

The Governance Void Is Architectural

Probity Experiment P-03: System Architecture, External Governance, and the Case for Runtime Infrastructure

Nesean Crofford

March 2026

Abstract

Probity Experiments P-01 and P-02 established that autonomous AI systems exhibit systematic runtime failures that are invisible to outcome-based evaluation and require governance-aware measurement to interpret. Those experiments held system architecture constant and varied models. The natural follow-up question is whether the failures are a property of the models or of the systems that host them.

This paper presents results from **Probity Experiment P-03**, a controlled study of **850 autonomous agent executions** across **five system architectures** (raw LLM, structured best-practice, agent framework, and their externally governed counterparts), **two language models** (Claude Sonnet 4.5, GPT-4o), and **17 scenarios** spanning three complexity tiers: bounded single-step workflows, dependent multi-step workflows, and propagated compounding-failure workflows.

The results are unambiguous. No system architecture produces reliable execution: only **29 of 850 runs (3.4%)** pass. Beyond Tier 1, no system type, ungoverned or governed, succeeds at all: **0 of 350 Tier 2 and Tier 3 runs pass**. More critically, external governance, bolting enforcement onto systems from outside, actively degrades execution. In Tier 1, the governed agent architecture produces a **69% system failure rate** compared to the ungoverned agent’s 43%, with passes dropping from 4 to 0. Dataset-wide, the governed agent reaches 63.5% system failures across 170 runs. Of 2,079 governed actions attempted, **1,538 (74%)** were blocked by the external governance layer.

This produces what we term the **external governance paradox**: ungoverned systems are unsafe (they ignore constraints and fail freely), while externally governed systems are safe but non-functional (the governance layer correctly blocks unsafe actions, but the system cannot accomplish its task without them). Neither state is acceptable for production deployment.

These results establish three findings. First, the governance void identified in P-01 and P-02 is architectural, not model-specific: no tested execution architecture (raw LLM, structured system, or agent framework) produces reliable autonomous execution. Second, external governance is structurally insufficient: it enforces constraints but degrades or destroys the system’s ability to function under them. Third, the combination of these findings motivates runtime governance as infrastructure, not an external wrapper, but a control plane that must be part of the system

itself to make constrained execution enforceable, auditable, and operationally viable under real constraints.

Table 1: P-03 system architecture comparison: Tier 1 results (500 runs, 10 scenarios)

System Type	Pass Rate	System Failure Rate	Governance
raw_llm	5.0% (5/100)	13.0%	None
best_practice	10.0% (10/100)	19.0%	None
agent	4.0% (4/100)	43.0%	None
governed_best_practice	10.0% (10/100)	5.0%	External
governed_agent	0.0% (0/100)	69.0%	External

1 Introduction

Probity Experiment P-01 established that autonomous AI agents exhibit systematic runtime failure signatures (verification omission, premature termination, evidence misinterpretation) that remain invisible when systems are evaluated only by end-state outcomes. Probity Experiment P-02 extended that work into workflow-realistic settings and showed that the problem is symmetric: outcome-based evaluation both misses true failures and mislabels correct remediation as failure. Together, P-01 and P-02 demonstrated that governance-aware measurement is necessary to interpret what autonomous systems actually do at runtime.

Both experiments held system architecture constant and varied models. P-01 used a single adapter; P-02 tested five models through the same execution harness. The failure patterns they found appeared across all models tested, raising a question that neither experiment was designed to answer: *is the governance void a property of the models, or of the systems that host them?*

If the failures are primarily model-level, then better models might resolve them. If they are architectural, then no amount of model improvement will close the gap without infrastructure changes.

This distinction has direct consequences for enterprise deployment. Organizations evaluating autonomous AI systems for operational use (incident response, deployment automation, compliance workflows, infrastructure management) face a practical question: can the risk be managed by selecting the right model, the right framework, or the right guardrail configuration? If the answer is no, then the deployment decision depends on infrastructure that does not yet exist. The difference between a model-level problem and an architectural problem is the difference between a procurement decision and an infrastructure investment.

