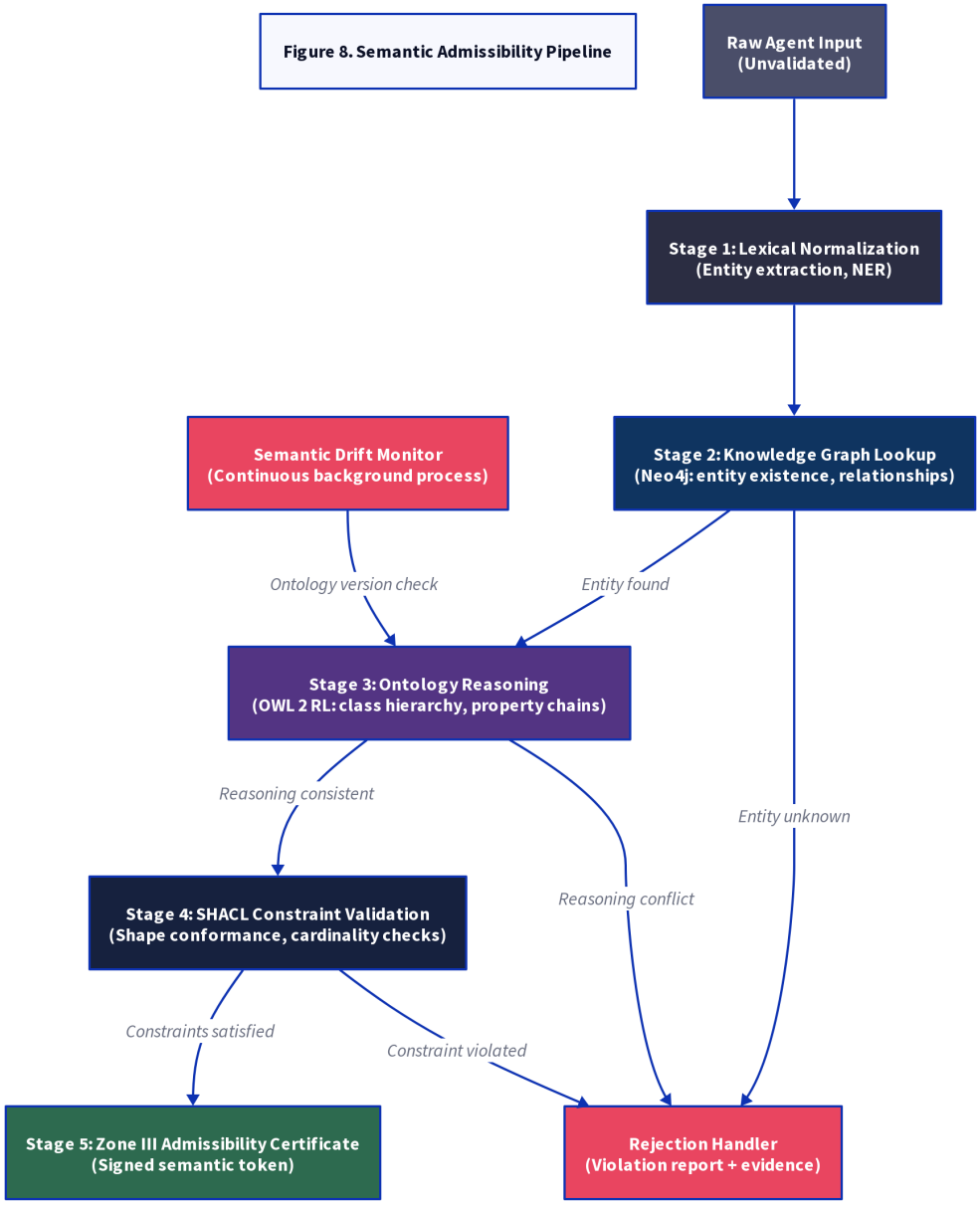
The GPR formalizes multi-agent coordination through:

- **DelegationContract**: The formal transfer of authority from one agent or human to another, specifying the scope of delegation, the duration, and the conditions for revocation.
- **Agent-to-Agent Coordination under PolicyPath Constraints**: All inter-agent communication must occur within the bounds of the shared PolicyPath. Agents cannot delegate authority they do not possess.
- **EconomicEnvelope**: A governance mechanism controlling cost and resource usage across the entire multi-agent execution. The total cost of all agents must remain within the envelope.
- **ConflictResolutionProtocol**: Formal protocols for resolving disagreements between agents, including escalation to the Orchestrator Agent or to a human arbiter.
- **Coordination Failure Modes**: Explicit handling of situations where agents fail to coordinate effectively, including DelegationContract revocation and task reassignment.

This approach is consistent with recent work on safe and policy-compliant multi-agent orchestration for enterprise AI (Pasupuleti et al., 2026) and scalable runtime governance for agentic AI in financial services (Szpruch et al., 2026).

12. Semantic Admissibility Pipeline

The semantic admissibility pipeline is the operational implementation of the OCG's semantic authority within the GPR.



Figure

Figure 8. Semantic Admissibility Pipeline

Figure 8. The Semantic Admissibility Pipeline, showing the five stages from raw agent input through lexical normalization, knowledge graph lookup, ontology reasoning, SHACL constraint validation, and Zone III admissibility certificate issuance.

The pipeline implements five stages of semantic validation:

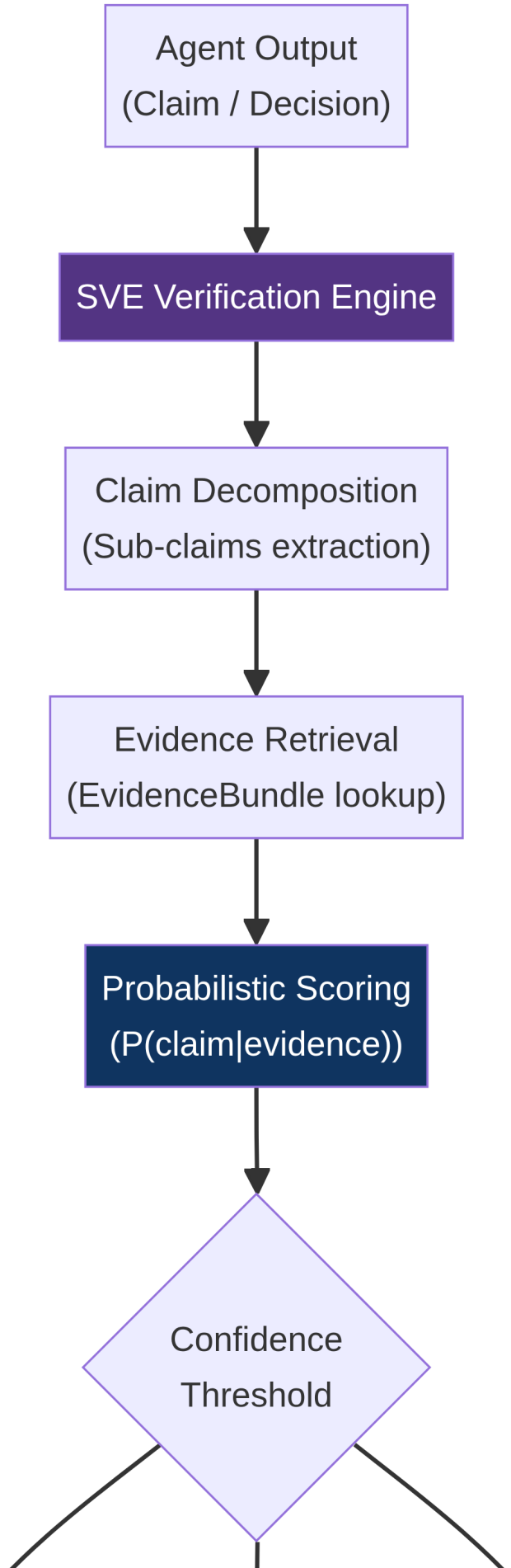
1. **Lexical Normalization**: Entity extraction and Named Entity Recognition (NER) to normalize the raw input into a structured form.
2. **Knowledge Graph Lookup**: Fast entity existence and relationship plausibility checks against the enterprise knowledge graph (Neo4j/Qdrant).
3. **Ontology Reasoning**: OWL 2 RL reasoning using Apache Jena to check class hierarchy consistency and property chain validity.
4. **SHACL Constraint Validation**: Shape conformance and cardinality checks against the enterprise SHACL shapes graph.
5. **Zone III Admissibility Certificate**: A signed semantic token confirming that the input is admissible for Zone III execution.

