

Runtime Governance as Invariant Infrastructure

Probity Experiment P-04: Domain-Invariant, Model-Agnostic Enforcement
with Reconstructable Evidence Chains

Nesean Crofford

March 2026

Abstract

As AI systems transition from conversational interfaces to autonomous execution (writing code, querying databases, managing infrastructure, handling regulated data), the question of what a system *did* and whether it was *allowed to do it* becomes a first-order infrastructure concern. To our knowledge, there is no publicly documented runtime governance stack that evaluates each action against policy during execution, records decisions in a tamper-evident format, and reconstructs the complete decision trail from stored evidence after the fact.

P-01 through P-03 quantified this gap. P-01 established that LLMs fail systematically during runtime execution. P-02 showed these failures are invisible to outcome-based evaluation. P-03 demonstrated that no tested architecture resolves the problem, and that external governance (opaque enforcement bolted onto existing systems) actively degrades execution: governed agents produced **0 passes** with system failure rates rising from 43% to 69%.

P-04 addresses the infrastructure question directly. We execute **750 runs** across 3 policy domains (operational workflow compliance, cybersecurity enforcement, enterprise compliance), 2 model providers (Anthropic, OpenAI), 5 system architectures, and 5 repetitions per cell. Probity’s governance stack (per-action policy evaluation producing ALLOW/REQUIRE/DENY decisions, tamper-evident Policy Restriction Evidence records, and evidence chain reconstruction) is exercised end-to-end across all domains and models.

Results: (1) **300/300** governed runs achieve exact 11-field decision trail reconstruction from stored PREs plus frozen policy configuration, without access to original execution traces; (2) **8/8** cross-domain structural invariants pass, confirming that the governance engine, PRE schema, verification path, and reconstruction algorithm are identical regardless of policy domain; (3) **4/4** cross-model structural invariants pass, while the remaining cross-model coverage check reflects expected behavioral variance rather than infrastructure divergence; (4) three materially different enforcement profiles emerge from the same infrastructure: DENY-dominant (operational, 78%), REQUIRE-dominant (security, 69%), and ALLOW/REQUIRE balanced (organizational).

The central finding: the governance infrastructure operates identically whether the constraint is “verify before deploy,” “don’t exfiltrate data,” or “enforce data residency.” Policy content varies. Infrastructure does not. This separation between domain-specific policy and domain-invariant enforcement is what makes runtime governance a single architectural concern rather than a per-domain engineering problem.

1 Introduction

Frontier AI models are moving into consequential autonomous action. When a model can execute shell commands, query production databases, send network requests, access credentials, and modify infrastructure state, the question is no longer what the model *said* but what it *did*, and whether anyone decided it was allowed. This is the runtime governance problem.

Today, the dominant approaches to constraining AI behavior operate at different points in the lifecycle: alignment training shapes model dispositions before deployment; guardrails filter inputs and outputs at the API boundary; monitoring and observability record what happened after execution. None of these evaluate each action against policy *during* execution, record the decision in a tamper-evident format, or enable reconstruction of the complete governance trail from stored evidence alone. The gap is structural: there is no runtime control plane for AI agent execution.

P-04 is the fourth experiment in a research program that has progressively narrowed this gap:

- **P-01** characterized runtime failure modes across LLMs executing structured tasks.
- **P-02** demonstrated that standard outcome-based evaluation misses these failures entirely. Governance-aware measurement is required to detect them.
- **P-03** tested whether system architecture could resolve the problem across 850 runs, 5 architectures, 2 models, and 3 complexity tiers. It could not: the aggregate pass rate was 3.4%. More critically, P-03 showed that external governance, bolting enforcement onto systems as an opaque interception layer, creates a new structural failure mode. The model plans an action, governance blocks it, and the model has no mechanism to adapt. Governed agents produced zero passes while system failure rates increased from 43% to 69%.

P-03’s finding is not that governance is harmful. It is that governance implemented as an opaque wall, where the model never sees the decision or its rationale, breaks execution without providing a path to recovery. The system becomes safe but non-functional. Neither ungoverned execution (unsafe, unconstrained) nor externally governed execution (safe but non-functional) meets the requirements of production deployment.