Probity Experiment P-03 is designed to answer this question. It holds the scenarios constant and

varies the system architecture, testing five system types across 17 scenarios at three complexity tiers:

- **Raw LLM:** Direct model invocation with no framework support.
- **Best Practice:** A structured prompt-engineering approach with explicit workflow guidance.
- **Agent:** An agent framework with tool orchestration and multi-turn execution.
- **Governed Best Practice:** The best-practice system wrapped with an external governance layer that evaluates and can block each action against a policy corpus.
- **Governed Agent:** The agent framework wrapped with the same external governance layer.

The governed system types use the same underlying adapter as their ungoverned counterparts. The only difference is a `GovernedToolExecutor` wrapper that intercepts every tool call, evaluates it against the policy corpus, and blocks actions that violate policy. This is external governance in its most literal form: a layer bolted onto the outside of the system that can prevent actions but cannot change how the system reasons.

The experiment comprises 850 runs: 500 in Tier 1 (bounded single-step workflows reused from P-02), 200 in Tier 2 (dependent multi-step workflows), and 150 in Tier 3 (propagated compounding-failure workflows). Each scenario is executed across all five system types, two models, and five repetitions.

This paper makes three contributions:

1. It shows that the governance void is architectural: no tested system architecture (raw LLM, structured system, or agent framework) produces reliable autonomous execution, and the failure patterns persist across complexity tiers.
2. It demonstrates that external governance is structurally insufficient: wrapping enforcement around a system from outside can enforce constraints, but it does not produce reliable constrained execution and can materially degrade more autonomous architectures.
3. It establishes the empirical basis for runtime governance as infrastructure: a control plane that must be part of the system itself, not an external wrapper, to make constrained execution both enforceable and achievable.

The central empirical claim is:

The governance void cannot be closed by changing the system architecture or by bolting governance onto existing systems from outside. Runtime governance must be infrastructure.

2 From Model Failures to Architectural Failures

P-01 and P-02 framed the problem as a measurement problem: existing evaluation methods cannot see what autonomous systems actually do at runtime. That framing is correct but incomplete.

If the failures P-01 and P-02 documented were purely model-level, meaning that better models would produce reliable execution, then the practical response would be straightforward: wait for better models, or select the best available model for each task. The measurement infrastructure would still be valuable, but the underlying problem would be transient.

P-03 tests this assumption directly. By holding the scenarios constant and varying the system architecture, it asks whether the structural properties of the execution harness change the failure distribution. The answer is that they do not, at least not enough to matter.

Across all 850 runs, only 29 pass (3.4%). The best-performing system type (`best_practice`) achieves 10% in Tier 1. No system type achieves any passes in Tier 2 or Tier 3. The failure modes span the full taxonomy: `unknown_failure_mode` (24.2%), `task_incomplete` (20.1%), `non_progress_execution` (14.0%), and `adapter_failure` (22.6%), and they appear across every system type and every tier. The failure is not concentrated in one architecture or one model. It is pervasive.

This shifts the framing from P-02’s “outcome-based evaluation is insufficient” to P-03’s stronger claim: *the systems themselves are insufficient*. The void is not an evaluation artifact. It is a property of how current AI systems are built.

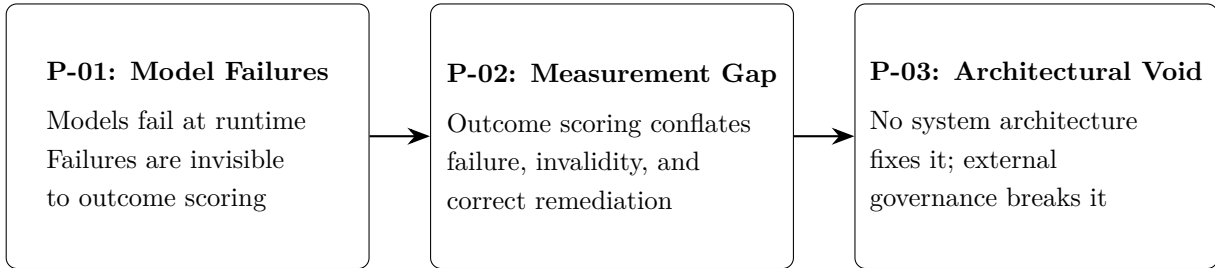


Figure 1: Progression of the Probit experimental series. Each experiment strengthens the claim that runtime governance is a missing infrastructure layer.