This pipeline is grounded in W3C standards for OWL 2 (W3C OWL Working Group, 2012) and SHACL (W3C Shapes Working Group, 2017), and is consistent with recent work on ontology-constrained neural reasoning in enterprise agentic systems (Tuan & Sanyal, 2026).

13. SVE Probabilistic Verification

The Sovereign Verification Engine (SVE) provides probabilistic verification of agent outputs within the GPR. Unlike deterministic rule-based checks, the SVE evaluates the probability that a given claim or decision is correct given the available evidence.

Figure 13. SVE Probabilistic Verification Integration



Figure

Figure 13. SVE Probabilistic Verification

Figure 13. The SVE Probabilistic Verification Integration, showing the claim decomposition, evidence retrieval, probabilistic scoring, and the three-way confidence threshold decision.

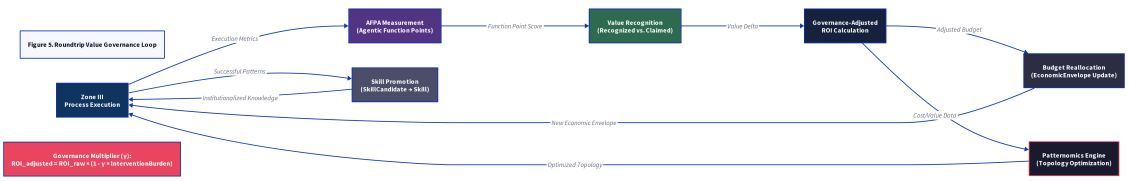
The SVE implements a three-threshold confidence model:

- **P ≥ 0.95**: Verified output. The claim is signed with a verification certificate and added to the EvidenceBundle.
- **0.70 ≤ P < 0.95**: Conditional approval. The claim is flagged for human review in the Operator Workbench.
- **P < 0.70**: Rejected output. A GoalDeviation is triggered and the agent is rolled back to the last verified state.

This approach is consistent with recent work on runtime verification of AI agents (Koohestani, 2025) and formal methods in the agentic AI era (Masood, 2026).

14. Roundtrip Value Governance Loop

The GPR introduces a formal economic model for agentic execution. This is not about tokenomics in the cryptocurrency sense, but about internal enterprise resource allocation and value measurement.



Figure

Figure 5. Roundtrip Value Governance Loop

Figure 5. The Roundtrip Value Governance Loop, showing how execution metrics flow through AFPA measurement, value recognition, governance-adjusted ROI calculation, budget reallocation, and Patternomics optimization.

The Roundtrip Value Governance Loop includes:

- **AFPA Measurement**: Agentic Function Points (AFP) are calculated for each execution, providing a standardized measure of the work performed by the agent.
- **Value Recognition**: The recognized value of the execution is calculated as the difference between the value delivered and the cost incurred.
- **Governance-Adjusted ROI**: The ROI is adjusted by the governance multiplier (γ), which accounts for the overhead introduced by governance mechanisms and the cost of HITL interventions: `ROI_adjusted = ROI_raw × (1 - γ × InterventionBurdenIndex)`.
- **Budget Reallocation**: The adjusted ROI informs the next cycle's budget allocation, ensuring that resources are directed toward the most valuable processes.
- **Patternomics Optimization**: The Patternomics engine analyzes the execution patterns and optimizes the topology for efficiency and reliability.

15. Patternomics and Topology Optimization

The GPR incorporates Patternomics to optimize the execution topology of multi-agent systems.

Figure 6. Patternomics: Topology-Dependent Error Amplification

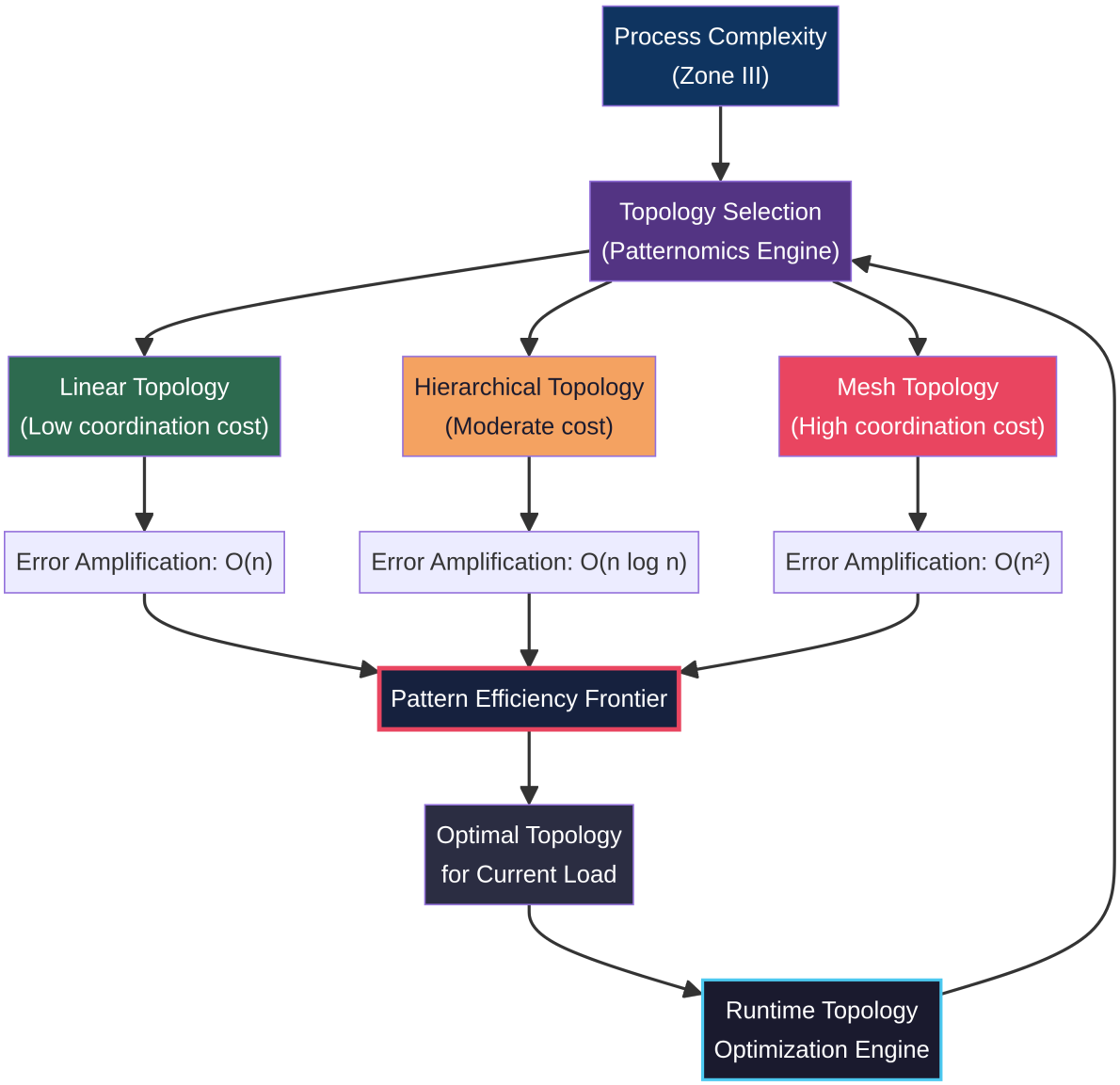


Figure 6. Patternomics

Topology-Dependent Error Amplification

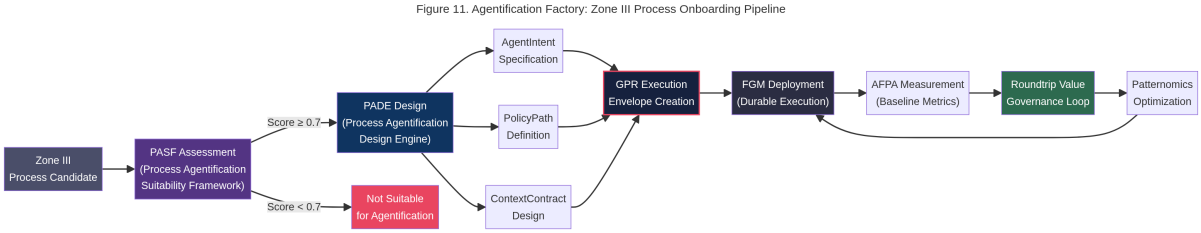
Figure 6. Patternomics: Topology-Dependent Error Amplification, showing how error propagation scales differently across linear (O(n)), hierarchical (O(n log n)), and mesh ($O(n^2)$) topologies.