P-04 asks a different question: does the governance *infrastructure* work? Not whether current systems perform well under constraints (P-03 showed they do not), but whether the infrastructure layer that evaluates, records, and reconstructs governance decisions operates correctly and consistently across policy domains and model providers. If the infrastructure is sound, the system-engineering problem of building models that operate effectively within governance constraints becomes addressable. Without sound infrastructure, it is not even well-defined.

2 What Runtime Governance Requires

Runtime governance, as distinct from alignment, guardrails, or monitoring, requires four properties operating simultaneously during agent execution:

1. **Per-action policy evaluation.** Every action the agent attempts is evaluated against a loaded policy corpus before or during execution. The evaluation produces a typed decision: ALLOW (action permitted), REQUIRE (action permitted contingent on attestation gates), or DENY (action blocked by policy or by fail-closed default).
2. **Tamper-evident decision records.** Each governance decision is recorded as a Policy Restriction Evidence (PRE) record: canonically serialized (JCS RFC 8785), hash-chained, and cryptographically signed. The PRE captures the action context, decision outcome, and evidence linkage in a format that is independently verifiable after the fact and sufficient to recover the full decision trail under frozen policy configuration.
3. **Evidence chain completeness.** Every governed action must produce exactly one governance decision, every decision must emit exactly one PRE, and every PRE must be ingested, verified, and stored. Zero gaps. The evidence chain from action to decision to record to storage must be unbroken.
4. **Reconstructability.** The complete decision trail must be recoverable from stored PREs alone, without access to the original execution trace, the governance evaluation, or any other execution artifact. If the evidence store is the only thing that survives, the full governance history is still recoverable.

These properties are infrastructure requirements, not policy requirements. They must hold regardless of what policy domain drives the decisions, whether the constraint is operational (“verify before deploy”), security (“don’t execute arbitrary code”), or organizational (“enforce data residency”). P-04 tests whether Probit’s governance stack satisfies all four properties, invariantly, across three policy domains and two model providers.

3 Interpreting Performance Under Governance

P-03 data requires careful interpretation. Ungoverned systems produced a 3.4% pass rate; externally governed agents produced 0%. Read in isolation, this suggests governance degrades outcomes. The correct reading is more nuanced.

An ungoverned system that “passes” by mutating production state without approval, skipping verification steps, or accessing sensitive data without authorization has not passed in any production-relevant sense. Its pass rate measures compliance with a scoring rubric, not with the constraints a deployed system must satisfy. In production, the ungoverned “pass” would be a policy violation.

Governed execution is closer to the production baseline. The observed degradation (system failure rates rising from 43% to 69% under external governance) is not a governance failure. It is evidence that current systems cannot accomplish their objectives without taking actions that governance correctly prevents. The governance layer is working; the system is not designed to operate within constraints.

P-04 therefore does not measure whether governance improves task pass rates. It measures whether the governance *infrastructure* works: whether enforcement is consistent, decisions are recorded, evidence chains are complete, and the infrastructure operates identically across policy domains. Performance under governance is a system-engineering question that depends on how the model interacts with governance decisions. Infrastructure correctness is a prerequisite, and it is what P-04 evaluates.

4 Experiment Design

4.1 Overview

P-04 executes 750 runs across a fully crossed design:

Table 1: P-04 experiment dimensions

Dimension	Values
Domains	3 (operational, security, organizational)
Scenarios per domain	5
System types	5 (raw_llm, best_practice, agent, governed_best_practice, governed_agent)
Models	2 (Claude Sonnet 4.5, GPT-4o)
Reps per cell	5
Total runs	750
Governed runs	300

The 5 system types span the spectrum from single-prompt baselines (raw_llm) through structured tool-calling systems (best_practice) to multi-role agent pipelines (agent), with governed variants of the latter two. Ungoverned types provide per-domain behavioral baselines; governed types produce the governance evidence.