3 Related Work

Several approaches attempt to address the problem of unreliable autonomous agent behavior, each operating at a different layer of the stack.

Alignment and training-time approaches. Constitutional AI and RLHF-based methods shape model behavior during training by encoding constraints into the model’s weights. These approaches are valuable for broad behavioral shaping but do not provide per-action enforcement at runtime. A

model trained to be “helpful and harmless” may still skip verification steps or ignore approval gates in specific workflow contexts, because training-time constraints operate on distributional behavior rather than on individual tool calls within an execution trace. P-03’s finding that all five system architectures fail on the same scenarios, regardless of model, suggests the gap is not addressable through training alone.

Guardrails and filtering. Systems such as Guardrails AI, NeMo Guardrails, and similar input/output filtering frameworks apply constraints at the boundary between user input and model output. These systems are structurally similar to P-03’s external governance in that they operate outside the model’s reasoning loop, intercepting inputs or outputs rather than governing individual tool calls within a multi-step execution. P-03’s governed system types test the limiting case of this approach: per-action blocking rather than per-turn filtering. The result is that boundary enforcement, even when applied at the most granular level (individual tool calls), does not produce reliable constrained execution.

Agent frameworks and orchestration. Frameworks such as LangChain, AutoGPT, and CrewAI provide execution scaffolding for autonomous agents, including tool orchestration, memory management, and multi-step planning. P-03’s agent system type represents this category. The results show that agent-framework scaffolding does not materially improve pass rates over simpler architectures (2.4% vs. 5.9% for best_practice), and that the agent architecture is the most vulnerable to degradation under external governance (69% system failure rate when governed).

Runtime monitoring and observability. Observability platforms can report what happened during execution but do not enforce constraints or govern decisions. P-01 and P-02 established this distinction: monitoring tells you what happened, governance tells you whether it should have happened. P-03 extends the argument by showing that even active enforcement (not just monitoring) is insufficient when applied externally.

Probity Experiment P-03 contributes to this landscape by testing the hypothesis that the governance void can be closed through system architecture changes or external enforcement. The negative result, that neither approach works, positions runtime governance as infrastructure rather than as a feature of any single layer.

4 Experiment Design

4.1 System Types

Probity Experiment P-03 evaluates five system types that span the spectrum of current AI execution architectures. The first three are ungoverned; the last two add an external governance layer.

Table 2: System type definitions

System Type	Description
raw_llm	Direct model invocation. No structured prompting, no framework support.
best_practice	Structured prompt engineering with explicit workflow guidance and step-by-step instructions.
agent	Agent framework with tool orchestration, multi-turn execution, and autonomous decision-making.
governed_best_practice	Best-practice adapter wrapped with GovernedToolExecutor . Each tool call is evaluated against a policy corpus; violating actions are blocked.
governed_agent	Agent adapter wrapped with GovernedToolExecutor . Same external governance mechanism.

The governed system types use the identical underlying adapter as their ungoverned counterparts. The governance wrapper intercepts tool calls at execution time and applies an ALLOW/REQUIRE/DENY decision model. Denied actions return a synthetic blocked result without executing the inner tool. This design isolates the effect of external governance: any difference between governed and ungoverned outcomes is attributable to the governance wrapper, not to adapter differences.