Key Patternomics concepts include:

- **Pattern Efficiency Frontier**: The optimal balance between coordination cost and execution speed for a given process complexity level.
- **Topology-dependent error amplification**: Errors propagate differently across linear (O(n)), hierarchical (O(n log n)), and mesh ($O(n^2)$) topologies. The Patternomics engine selects the topology that minimizes error amplification for the current process.
- **Coordination cost accumulation**: The cost of inter-agent communication is tracked and factored into the EconomicEnvelope.
- **Runtime topology optimization engine**: The GPR dynamically adjusts the execution topology based on real-time performance data from the Roundtrip Value Governance Loop.

16. Agentification Factory Integration

The GPR is designed to be the runtime target of the Agentification Factory — the process by which Zone III processes are identified, assessed, designed, and deployed as governed agentic workflows.



Figure

Figure 11. Agentification Factory Pipeline

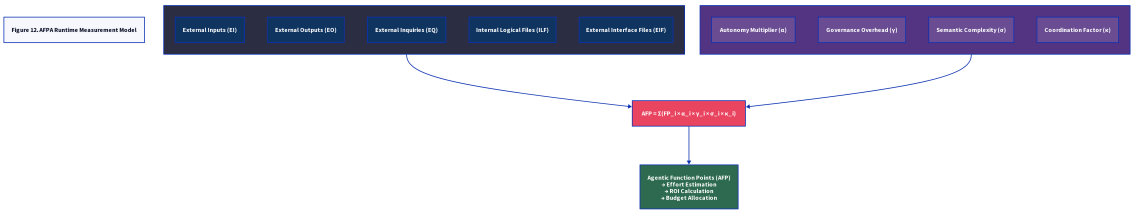
Figure 11. The Agentification Factory: Zone III Process Onboarding Pipeline, showing the PASF assessment, PADE design, GPR Execution Envelope creation, FGM deployment, AFPA measurement, and Patternomics optimization.

The Agentification Factory pipeline includes:

1. **PASF Assessment**: The Process Agentification Suitability Framework (PASF) evaluates whether a process is suitable for Zone III agentification, producing a suitability score (0–1). Processes scoring ≥ 0.7 proceed to design.
2. **PADE Design**: The Process Agentification Design Engine (PADE) produces the formal AgentIntent specification, PolicyPath definition, and ContextContract design for the process.
3. **GPR Execution Envelope Creation**: The GPR instantiates an Execution Envelope for the process based on the PADE output.
4. **FGM Deployment**: The FGM deploys the process as a durable agentic workflow using Temporal.io and LangGraph.
5. **AFPA Measurement**: Baseline metrics are established for the process using the AFPA model.
6. **Roundtrip Value Governance Loop**: The process enters the continuous improvement cycle.

17. AFPA Runtime Measurement Model

The Agentic Function Point Analysis (AFPA) model provides a standardized measure of the work performed by agentic systems, enabling consistent ROI calculation and budget allocation.



Figure

Figure 12. AFPA Runtime Measurement Model

Figure 12. The AFPA Runtime Measurement Model, showing the five input categories (EI, EO, EQ, ILF, EIF), the four agentic complexity weights (Autonomy Multiplier, Governance Overhead, Semantic Complexity, Coordination Factor), and the AFP formula.

The AFPA model extends the traditional Function Point Analysis (FPA) methodology with four agentic complexity weights:

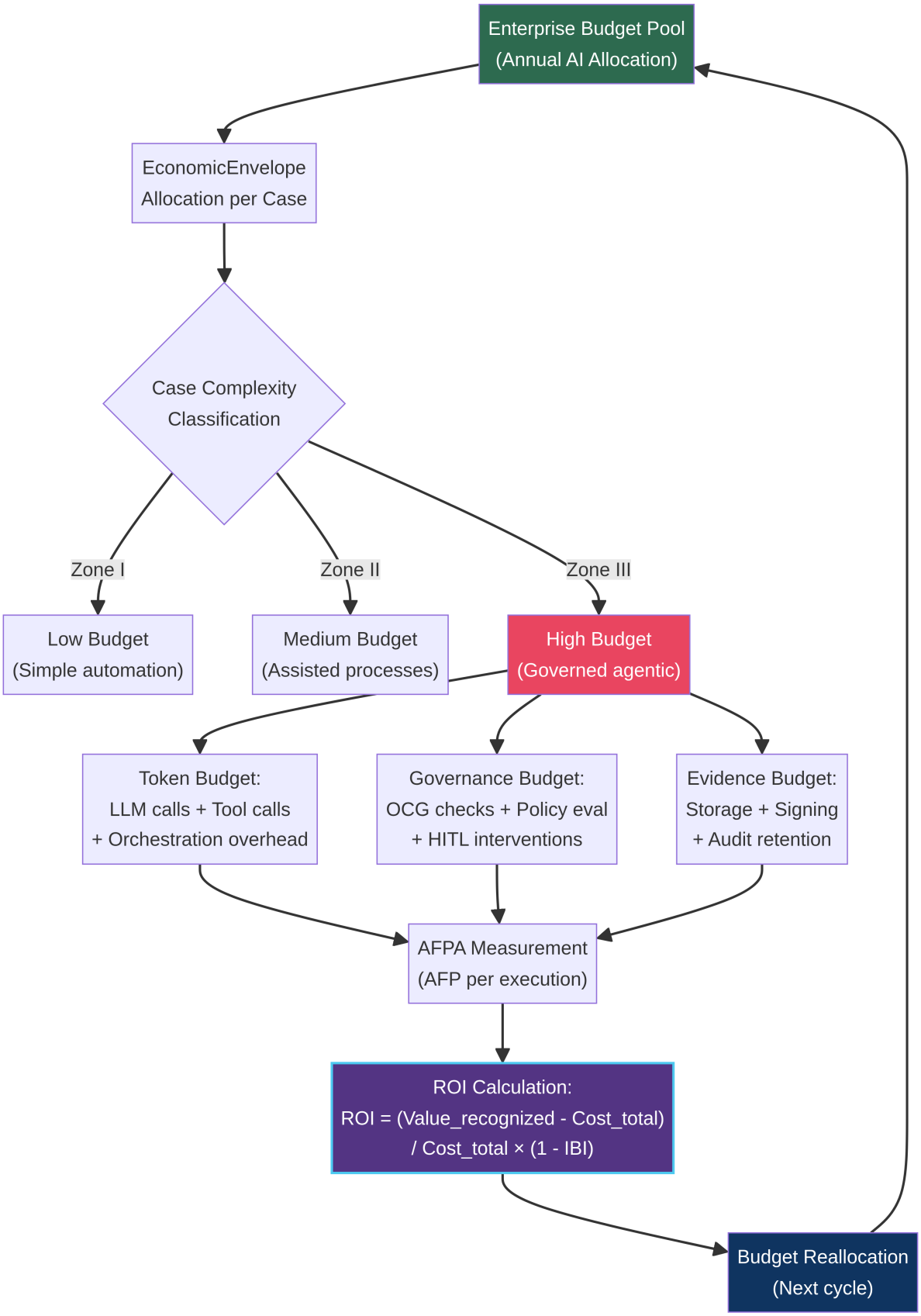
- **Autonomy Multiplier (α)**: Reflects the degree of autonomous decision-making required.
- **Governance Overhead (γ)**: Reflects the cost of governance mechanisms (OCG checks, policy evaluations, HITL interventions).
- **Semantic Complexity (σ)**: Reflects the complexity of the semantic validation required.
- **Coordination Factor (κ)**: Reflects the coordination overhead of multi-agent execution.