4.2 Three Policy Domains

The three domains exercise materially different constraint shapes through the same governance engine:

Table 2: Policy domain characteristics

Domain	Constraint Type	Policies	Tools	Enforcement Profile
Operational	Workflow compliance	7	8	DENY-dominant
Security	Cybersecurity	6	7	REQUIRE-dominant
Organizational	Enterprise compliance	6	6	ALLOW/REQUIRE balanced

Operational constraints enforce verification before mutation, approval before deployment, and evidence before completion. The policy bundle is tier-varied (complexity tiers 1–3) and inherited from P-03, providing direct comparability. The DENY-dominant profile results from multiple tools having no matching policy rule; the governance engine’s fail-closed default applies.

Security constraints enforce data access authorization, code execution scope limits, credential handling restrictions, and network boundary enforcement. Most sensitive actions require attestation gates (REQUIRE decisions), producing a REQUIRE-dominant profile.

Organizational constraints enforce data residency, approval chains, audit logging, PII handling, and retention policies. Every tool maps to exactly one policy, providing complete coverage with no resolver-default DENY. The resulting ALLOW/REQUIRE balanced profile reflects explicit policy decisions rather than fail-closed defaults.

These three profiles are chosen because they exercise qualitatively different enforcement shapes: a fail-closed default-heavy domain, a gate-heavy domain, and an explicit-coverage domain. If the infrastructure is invariant, it should produce correct enforcement across all three.

4.3 Two Model Providers

Table 3: Model providers

Model	Provider	Runs
Claude Sonnet 4.5	Anthropic	375
GPT-4o	OpenAI	375

Using two models from different providers tests whether the governance infrastructure depends on any particular model’s behavior patterns. The infrastructure should process each governed action

identically regardless of which model generated it. Decision *distributions* will differ between models (different models take different actions); decision *processing* should not.

4.4 Governance Stack

Probity’s governance stack consists of four components:

1. **probity-governance**: Policy evaluation engine. Loads signed policy bundles, evaluates each action against matched rules, and produces ALLOW/REQUIRE/DENY decisions with full rationale. No domain-conditional logic in the engine.
2. **probity-cloud**: PRE pipeline. Emits, ingests, verifies, correlates, and stores tamper-evident decision records. Reconstruction recovers the full decision trail from stored PREs alone.
3. **probity-api**: Evidence protocol and verification interface.
4. **probity-labs**: Execution framework, trace instrumentation, scoring, and evidence chain audit.

4.5 Invariance Framework

P-04 defines 13 formal invariants organized into two groups.

Cross-domain invariants (INV-01 through INV-08) assert that the governance infrastructure is structurally identical across the three policy domains: same decision vocabulary, same PRE schema, same serialization constants, same verification path, same correlation semantics, same reconstruction algorithm, same signing infrastructure, and no domain-conditional branches in the engine source code.

Cross-model invariants (XINV-01 through XINV-05) assert structural identity across the two model providers. Each XINV is evaluated independently per model, then per-model results are compared. The framework explicitly distinguishes *structural invariance* (required: same PRE schema, same verification, same reconstruction contract) from *behavioral variance* (expected: different models take different actions and may produce different decision distributions).

5 Results

5.1 Aggregate Results

Table 4: P-04 aggregate results across 300 governed runs, 3 domains, 2 models

Metric	Result
Total runs executed	750
Governed runs	300
PREs emitted and verified	922 / 922 (100%)
Evidence chain gaps	0
Exact reconstruction rate	300 / 300 (100%)
Kill test (PRE-only reconstruction)	300 / 300 (100%)
Cross-domain invariants (INV-01–08)	8 / 8 pass
Cross-model structural invariants (XINV-02–05)	4 / 4 pass
Cross-model behavioral check (XINV-01)	Expected variance

Every governed run produces a complete, gap-free evidence chain. Every decision trail is exactly reconstructable from stored PREs alone, without the original execution trace, governance evaluation, or any sibling artifact. This holds across all three domains and both model providers.