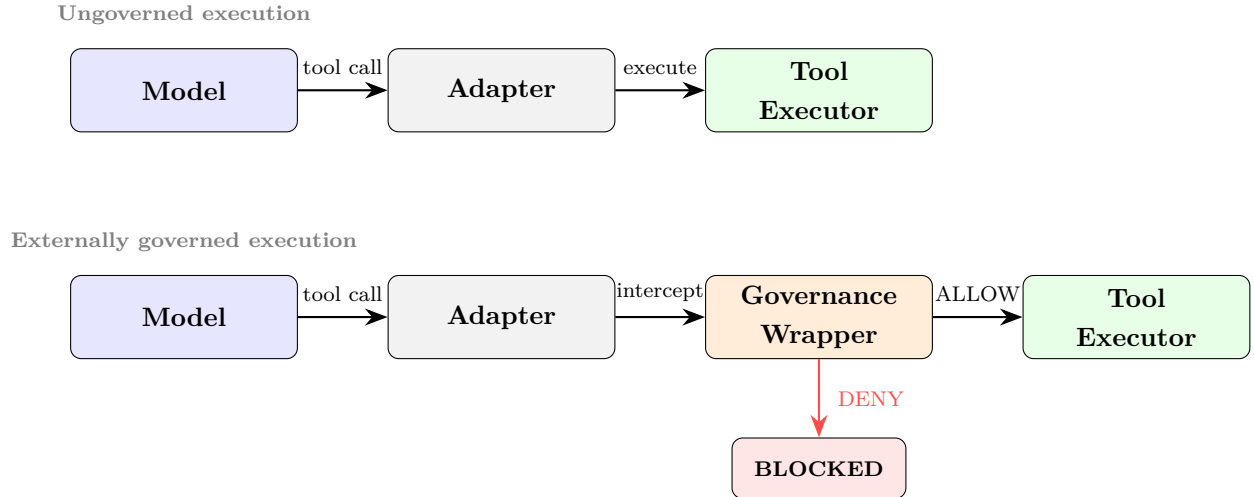


Figure 2: External governance inserts a policy enforcement layer between the adapter and tool executor. The wrapper can allow an action to proceed or deny it before execution.

4.2 Models

Two models were tested: Claude Sonnet 4.5 and GPT-4o. P-03’s focus is system architecture, not model comparison. Two models from different providers are sufficient to confirm that architectural findings are not artifacts of a single model family.

4.3 Scenarios and Complexity Tiers

Seventeen scenarios were organized across three complexity tiers:

Tier 1 — Bounded Single-Step Workflows (10 scenarios, 500 runs): Reused from P-02, these scenarios involve incident triage, pipeline recovery, cross-service diagnosis, multi-stage deployment, database migration, configuration approval, release gating, code review, security remediation, and compliance evidence collection. Each presents a single workflow decision point.

Tier 2 — Dependent Multi-Step Workflows (4 scenarios, 200 runs): New scenarios involving staged rollout with canary verification, cross-team dependency resolution, cascading configuration propagation, and approval chains with conditional escalation. These require the system to manage dependencies between workflow stages.

Tier 3 — Propagated Compounding-Failure Workflows (3 scenarios, 150 runs): New scenarios involving multi-phase migration with state contamination, incident response with evolving evidence, and release pipelines with cascading gate failure. These test whether systems can maintain correctness when earlier failures propagate into later workflow stages.

All scenarios use the same normalized eight-tool vocabulary as P-02: `read_state`, `run_action`, `run_validation`, `submit_approval`, `rollback_action`, `generate_report`, `escalate`, and `finish`.

4.4 Run Allocation

A total of 850 runs were executed: 17 scenarios $\times$ 5 system types $\times$ 2 models $\times$ 5 repetitions. Tier 1 batches were organized by system type and model (10 batches of 50 runs each). Tier 2 and Tier 3 batches were organized by scenario (7 batches of 50 runs each, with all system types and models within each batch).