The AFP formula is: `AFP = Σ(FP_i × α_i × γ_i × σ_i × κ_i)`

18. Agentic Tokenomics: Internal Enterprise Resource Allocation

The GPR introduces a formal internal economic model for agentic execution, which we term **Agentic Tokenomics** to distinguish it from cryptocurrency-based tokenomics.

Figure 18. Agentic Tokenomics: Internal Enterprise Resource Allocation



Figure

Figure 18. Agentic Tokenomics Economic Model

Figure 18. The Agentic Tokenomics Economic Model, showing the budget allocation cascade from the Enterprise Budget Pool through Zone classification, token/governance/ evidence budgets, AFPA measurement, ROI calculation, and budget reallocation.

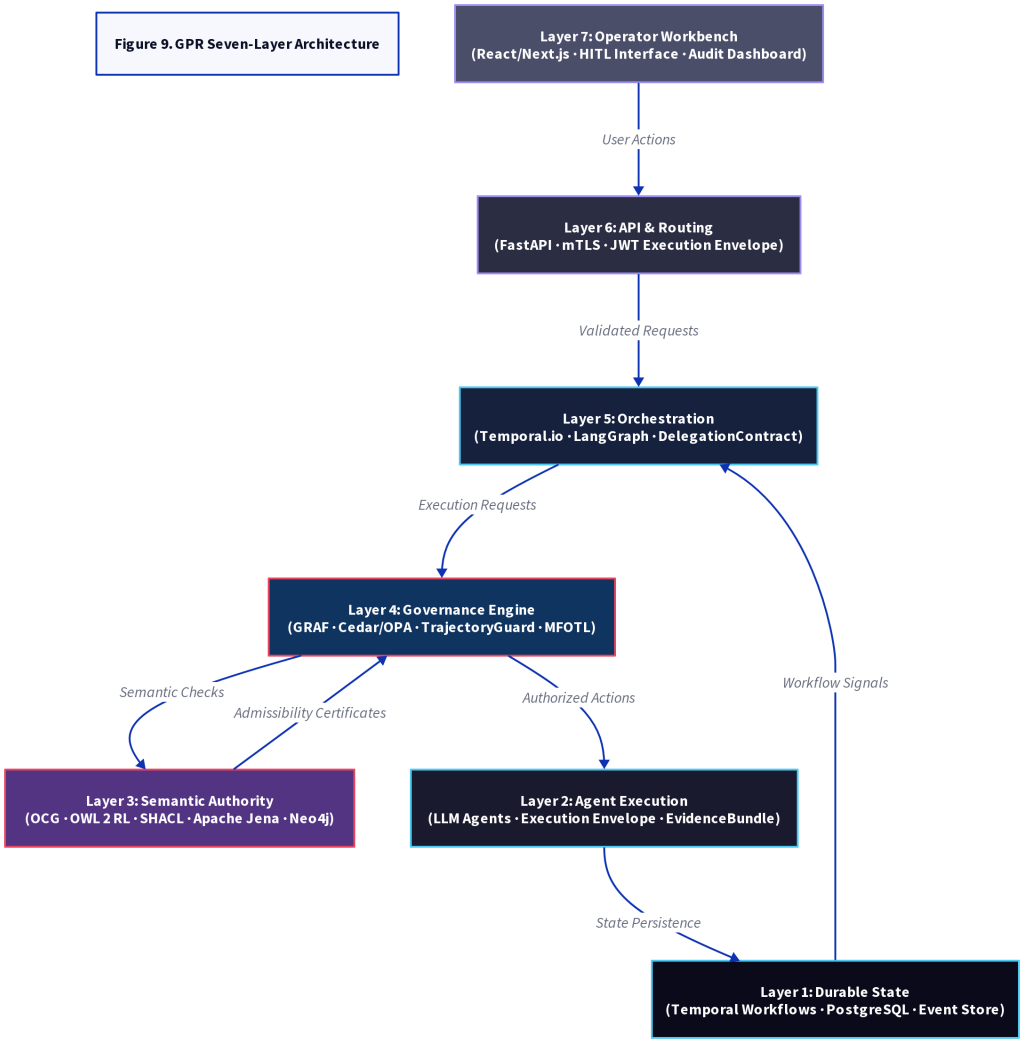
The economic model includes three budget categories for Zone III execution:

1. **Token Budget**: The budget for LLM calls, tool calls, and orchestration overhead.
2. **Governance Budget**: The budget for OCG checks, policy evaluations, and HITL interventions.
3. **Evidence Budget**: The budget for storage, signing, and audit retention of the EvidenceBundle.

The total cost of a Zone III execution is the sum of these three budgets, and the ROI is calculated as: `ROI = (Value_recognized - Cost_total) / Cost_total × (1 - IBI)` where IBI is the Intervention Burden Index.

19. The GPR Seven-Layer Architecture

The GPR implements a seven-layer architecture that separates concerns across the full stack from user interaction to durable state persistence.



Figure

Figure 9. GPR Seven-Layer Architecture

Figure 9. The GPR Seven-Layer Architecture, showing the seven layers from the Operator Workbench (Layer 7) through the Durable State layer (Layer 1), with the communication protocols between layers.

The seven layers are:

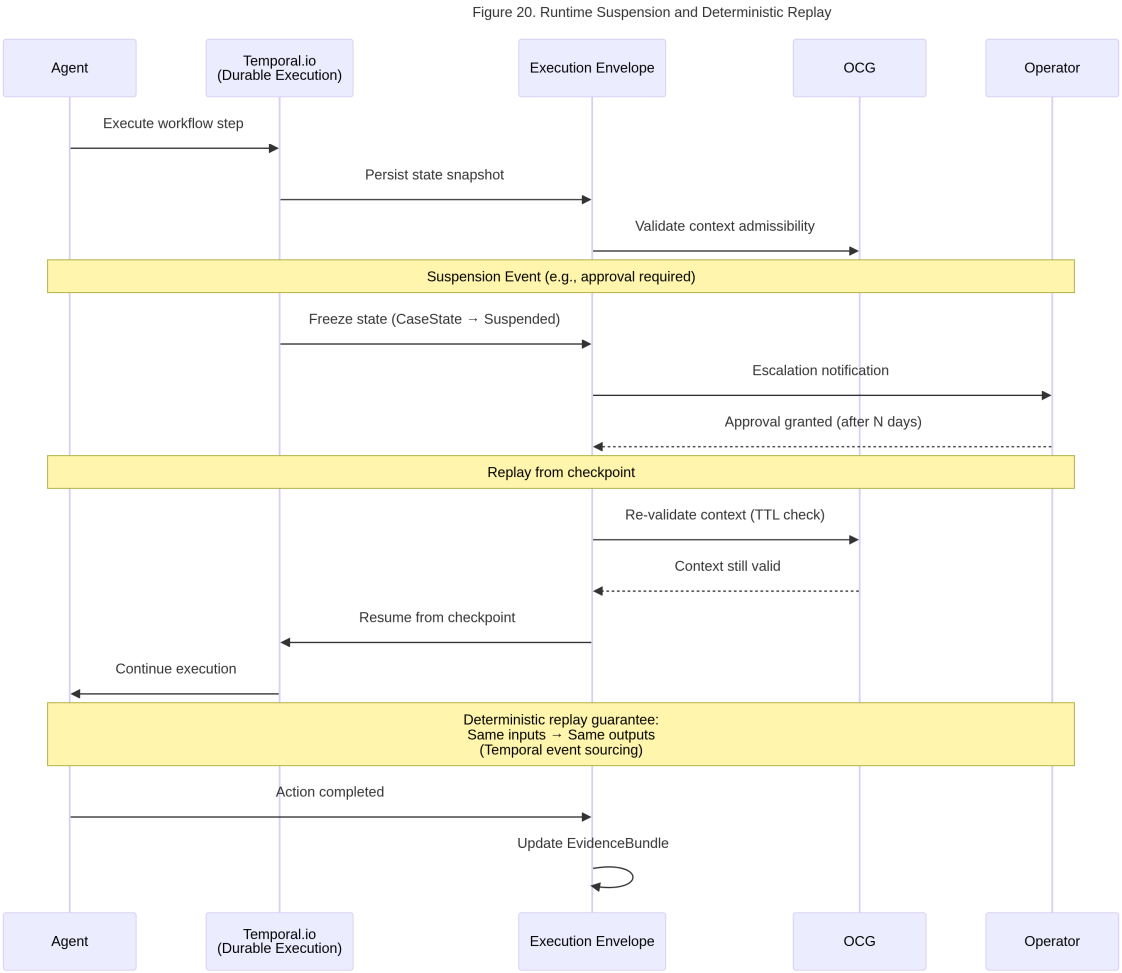
Table 3

Summary of data

Layer	Name	Technologies
7	Operator Workbench	React/Next.js, HITL Interface, Audit Dashboard
6	API & Routing	FastAPI, mTLS, JWT Execution Envelope
5	Orchestration	Temporal.io, LangGraph, DelegationContract
4	Governance Engine	GRAF, Cedar/OPA, TrajectoryGuard, MFOTL
3	Semantic Authority	OCG, OWL 2 RL, SHACL, Apache Jena, Neo4j
2	Agent Execution	LLM Agents, Execution Envelope, EvidenceBundle
1	Durable State	Temporal Workflows, PostgreSQL, Event Store

20. Runtime Suspension and Deterministic Replay

One of the most critical capabilities of the GPR for Zone III processes is the ability to suspend execution, persist state, and resume deterministically after an arbitrary period of time.



Figure

Figure 20. Runtime Suspension and Deterministic Replay

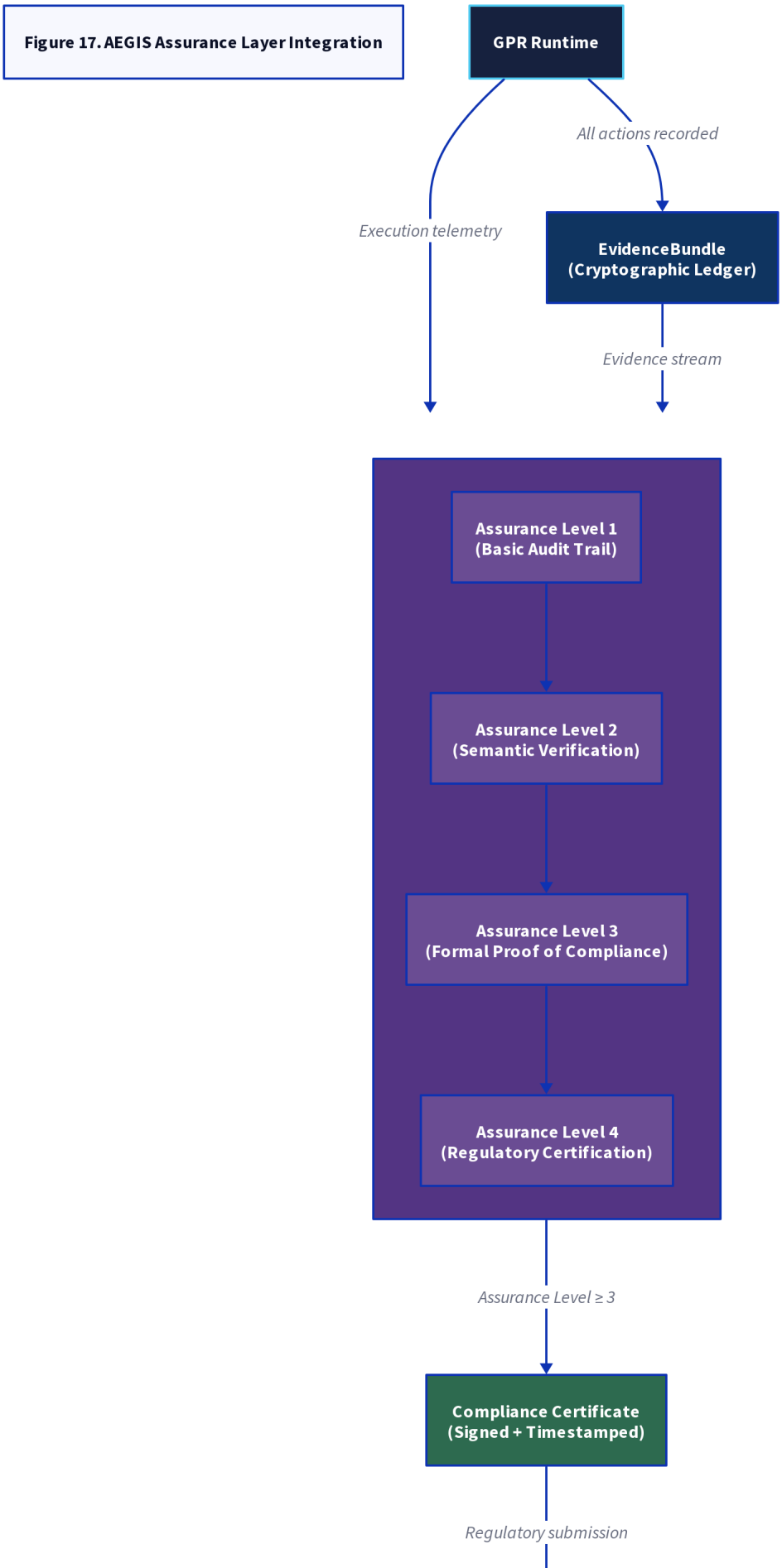
Figure 20. Runtime Suspension and Deterministic Replay, showing the sequence of events from agent execution through state freezing, escalation, approval, context re-validation, and deterministic replay from checkpoint.

The GPR implements deterministic replay through Temporal.io's event sourcing model. Every state change is recorded as an immutable event in the event store. When execution is resumed after a suspension, the system replays all events from the last checkpoint, ensuring that the same inputs produce the same outputs. The ContextContract is re-validated by the OCG before resumption to ensure that the context is still admissible.

21. AEGIS Assurance Integration

The GPR integrates the AEGIS assurance layer to provide formal compliance certification for regulated industries.

Figure 17. AEGIS Assurance Layer Integration



Figure

Figure 17. AEGIS Assurance Layer Integration

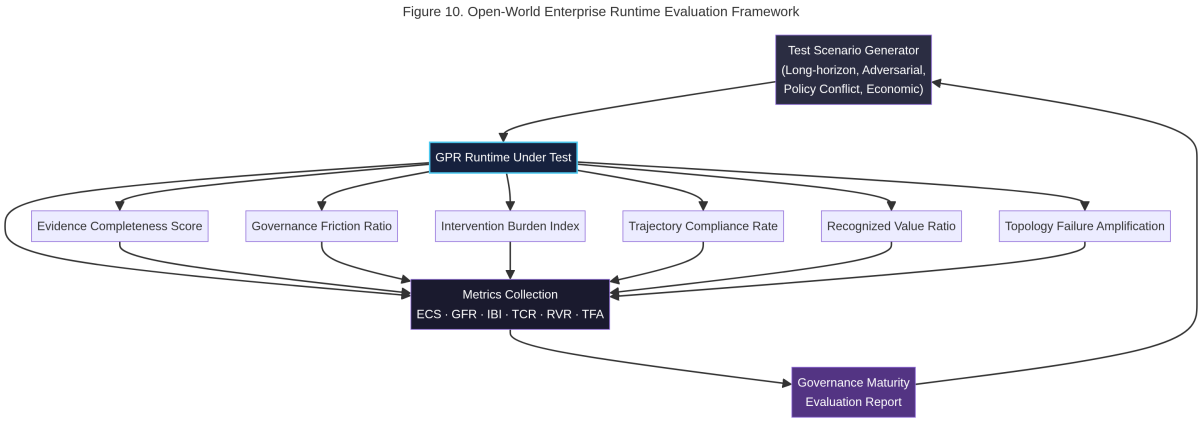
Figure 17. The AEGIS Assurance Layer Integration, showing the four assurance levels from basic audit trail through regulatory certification, and the interface with external regulators.