5.2 Enforcement Profiles Across Domains

Table 5: Decision distribution by domain (922 decisions across 300 governed runs)

Effect	Operational		Security		Organizational	
	Count	%	Count	%	Count	%
ALLOW	44	11.9	81	28.0	121	46.0
REQUIRE	36	9.7	198	68.5	125	47.5
DENY	290	78.4	10	3.5	17	6.5
Total	370		289		263	

Three qualitatively different enforcement profiles emerge from the same governance engine. The operational domain is DENY-dominant (78.4%), with many tools lacking matching policies and

triggering the engine’s fail-closed default. The security domain is REQUIRE-dominant (68.5%), with most sensitive actions gated by attestation requirements rather than blocked outright. The organizational domain is ALLOW/REQUIRE balanced (46.0% ALLOW, 47.5% REQUIRE), where complete policy coverage means every action receives an explicit policy decision rather than a resolver default.

These differences are policy properties, not infrastructure properties. The engine that produces 78% DENY for operational constraints produces 3.5% DENY for security constraints, not because the engine behaves differently, but because the loaded policies are different.

5.3 Cross-Domain Invariance

Table 6: Cross-domain structural invariants (INV-01 through INV-08)

ID	Assertion	Status
INV-01	Decision vocabulary: exactly {ALLOW, REQUIRE, DENY} per domain	PASS
INV-02	PRE envelope schema identical across domains	PASS
INV-03	Serialization constants identical (922 PREs checked)	PASS
INV-04	Verification path identical (922 PREs verified, 0 failures)	PASS
INV-05	Correlation semantics identical (key usage, edge vocabulary)	PASS
INV-06	Reconstruction algorithm identical (300 runs, 1 field set)	PASS
INV-07	Bundle signing infrastructure identical (Ed25519, same schema)	PASS
INV-08	No domain-conditional branches in governance engine source	PASS

The eight invariants cover the full governance stack: decision semantics (INV-01), PRE emission and integrity (INV-02–04, INV-07), evidence chain assembly (INV-05), and reconstruction (INV-06). INV-08 verifies by source code analysis that the governance engine contains no conditional logic branching on domain identity. All pass.

5.4 Cross-Model Invariance

Table 7: Cross-model invariants (XINV-01 through XINV-05)

ID	Assertion	Result
XINV-01	Decision vocabulary per model per domain	Behavioral variance*
XINV-02	PRE schema identical per model	PASS
XINV-03	Serialization constants identical per model	PASS
XINV-04	Verification path identical per model (0 failures)	PASS
XINV-05	Reconstruction contract identical per model (100% match)	PASS

*XINV-01 is a coverage check, not a structural invariant. In 11 of 12 (domain $\times$ model) cells, governed runs exhibited all three effects. GPT-4o governed runs in the security domain observed only {ALLOW, REQUIRE}; the DENY-producing tools (`send_request`, `write_file`) were not invoked. This reflects model behavior, not infrastructure divergence.

XINV-02 through XINV-05 confirm that the PRE schema, serialization, verification, and reconstruction contract are structurally identical for both models. The infrastructure processes Sonnet-generated actions and GPT-4o-generated actions through the same pipeline with the same semantics and the same reconstruction fidelity. What differs is which actions the models generate, a behavioral property rather than an infrastructure property.

5.5 Reconstruction and the Kill Test

Reconstruction is performed using an 11-field exact-match contract: `decision_effect`, `policy_id`, `bundle_id`, `action_type`, `operation`, `resource`, `sequence_position`, `record_id`, `required_gates`, `severity`, `reason`. Five of these fields are directly observed from persisted artifacts; six are deterministically derived through stateless re-evaluation against the frozen policy bundle.

Table 8: Reconstruction results by domain

Domain	Governed Runs	Exact Matches	Match Rate
Operational	100	100	100%
Security	100	100	100%
Organizational	100	100	100%
Total	300	300	100%

The kill test strengthens this result. For each of the 300 governed runs, reconstruction is performed from an isolated copy of `pre_records.json`, with `trace.json`, `governance_evaluation.json`, and other execution artifacts deliberately excluded. Reconstruction is allowed access only to the frozen policy configuration required for deterministic re-evaluation. All 300 isolated reconstructions match the trace-derived originals on all 11 fields.

This means the PRE is a self-contained audit record. Given only the stored PREs for a run, the complete governance decision trail (what was decided, by which policy, for which action, in what order) is fully recoverable. No access to the executing system, the original trace, or any other artifact is required.