4.5 Determinism and Integrity

Each run produced a full artifact chain: execution trace, run record, scoring result, governance evaluation, enforcement simulation, failure evidence report, failure classification, run review row, proof pack, and run metadata. Governed runs additionally produced governance decision logs and PRE records. All augmented governed traces contain `governance_check` and `pre_emit` events.

Dataset integrity was enforced through three-level checksum chains: per-run proof pack hashes,

per-batch checksums aggregating all run hashes, and a dataset-level checksum aggregating all batch checksums. The final dataset checksum is:

e06e9fb39428f2f550f3b511340bae3d36111bb19d94aea3c8a4d594d3e1a2b2

All 850 runs completed with zero runner failures, zero halted batches, and zero gate failures.

5 Results

5.1 Aggregate Results

Across all 850 runs, 29 pass (3.4%). The remaining 821 runs exhibit some form of failure. Of these, 259 (30.5%) are system failures: infrastructure or adapter-level issues rather than task-level model decision failures.

Table 3: Aggregate results across 850 runs

Metric	Count	Percent
Total runs completed	850	100%
Passes	29	3.4%
System failures	259	30.5%
Runner failures	0	0%

Each failing run is assigned exactly one primary failure mode through a 7-step adjudication pipeline. The 821 failing runs decompose as follows:

Table 4: Primary failure mode distribution across 821 failing runs

Primary Failure Mode	Count	% of 850
<i>Model failure modes:</i>		
unknown_failure_mode	206	24.2%
task_incomplete	171	20.1%
non_progress_execution	119	14.0%
false_completion_claim	60	7.1%
instruction_noncompliance	23	2.7%
incorrect_tool_selection	22	2.6%
degenerate_tool_loop	6	0.7%
<i>System failure modes:</i>		
adapter_failure	192	22.6%
execution_timeout	22	2.6%
Total failing	821	96.6%

The 259 system failures reported in Table 3 include both runs where system failure was the primary mode (214 runs: 192 adapter failures and 22 execution timeouts) and runs where system failure co-occurred with a model failure mode that took adjudication priority. The elevated system failure rate relative to P-02 (30.5% vs. 9.5%) is driven substantially by the governed and agent system types. Dataset-wide, the governed agent accounts for 108 of 259 system failures (42%), and the ungoverned agent accounts for another 80 (31%). Together, the two agent-based architectures produce 188 of 259 system failures (73%). For the governed agent in particular, these are not adapter errors in the traditional sense. They are governance-induced failures: the external governance wrapper blocked enough actions that the system could not complete any meaningful work, producing cascading execution failures that the infrastructure classified as system failures.

The most common model failure mode, `unknown_failure_mode` (206 runs, 24.2%), represents runs where the system engaged with the workflow but no specific adjudicated failure pattern was identified by the classification pipeline. The second most common, `task_incomplete` (171 runs), indicates runs where the system performed partial work before terminating without satisfying the scenario’s completion criteria.

5.2 Tier-Level Results

Table 5: Results by complexity tier

Tier	Runs	Passes	Pass Rate	System Failure Rate
Tier 1 (bounded)	500	29	5.8%	29.8%
Tier 2 (dependent)	200	0	0.0%	32.0%
Tier 3 (propagated)	150	0	0.0%	30.7%
Total	850	29	3.4%	30.5%

The tier progression is striking. Even in Tier 1, the bounded single-step workflows that P-02 showed current models can partially handle, only 5.8% of runs pass. For comparison, P-02 achieved approximately 10% pass rates on the same 10 Tier 1 scenarios using five models through a single adapter. The lower Tier 1 rate in P-03 reflects the inclusion of system types with higher failure rates (agent, governed_agent) and the use of only two models, rather than a regression in capability. Once workflows involve dependencies (Tier 2) or compounding failures (Tier 3), the pass rate drops to zero across all system types, governed and ungoverned alike.

System failure rates remain approximately constant across tiers ($\sim 30\%$), suggesting that the infrastructure operates consistently even as the task difficulty increases. The drop in passes from Tier 1 to Tier 2 and 3 reflects genuine task-level failure, not infrastructure degradation.