AEGIS provides four assurance levels:

1. **Assurance Level 1**: Basic audit trail — the EvidenceBundle is maintained and accessible.
2. **Assurance Level 2**: Semantic verification — all agent outputs are verified by the SVE.
3. **Assurance Level 3**: Formal proof of compliance — the TrajectoryGuard provides a formal proof that the execution trajectory complied with all PolicyPath constraints.
4. **Assurance Level 4**: Regulatory certification — the compliance certificate is signed and submitted to external regulators.

22. Open-World Enterprise Runtime Evaluation Framework

The GPR requires a new approach to evaluation. Standard benchmarks are insufficient for Zone III processes.



Figure

Figure 10. Open-World Evaluation Framework

Figure 10. The Open-World Enterprise Runtime Evaluation Framework, showing the test scenario generator, the GPR runtime under test, the metrics collection layer, and the governance maturity evaluation report.

The open-world evaluation framework includes twelve evaluation dimensions:

Table 4

Summary of data

Metric	Abbreviation	Description
Evidence Completeness Score	ECS	Measures the quality and completeness of the generated EvidenceBundle
Governance Friction Ratio	GFR	Quantifies the overhead introduced by governance mechanisms
Intervention Burden Index	IBI	Tracks the cost and frequency of human interventions
Trajectory Compliance Rate	TCR	Measures the percentage of executions that comply with all PolicyPath constraints
Recognized Value Ratio	RVR	Measures the ratio of recognized value to claimed value
Topology Failure Amplification	TFA	Tests how failures propagate through different execution topologies
Long-Horizon Trajectory Score	LHTS	Evaluates agent behavior over extended periods (>100 steps)
Interruption Recovery Score	IRS	Tests the system's ability to recover from interruptions
Semantic Resilience Score	SRS	Tests the system's resilience to adversarial ontology injection
Policy Conflict Resolution Rate	PCRR	Evaluates how the system resolves conflicting rules
Replay Verification Score	RVS	Ensures that execution traces can be accurately replayed
Governance Maturity Index	GMI	Composite score across all dimensions

23. Mathematical and Formal Models

The GPR formalizes its core concepts using mathematical models that provide a rigorous foundation for the runtime's behavior and enable formal verification.

23.1 AFPA Equations

$$AFP = \Sigma(FP_i \times \alpha_i \times \gamma_i \times \sigma_i \times \kappa_i)$$

where FP_i is the base function point count for component i, α_i is the autonomy multiplier, γ_i is the governance overhead factor, σ_i is the semantic complexity factor, and κ_i is the coordination factor.

23.2 Governance-Adjusted ROI

$$ROI_{adjusted} = (Value_{recognized} - Cost_{total}) / Cost_{total} \times (1 - \gamma \times IBI)$$

where γ is the governance multiplier ($0 \leq \gamma \leq 1$) and IBI is the Intervention Burden Index ($0 \leq IBI \leq 1$).

23.3 Trajectory Compliance Predicate

The CumulativePolicyPredicate P is defined over the execution trace $T = (a_1, a_2, ..., a_n)$ as:

$$P(T) = \forall i \in [1,n]: \varphi(a_i) \wedge \psi(a_{i-1}, a_i) \wedge \chi(T[1..i])$$

where $\varphi(a_i)$ is the action-level policy predicate, $\psi(a_{i-1}, a_i)$ is the transition-level policy predicate, and $\chi(T[1..i])$ is the trajectory-level policy predicate evaluated over the full history.

23.4 Economic Boundedness

The EconomicEnvelope enforces the constraint:

$$Cost_{total}(T) \leq Budget_{allocated} \times (1 + \varepsilon)$$

where ε is the allowed budget overrun tolerance (default: 0.05). If this constraint is violated, the execution is suspended and an AutonomyEscalationTrigger is fired.

24. Implementation and Deployment

The GPR is designed for sovereign enterprise deployment. The reference implementation relies on:

- **LangGraph** for agent orchestration and workflow definition
- **Temporal.io** for durable execution and state management
- **Cedar (AWS)** or **OPA** for policy enforcement
- **Apache Jena** for ontology reasoning (OWL 2 RL)
- **Neo4j** for the enterprise knowledge graph
- **PostgreSQL** for relational data and the event store
- **React/Next.js** for the Operator Workbench

The deployment model includes three options:

1. **Sovereign**: Fully on-premises or private cloud. The enterprise retains complete control over all data and execution.
2. **Managed**: Hosted by a trusted provider. The enterprise retains control over policies and ontologies but delegates infrastructure management.
3. **Hybrid**: A mix of on-premises and cloud components. Sensitive data remains on-premises; compute-intensive tasks are offloaded to the cloud.

The GPR is designed for incremental adoption, following a four-phase implementation roadmap:

1. **Phase 1: Observability** (Months 1–3): Deploy in shadow mode to monitor existing processes without intervening.
2. **Phase 2: Bounded Autonomy** (Months 4–6): Enable active governance for low-risk processes.
3. **Phase 3: Trajectory Governance** (Months 7–9): Implement full trajectory-aware governance for complex processes.
4. **Phase 4: Economic Optimization** (Months 10–12): Activate the Roundtrip Value Governance Loop and Patternomics engine.

25. Limitations and Future Work

The GPR architecture has several important limitations that must be acknowledged.

Empirical validation: The GPR is a design-science contribution. The formal models and architectural claims have not yet been empirically validated in production deployments. Future work should include controlled experiments measuring the governance friction ratio, intervention burden index, and trajectory compliance rate in real-world Zone III processes.

Ontology maintenance: The GPR's semantic authority depends on a well-maintained enterprise ontology. Ontology maintenance is a significant ongoing cost that is not fully addressed in this paper. Future work should address automated ontology evolution and conflict resolution.

LLM non-determinism: The GPR assumes that LLM outputs can be deterministically replayed given the same inputs. In practice, LLM outputs are stochastic. Future work should address how to handle LLM non-determinism in the context of deterministic replay.

Scalability: The formal ontology reasoning required by Gate 3 of the PVG introduces latency that may be unacceptable for high-throughput processes. Future work should address ontology reasoning optimization and caching strategies.

26. Conclusion

The Governed Process Runtime (GPR) represents a fundamental shift in enterprise AI architecture. It moves the focus from model intelligence to governed execution. By formalizing the Execution Envelope, PolicyPath, ContextContract, and the nine new agentification primitives, the GPR provides a rigorous foundation for automating consequential Zone III processes.

The GPR is not a marginal improvement over existing agentic frameworks. It is a new architectural class that addresses the fundamental governance gaps identified in the EIP-FGM analysis: the absence of formal business-level case state machines, the lack of trajectory-

aware policy enforcement, the absence of continuous semantic coherence monitoring, the lack of governed learning protocols, and the absence of multi-agent economic control.

The primary obstacles to production-grade enterprise agentic AI are not AI model limitations. They are distributed systems engineering challenges: formal state management, trajectory-aware policy enforcement, continuous semantic coherence monitoring, governed learning protocols, and multi-agent economic control. The GPR provides the architectural blueprint for solving these challenges.