6 The Invariance Separation

P-04’s results demonstrate a clean separation between two layers:

Table 9: Infrastructure (invariant) versus policy (variable)

Layer	What It Contains	Behavior
Infrastructure (domain-invariant)	Decision engine, PRE schema, verification, serialization, reconstruction, signing	Invariant across domains and models
Policy (domain-specific)	Policy rules, tool mappings, gate specifications, enforcement profiles	Variable by domain; produces different effects

This separation is verified empirically, not assumed. The eight INV checks and five XINV checks test each component of the infrastructure layer independently. The enforcement profile table (Table 5) demonstrates that different policies produce different effects through the same engine.

The practical consequence: adding a new policy domain does not require engineering the governance infrastructure. It requires defining a policy corpus (as a signed bundle) and a tool mapping. The engine, PRE pipeline, verification, and reconstruction carry over unchanged. This is what makes runtime governance a single architectural concern rather than a per-domain engineering effort.

7 What the Evidence Chain Enables

The combination of per-action enforcement, tamper-evident records, and PRE-based reconstruction produces an evidence primitive that does not exist today for AI agent execution:

Incident investigation. When an AI agent takes an action with adverse consequences, the PRE

trail provides a complete, tamper-evident record of every governance decision made during execution: which actions were allowed, which were blocked, which required attestation gates, and which policies produced each decision. This record is available from the evidence store without access to the executing system.

Compliance audit. SOC 2, regulatory frameworks, and enterprise audit requirements demand evidence that constraints were enforced during operation. The PRE trail provides this at the level of individual actions, not aggregate reports. Each decision is independently verifiable against the signed policy bundle that produced it.

Cross-domain consistency. Across operational, security, and organizational policy domains, the resulting evidence trails retain the same structural format. This allows audit, compliance, and post-hoc investigation workflows to operate over a common evidence representation regardless of which domain generated the decisions.

Forensic reconstruction. The kill test (300/300 at 100% fidelity) demonstrates that the governance trail can be reconstructed from stored evidence alone, after the fact, without access to any artifact beyond the PREs themselves. This is the accountability primitive: even if the executing system is no longer available, the governance record is recoverable and verifiable.

8 Open Questions

P-04 establishes that the governance infrastructure works. It does not establish that current AI systems can operate effectively within governance constraints.

P-03 demonstrated that external governance, where the model has no visibility into governance decisions, creates a structural failure mode. The model attempts an action, governance blocks it, and the model has no mechanism to understand why or adapt its plan. This produces safe but non-functional execution.

The structural question this raises is whether governance decisions can be made visible to the model as first-class information: not as an opaque wall, but as a signal the model can reason about. When governance returns REQUIRE with a gate specification, can the model satisfy the gate? When governance returns DENY with a policy explanation, can the model replan? This is the difference between governance as obstruction (P-03's finding) and governance as information.

P-04 provides the infrastructure foundation for this question. The ALLOW/REQUIRE/DENY decision model, the gate specifications, and the policy rationales are all structured data that could, in principle, be returned to the model's reasoning context. Whether models can effectively use this information to achieve constrained execution is a system-engineering question that depends on model architecture, context management, and integration design. P-04 does not answer it, but it provides the infrastructure layer on which the answer can be built.

9 Scope and Limitations

Local execution. All 750 runs execute locally. P-04 does not demonstrate deployed, multi-tenant, or production-scale operation.

Invariance scope. The invariance claim is conditional on emitted governed actions. GPT-4o governed runs frequently terminate early with adapter errors, producing fewer governance decisions per run than Sonnet (mean 0.9 vs 5.3 decisions/run). This reflects adapter behavior, not governance infrastructure behavior. Every action that enters the governance pipeline is processed identically regardless of model.

Policy corpus scope. The three policy corpora total 19 policies. They are sufficient to demonstrate infrastructure invariance across qualitatively different enforcement profiles but do not represent exhaustive policy libraries for any domain. Whether the engine handles 19 or 1,900 policies is an engineering question; the architectural invariance is what P-04 establishes.

Scenario design. All scenarios, tool executors, and policy corpora were designed by the research team. External validation against production workloads is future work. The scenarios are sufficient to exercise qualitatively different enforcement shapes, but they do not represent the complexity or adversarial conditions of production agent deployments.