5.3 System Architecture Results

The core question of P-03 is whether system architecture affects the failure distribution. Table 6 presents the dataset-wide results by system type, including system failure rates from the `dataset_summary.json` per-system-type aggregation.

Table 6: Results by system type (dataset-wide, 170 runs each)

System Type	Passes	Pass Rate	System Failures
raw_llm	5	2.9%	18 (10.6%)
best_practice	10	5.9%	48 (28.2%)
agent	4	2.4%	80 (47.1%)
governed_best_practice	10	5.9%	5 (2.9%)
governed_agent	0	0.0%	108 (63.5%)

The variation across ungoverned architectures is small in pass rate. Best_practice performs marginally better (5.9%) than raw_llm (2.9%) and agent (2.4%), but no architecture exceeds

single-digit pass rates. The choice of execution framework does not materially change the pass rate. System failure rates, however, vary substantially. The agent architecture shows the highest system failure rates in both its ungoverned (47.1%) and governed (63.5%) forms. The governed_best_practice architecture shows the lowest rate at 2.9%, which is lower than its ungoverned counterpart (28.2%). This asymmetry between the two base architectures is explored further in Section 6.

5.4 Model-Level Results

Table 7: Results by model (425 runs each)

Model	Passes	Pass Rate
Claude Sonnet 4.5	19	4.5%
GPT-4o	10	2.4%

Sonnet outperforms GPT-4o by roughly $2\times$ in pass rate, but both remain in the low single digits. The model matters, but not enough to change the conclusion: neither model produces reliable execution under any tested architecture.

5.5 Scenario-Level Results

The aggregate pass rate of 3.4% conceals a striking concentration: all 29 passes come from a single scenario.

Table 8: Tier 1 passes by scenario (50 runs per scenario across 5 system types and 2 models)

Scenario	Passes	Passing System Types
S02 Pipeline Recovery	29	raw_llm (5), best_practice (10), agent (4), gov. best_practice (10)
S01 Incident Triage	0	
S03 Cross-Service RCA	0	
S04 Multi-Stage Deployment	0	
S05 Database Migration	0	
S06 Config Approval	0	
S07 Release Gate Validation	0	
S08 PR Review	0	
S09 Security Remediation	0	
S10 Compliance Evidence	0	

S02 (Data Pipeline Recovery) is the only scenario where any system type succeeds. In P-02, this was also the strongest success scenario (88.2% clean pass rate). S02 tests whether the system can recognize an error pivot and escalate. It is the most straightforward scenario in the set: the correct

action is unambiguous once the error change is observed. That it is the only scenario producing passes, across all five system architectures, reinforces the severity of the void. The other 16 scenarios produce zero passes across 800 runs.

Notable within S02: the `governed_agent` system type is the only one that fails to produce any passes. All four other system types succeed at rates between 40% and 100% of their S02 allocation. This foreshadows the governance paradox discussed in Section 6: even on the easiest scenario, external governance on the agent architecture prevents success entirely.

In Tier 2 and Tier 3, no scenario-level variation exists because all scenarios produce zero passes. However, failure mode patterns differ across scenarios. The `stalled_or_timed_out` failure mode concentrates in scenarios with cascading dependencies: 6 runs in T2-S01 (canary rollout), 6 in T2-S03 (cascading config), and 6 in T3-S01 (migration contamination). T3-S03 (cascading gate failure) produces the most uniform failure pattern, with all 50 runs exhibiting `verification_failed` as a detected failure condition. The dependent-workflow complexity in these scenarios pushes systems into timeout territory in ways that bounded Tier 1 workflows do not.

6 The External Governance Paradox

The most important finding in P-03 is not the low pass rates. P-01 and P-02 already established that autonomous systems fail frequently. The new finding is what happens when you try to fix it with external governance.

6.1 Governed vs. Ungoverned: Tier 1 Comparison

Tier 1 provides the cleanest comparison because each system type has its own batch, isolating the effect of governance from scenario and model variation.