References

Bandi, A., Kongari, B., Naguru, R., Pasnoor, S., & Vilipala, S. V. (2025). The rise of agentic AI: A review of definitions, frameworks, architectures, applications, evaluation metrics, and challenges. *Future Internet, 17*(9), 404. https://doi.org/10.3390/fi17090404

Berente, N., Rossi, F., Bevilacqua, M., Ganapini, M., Goehring, B., & Domin, H. (2024). *Assessing ROI of AI ethics and governance initiatives*. IBM Institute for Business Value. https://www.ibm.com/thought-leadership/institute-business-value/en-us/report/roi-ai-ethics

Casovan, A., Jones, J., & Chaudhry, U. (2024). *AI governance in practice report 2024*. IAPP and FTI Consulting. https://iapp.org/resources/article/ai-governance-in-practice-report

Chen, C., Gong, X., Liu, Z., Jiang, W., Goh, S. Q., & Lam, K.-Y. (2024). Trustworthy, responsible, and safe AI: A comprehensive architectural framework for AI safety with challenges and mitigations. *arXiv preprint arXiv:2408.12935*. https://arxiv.org/abs/2408.12935

DeLong, L. N., Mir, R. F., & Fleuriot, J. D. (2023). Neurosymbolic AI for reasoning over knowledge graphs: A survey. *arXiv preprint arXiv:2302.07200*. https://doi.org/10.48550/arXiv.2302.07200

Eisenberg, I. W., Gamboa, L., & Sherman, E. (2025). The unified control framework: Establishing a common foundation for enterprise AI governance, risk management

and regulatory compliance. *arXiv preprint arXiv:2503.05937*. https://arxiv.org/abs/2503.05937

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design science in information systems research. *MIS Quarterly*, *28*(1), 75–105. https://doi.org/10.2307/25148625

International Organization for Standardization. (2023). *ISO/IEC 42001:2023 — Information technology — Artificial intelligence — Management system*. ISO. https://www.iso.org/standard/80062.html

Jackson, F. (2025). Designing a policy engine for agentic AI systems: From governance requirements to runtime enforcement. *SSRN Electronic Journal*. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=5904104

Jackson, F. (2025). Governing autonomous AI agents with policy-as-code: A multi-layer architecture for risk, compliance, and zero-trust control. *SSRN Electronic Journal*. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=5820262

Kalia, S. (2026). AI autonomy governance: A governance framework for agentic AI enabling safe, accountable, and scalable autonomous intelligence. *TM Forum Inform*. https://inform.tmforum.org/features-and-opinion/ai-autonomy-governance-a-governance-framework-for-agentic-ai-enabling-safe-accountable-and-scalable-autonomous-intelligence

Kaptein, M., Khan, V. J., & Podstavnychy, A. (2026). Runtime governance for AI agents: Policies on paths. *arXiv preprint arXiv:2603.16586*. https://arxiv.org/abs/2603.16586

Koohestani, R. (2025). AgentGuard: Runtime verification of AI agents. *arXiv preprint arXiv:2509.23864*. https://arxiv.org/abs/2509.23864

Masood, A. (2026). Formal methods in the agentic AI era: A strategic agenda for high-assurance software. *Medium*. https://medium.com/@adnanmasood/formal-methods-in-the-agentic-ai-era-a-strategic-agenda-for-high-assurance-software-8f91bb6f6c25

Mazzocchetti, A. M. (2026). Cryptographic runtime governance for autonomous AI systems: The Aegis architecture for verifiable policy enforcement. *arXiv preprint arXiv:2603.16938*. https://doi.org/10.48550/arXiv.2603.16938

Mei, L., Yao, J., Ge, Y., Wang, Y., Bi, B., Cai, Y., Liu, J., Li, M., Li, Z.-Z., Zhang, D., Zhou, C., Mao, J., Xia, T., Guo, J., & Liu, S. (2025). A survey of context engineering for large language models. *arXiv preprint arXiv:2507.13334*. https://arxiv.org/abs/2507.13334

Meroño-Peñuela, A., Simperl, E., Kurteva, A., & Reklos, I. (2025). KG.GOV: Knowledge graphs as the backbone of data governance in AI. *Journal of Web Semantics*, *85*, 100847. https://doi.org/10.1016/j.websem.2024.100847

Myla, C. K. R., & Vyas, K. (2025). Orchestrating autonomy: Patterns, protocols, and governance for enterprise agentic AI. *TechRxiv*. https://www.techrxiv.org/doi/full/10.36227/techrxiv.176238026.60445463

National Institute of Standards and Technology. (2023). *Artificial intelligence risk management framework (AI RMF 1.0)*. U.S. Department of Commerce. https://doi.org/10.6028/NIST.AI.100-1

OpenAI. (2024). *Practices for governing agentic AI systems*. OpenAI Technical Report. https://cdn.openai.com/papers/practices-for-governing-agentic-ai-systems.pdf

Pasupuleti, V., Allala, S. R., Bayyavarapu, S. R. K. V., & Tyagi, S. (2026). Safe and policy-compliant multi-agent orchestration for enterprise AI. *arXiv preprint arXiv:2604.17240*. https://arxiv.org/abs/2604.17240

Pery, A., Rafiei, M., Simon, M., & van der Aalst, W. M. P. (2021). Trustworthy artificial intelligence and process mining: Challenges and opportunities. *arXiv preprint arXiv:2110.02707*. https://arxiv.org/abs/2110.02707

Piazza, M. (2026). Operational AI fabric: A unified runtime architecture for governed enterprise AI execution. *SSRN*. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=6596058

Saini, S. (2026). Governing the agentic enterprise: A new operating model for autonomous AI at scale. *California Management Review*. https://cmr.berkeley.edu/2026/03/governing-the-agentic-enterprise-a-new-operating-model-for-autonomous-ai-at-scale/

Sapkota, R., & Myla, C. K. R. (2025). LangChain vs. LangGraph vs. LangSmith: Taxonomies of agentic AI toolchains for end-to-end orchestration. *TechRxiv*. https://www.techrxiv.org/doi/full/10.36227/techrxiv.175695645.52670060

Sohail, S., & Haider, G. (2026). Bounded autonomy for enterprise AI: Typed action contracts and consumer-side execution. *arXiv preprint arXiv:2604.14723*. https://arxiv.org/pdf/2604.14723

Szpruch, L., Sudjianto, A., Bhatti, T., & Ang, G. (2026). Scalable runtime governance for agentic AI in financial services. *SSRN*. https://papers.ssrn.com/sol3/papers.cfm?abstract_id=6567199

Tuan, T. L., & Sanyal, A. (2026). Ontology-constrained neural reasoning in enterprise agentic systems: A neurosymbolic architecture for domain-grounded AI agents. *arXiv preprint arXiv:2604.00555*. https://doi.org/10.48550/arXiv.2604.00555

van Hurne, M., Valk, J., de Koning, H., & van der Laan, V. (2026). The ontological compliance gateway: A neuro-symbolic architecture for verifiable agentic AI. *Eigenvector Research*. https://marcovanhurnedaily-blacd.wordpress.com/wp-content/uploads/2026/03/ocg_paper_arxiv_final-1.pdf

Venkiteela, P. (2026). An enterprise agentic architecture framework for agentic AI governance and scalable autonomy. *Scientific Journal of Computer Science*, *2*(1), 17. https://doi.org/10.64539/sjcs.v2i1.2026.368

W3C OWL Working Group. (2012). *OWL 2 web ontology language document overview (second edition)*. W3C Recommendation. https://www.w3.org/TR/owl2-overview/

W3C Shapes Working Group. (2017). *Shapes constraint language (SHACL)*. W3C Recommendation. https://www.w3.org/TR/shacl/

Wang, C. L., Singhal, T., Kelkar, A., & Tuo, J. (2025). MI9: An integrated runtime governance framework for agentic AI. *arXiv preprint arXiv:2508.03858*. https://arxiv.org/abs/2508.03858

Zhu, H., Liang, J., Hou, M., Tang, R., Zhu, X., Yang, J., Mao, Y., & Wu, F. (2026). From business events to auditable decisions: Ontology-governed graph simulation for enterprise AI. *arXiv preprint arXiv:2604.08603*. https://arxiv.org/abs/2604.08603

Author Note. This paper is a design-science contribution (Hevner et al., 2004). All architectural claims are grounded in the Eigenvector Research corpus and the external academic literature cited above. No empirical data was fabricated. The 42 runtime primitives specified in this paper represent the authors' formal architectural specification, not empirically validated measurements.

Correspondence concerning this article should be addressed to Eigenvector Research Institute.

Appendix A: Execution Flows and End-to-End Scenarios

This appendix details the runtime execution flows for common Zone III scenarios, demonstrating how the GPR primitives interact in practice.