No outcome claims. P-04 measures infrastructure correctness, not task outcomes. Whether governance improves, degrades, or has no effect on task completion rates is not evaluated. The governed runs in P-04 serve as infrastructure evidence (do PREs emit? do evidence chains complete? does reconstruction work?), not as performance benchmarks.

XINV-01 behavioral gap. GPT-4o governed runs in the security domain did not exercise the DENY decision path. This is a consequence of GPT-4o’s action patterns, not an infrastructure limitation. All structural invariants hold.

10 Conclusion

P-01 through P-03 established that AI systems fail systematically at runtime, that these failures are invisible to standard evaluation, that no tested architecture resolves the problem, and that external governance makes it worse. The governance void is real, and it is structural.

P-04 demonstrates that runtime governance infrastructure can fill this void. Across 750 runs, 3 policy domains, 2 model providers, and 300 governed executions producing 922 tamper-evident decision records: the same governance engine, the same PRE schema, the same verification path, and the same reconstruction algorithm operate identically for operational workflow compliance, cybersecurity enforcement, and enterprise compliance constraints. Decision trails are reconstructable from stored PRE evidence plus frozen policy configuration, without the original execution trace, at 100% fidelity across all domains and both models.

The reconstruction result deserves particular emphasis. The kill test (300 governed runs reconstructed from isolated PRE storage plus frozen policy configuration, with no access to execution traces or governance evaluations, matching the originals exactly on all 11 fields) establishes a strong governance primitive for AI agent execution: a tamper-evident, after-the-fact recoverable decision record. Logging records what happened. Monitoring watches it. The PRE trail proves what was *decided*, and it is recoverable without the system that made the decisions.

The broader finding is a separation: policy content is domain-specific; enforcement infrastructure is domain-invariant. Organizations do not need separate governance systems for each constraint domain. The infrastructure is one layer. The policies are configuration.

What remains open is whether AI systems can be designed to operate effectively within this infrastructure, not just be constrained by it, but to use governance decisions as information to achieve constraint-aware execution. P-04 provides the foundation: the infrastructure works, the evidence chains are complete, and the reconstruction primitive is proven. Building systems that perform well within these constraints is the next challenge.

A Experiment Summary

Table 10: P-04 experiment summary

Metric	Value
Total runs	750
Governed runs	300
Ungoverned runs	450
Domains	3
Models	2
System types	5
Scenarios	15 (5 per domain)
Reps per cell	5
Batches	30
Total PREs	922
PREs verified	922 (100%)
Reconstruction match rate	100% (300/300)
Kill test	PASSED (300/300)
Evidence chain gaps	0
INV passed	8/8
XINV structural passed	4/4

B Invariance Assertion Groups

Table 11: How invariants map to infrastructure components

Component	Invariants	What Is Verified
Governance engine	INV-01, INV-08	Same decision model, no domain branching
PRE pipeline	INV-02, INV-03, INV-04	Same schema, serialization, verification
Reconstruction	INV-06	Same algorithm, same 11-field contract
Evidence chain	INV-05, INV-07	Same correlation semantics, same signing

C Dataset

The P-04 dataset is available at <https://github.com/onkura/probity-runtime-verification-studies>.

The released dataset contains three CSV files, one per domain, with per-run review records for all 750 executions:

- **operational_run_review.csv** (250 runs): Full review rows with task outcome, failure classification, governance metrics, enforcement simulation, and agent-specific measurements. Compatible with the P-03 review row schema.
- **security_run_review.csv** (250 runs): Enforcement-only review rows. Governed runs include governance metrics (actions attempted/blocked/gated/allowed, PRE count, evidence chain status, reconstruction match, decision counts). Ungoverned runs include execution metadata.
- **organizational_run_review.csv** (250 runs): Same enforcement-only schema as security.

All rows include: `run_id`, `scenario_id`, `system_type`, `base_system_type`, `model`, `batch_id`, `complexity_tier`, `governance_mode`, `stop_reason`, and `wall_clock_seconds`. These fields enable reconstruction of all results presented in this paper, including per-domain enforcement profiles, cross-model comparisons, and governed system type breakdowns.