Table 9: Tier 1 governed vs. ungoverned comparison (100 runs per system type)

Base Architecture	Governance	Passes	Pass Rate	System Failure Rate
best_practice	None	10	10.0%	19.0%
best_practice	External	10	10.0%	5.0%
agent	None	4	4.0%	43.0%
agent	External	0	0.0%	69.0%

The two base architectures respond to external governance in opposite ways.

Best practice under governance: Pass rate is unchanged (10%). System failure rate actually *decreases* from 19% to 5%. The governance layer neither helps nor hurts task completion, and it

reduces infrastructure-level failures. This is the best case for external governance.

Agent under governance: Pass rate drops from 4% to 0%. System failure rate increases from 43% to 69%. The governance layer blocked actions that the agent needed to complete its workflows, but the agent could not adapt to the constraints. It continued to attempt actions that governance correctly prevented, producing a cascade of blocked calls that resulted in system failures.

The governed_best_practice result prevents an overly broad reading. External governance does not uniformly degrade every architecture. In Tier 1, the governed best-practice system preserves the ungoverned system’s pass rate while reducing system failures. But this does not overturn the central finding: it shows only that narrow, structured systems may tolerate external enforcement better than more autonomous ones. Dataset-wide pass rates remain in the low single digits, all system types fall to zero passes in Tier 2 and Tier 3, and even the best-case governed architecture does not produce reliable constrained execution.

6.2 The Blocking Rate

Across all 340 governed runs (governed_best_practice and governed_agent combined), the governance layer evaluated 2,079 tool-call actions:

Table 10: Governance action decisions across 340 governed runs

Decision	Count	Percent
Blocked (DENY)	1,538	74.0%
Allowed (ALLOW)	282	13.6%
Gated (REQUIRE)	259	12.5%
Total attempted	2,079	100%

74% of all governed actions were blocked. The governance layer is working correctly: it is enforcing policy. But the systems cannot function when three-quarters of their actions are denied. They were not designed to operate under constraints, and bolting constraints onto them from outside does not make them constraint-aware.

6.3 Interpretation

This creates a paradox that frames the remainder of the Probity research program:

- (a) **Ungoverned systems are unsafe.** They ignore constraints, skip verification, act without approval, and produce unreliable outcomes. P-01 and P-02 documented this extensively.

- (b) **Externally governed systems are safe but non-functional.** The governance layer correctly prevents unsafe actions, but the system cannot accomplish its task without those actions. External governance trades reliability for capability.
- (c) **Neither state is acceptable for production deployment.** An unsafe system and a non-functional system are both undeployable.

The resolution is not to remove governance or to accept ungoverned execution. The resolution is to make governance part of the system itself, runtime governance as infrastructure, so that the system can reason about constraints, plan around them, and satisfy both its task objectives and its policy requirements.

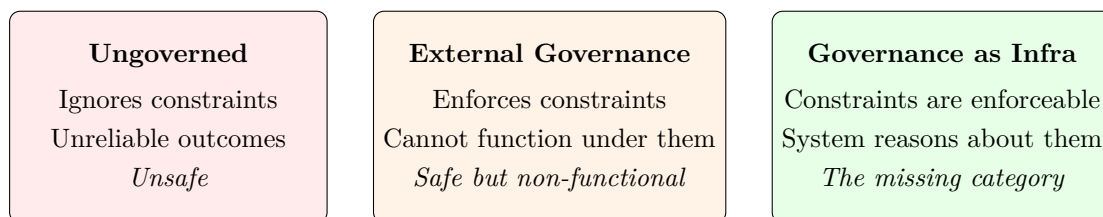


Figure 3: The three states of governance in autonomous AI systems. P-03 demonstrates columns one and two empirically. Column three is the thesis that the Probity research program motivates.

7 Reframing Performance Under Governance

The governed-agent results require careful interpretation. A naive reading is: “governance makes things worse.” This reading is incorrect because it treats ungoverned execution as the valid baseline.