A.1 Regulated Document Approval Flow

1. **Initialization**: A user submits a document for approval. The GPR creates a new `CaseState` (Open) and instantiates an `ExecutionEnvelope`.
2. **Intent Parsing**: The `AgentIntent` is parsed from the user request ("Approve this document").
3. **Semantic Admissibility (Gate 0)**: The OCG validates the document against the enterprise ontology. If valid, the `CaseState` transitions to Active.
4. **Policy Evaluation**: The `TrajectoryGuard` evaluates the `PolicyPath`. It identifies a Separation of Duties (SoD) constraint: the approver cannot be the author.
5. **Agent Execution**: The agent attempts to approve the document. The `TransitionGuard` checks the agent's identity against the SoD constraint.
6. **Governance Intervention**: If the agent is the author, the action is blocked. The `CaseState` transitions to Escalated, and an `AutonomyEscalationTrigger` alerts a human supervisor.

7. **Resolution**: The supervisor reassigns the approval task. The agent completes the approval. The `EvidenceBundle` is sealed, and the `CaseState` transitions to Closed.

Appendix B: Formal Python Specifications

This appendix provides formal Python specifications (using Pydantic) for the core GPR primitives.

```
 from pydantic import BaseModel, Field
from typing import List, Optional, Dict, Any
from enum import Enum
      class CaseStateEnum(str, Enum):
    OPEN = "Open"
    ACTIVE = "Active"
    SUSPENDED = "Suspended"
    ESCALATED = "Escalated"
    GOAL_DEVIATION = "GoalDeviation"
    CLOSED = "Closed"
    FAILED = "Failed"

      class AgentIntent(BaseModel):
    goal_description: str
    target_entity: str
    authorized_actions: List[str]

       class ExecutionEnvelope(BaseModel):
    case_id: str
    state: CaseStateEnum
    intent: AgentIntent
    policy_path_id: str
    context_contract_id: str
    evidence_bundle_id: str
```

Appendix C: Conceptual Architecture Diagrams

(Diagrams are embedded in the main text. This section provides detailed descriptions of the architectural layers.)

1. **Semantic Authority Layer**: The OCG and enterprise ontology.
2. **Governance Layer**: The GRAF policy engine and trajectory guards.

3. **Orchestration Layer**: LangGraph and Temporal for durable execution.
4. **Agent Execution Layer**: The LLMs and agent logic operating within the Execution Envelope.
5. **Evidence Layer**: The immutable ledger of all actions and decisions.

Appendix D: Failure Taxonomy and Recovery Models

The GPR defines a formal taxonomy of failure modes and corresponding recovery models.

Table 5

Summary of data

Failure Mode	Description	Recovery Model
Semantic Drift	The ontology becomes inconsistent with reality.	`OntologyVersionGuard` triggers a manual review.
Policy Conflict	Two policies dictate mutually exclusive actions.	`ConflictResolutionProtocol` escalates to a human arbiter.
Economic Exhaustion	The agent exceeds its budget.	`EconomicEnvelope` freezes execution and requests additional funds.
Coordination Failure	Agents fail to reach consensus.	`DelegationContract` revokes authority and reassigns the task.

Appendix E: PolicyPath DSL and YAML Examples

The GPR uses a Domain-Specific Language (DSL) based on YAML to define PolicyPaths.

```
policy_path:
  id: "pp-doc-approval-001"
  name: "Standard Document Approval"
  constraints:
    - type: "SeparationOfDuties"
      description: "Author cannot be approver"
      enforcement: "Strict"
  allowed_trajectories:
    - [Draft, Review, Approve, Publish]
    - [Draft, Review, Reject, Draft]
  forbidden_trajectories:
    - [Draft, Approve] # Skips review
```

Appendix F: JSON Schemas for Core Primitives

This appendix provides the JSON schemas for the `EvidenceBundle` and `DelegationContract`.

```
{
  "$schema": "http://json-schema.org/draft-07/schema#",
  "title": "EvidenceBundle",
  "type": "object",
  "properties": {
    "bundle_id": { "type": "string" },
    "case_id": { "type": "string" },
    "events": {
      "type": "array",
      "items": {
        "type": "object",
        "properties": {
          "timestamp": { "type": "string", "format": "date-
time" },
          "actor": { "type": "string" },
          "action": { "type": "string" },
          "result": { "type": "string" }
        }
      }
    },
    "cryptographic_signature": { "type": "string" }
  }
}
```

Appendix G: Implementation Roadmap

The GPR is designed for phased adoption.

1. **Phase 1: Observability (Months 1-3)**: Deploy the GPR in "shadow mode" to monitor existing processes without intervening. Establish the baseline ontology and policy paths.
2. **Phase 2: Bounded Autonomy (Months 4-6)**: Enable active governance for low-risk processes. Agents operate within strict Execution Envelopes with frequent HITL escalation.
3. **Phase 3: Trajectory Governance (Months 7-9)**: Implement full trajectory-aware governance for complex, multi-step processes.
4. **Phase 4: Economic Optimization (Months 10-12)**: Activate the Roundtrip Value Governance loop and Patternomics engine to optimize execution topologies and minimize costs.

Appendix H: Technical Architecture

The technical architecture of the GPR reference implementation consists of:

- **Frontend**: React/Next.js for the Operator Workbench.
- **API Gateway**: FastAPI for routing and initial validation.
- **Orchestration**: Temporal for durable execution and state management.
- **Agent Framework**: LangGraph for defining agent workflows.
- **Policy Engine**: Cedar (AWS) or OPA for evaluating PolicyPaths.
- **Semantic Engine**: Apache Jena for ontology reasoning.
- **Data Store**: PostgreSQL for relational data, Neo4j for the knowledge graph.

Appendix I: Reference Implementation Details

The reference implementation uses a microservices architecture deployed on Kubernetes. Communication between services is secured using mutual TLS (mTLS). The `ExecutionEnvelope` is passed as a signed JWT token in the header of every request.

Appendix J: Developer Guide

J.1 Repository Structure

```
gpr-monorepo/
├── apps/
│   └── operator-workbench/ # React/Next.js frontend
├── validation-gateway/ # FastAPI service
├── orchestrator/ # Temporal/LangGraph workers
├── packages/
│   ├── gpr-core/ # Shared Pydantic models
│   ├── gpr-policy/ # Cedar policy definitions
│   └── gpr-ontology/ # OWL/SHACL definitions
└── infrastructure/ # Kubernetes manifests
```

J.2 Local Development Setup

Prerequisites include Docker, Docker Compose, Python 3.11+, and Node.js 20+. Run `docker-compose up -d` to start the local Temporal cluster, Neo4j, Apache Jena, and Keycloak.

Appendix K: Code Examples

K.1 FastAPI Validation Gateway

```
from fastapi import FastAPI, HTTPException
from pydantic import BaseModel
import requests
      app = FastAPI()

      class ActionIntent(BaseModel):
    actor: str
    action_type: str
    target_entity: str
    risk_classification: str

      @app.post("/api/v1/validate")
async def validate_intent(intent: ActionIntent):
    if intent.risk_classification == "Zone III":
        # Simplified semantic check
        response = requests.post("http://jena:3030/ds/sparql",
data={"query": "ASK { ... }"})
        if not response.json().get("boolean", False):
            raise HTTPException(status_code=403,
detail="Semantic validation failed")
    return {"status": "PASSED"}
```

Appendix L: Patent Analysis

L.1 Novelty Claims

The GPR architecture introduces several novel mechanisms:

1. **The Execution Envelope**: A policy-bound, stateful container that wraps agentic execution, enforcing trajectory-aware governance and semantic admissibility at runtime.
2. **Trajectory-Aware Governance (PolicyPath)**: Evaluating cumulative sequences of agent actions against formal policy predicates to detect violations that action-level checks cannot identify.

3. **Dual-Layer Semantic Validation**: A staged pipeline using a lightweight knowledge graph for fast operational checks and a formal ontology for strict Zone III admissibility validation.

L.2 Intellectual Property Strategy

The IP strategy focuses on protecting the core architectural mechanisms (Execution Envelope, PolicyPath, Dual-Layer Semantics) while maintaining an open-stack realization path. This allows the enterprise to retain sovereign control over the runtime while leveraging open-source components for implementation.