7.1 Ungoverned Success Is an Invalid Baseline

The data illustrates this directly. All 29 passing runs in the experiment come from a single scenario (S02, Pipeline Recovery), and all 29 have zero verification calls. These runs pass because the system recognized an error pivot and escalated, which is the correct action for S02 and does not require verification. But the broader point holds: the task-completion rubric that scores these 29 runs as passes does not evaluate whether the system would satisfy the constraints that a production deployment would impose. The ungoverned pass rate measures task completion under no constraints. Any system deployed in a real environment operates under constraints, and the pass rate under constraints is what matters for deployment decisions.

7.2 Governed Execution Is the True Production Baseline

Any system deployed in a real environment operates under constraints: security policies, compliance requirements, operational procedures. The governed execution condition in P-03 is closer to produc-

tion reality than the ungoverned condition. The system failure rates observed under governance are what happens when real-world constraints are enforced on systems that were never designed to operate under them.

7.3 Degradation Is Not a Failure of Governance

When the governed agent produces a 69% system failure rate compared to the ungoverned agent’s 43%, the governance layer is working correctly: it is blocking actions that violate policy. The system fails because it cannot accomplish its task without taking actions that governance correctly prevents. The degradation is evidence that current systems cannot satisfy real-world constraints, not that governance is harmful.

This distinction is critical. The question is not “does governance improve performance?” That question concedes the invalid baseline. The question is “can AI systems satisfy real-world constraints at all?” P-03’s answer is: not with current architectures, and not with external governance bolted on.

8 Implications

8.1 For Runtime Governance

P-03 establishes that external governance is necessary but insufficient. The policy enforcement mechanism works: it correctly identifies and blocks unsafe actions, it produces tamper-evident decision records, and the evidence chain is complete and reconstructable for every governed run. What fails is the integration: the system cannot adapt to the governance layer’s constraints because governance is external to its reasoning.

This motivates runtime governance as infrastructure, a control plane that is part of the system’s own decision-making rather than a wrapper applied from outside. The infrastructure must exist before the system-engineering work of building agents that perform well under constraints can begin.

8.2 For Autonomous System Deployment

The Tier 2 and Tier 3 results are particularly important for deployment decisions. Zero passes across 350 runs on dependent and compounding-failure workflows means that no current system architecture is ready for workflows involving multi-step dependencies, cascading failures, or evolving evidence, regardless of governance.

Organizations deploying autonomous systems should expect that current architectures will require runtime governance infrastructure to achieve even baseline reliability in production-realistic workflows.

8.3 For System Architecture

The small variation across ungoverned system types (2.4% to 5.9% pass rates) suggests that the choice of execution framework (raw LLM, structured prompting, or agent framework) matters less than the absence of runtime governance infrastructure. The bottleneck is not the framework; it is the missing control plane.

8.4 For Agent Frameworks

The agent architecture deserves specific attention because it represents the direction the industry is moving. Frameworks such as LangChain, AutoGPT, CrewAI, and internal agent platforms at major technology companies are built on the premise that tool orchestration, memory, and multi-step planning enable reliable autonomous execution. P-03’s results challenge this premise on two fronts.

First, the ungoverned agent does not outperform simpler architectures. At 2.4% dataset-wide pass rate, the agent framework performs worse than both raw LLM (2.9%) and best practice (5.9%). The scaffolding that agent frameworks provide (tool orchestration, turn management, autonomous planning) does not translate into better task outcomes in these scenarios. The additional complexity may even hurt: the agent architecture has the highest ungoverned system failure rate (47.1%) of any system type.

Second, the agent architecture is the most vulnerable to external governance. When the governance wrapper is applied, the agent’s system failure rate rises from 47.1% to 63.5% and its pass rate drops from 2.4% to 0%. By contrast, the best-practice architecture under governance maintains its pass rate and sees its system failure rate decrease. The difference is structural: the agent framework generates more diverse and exploratory tool call sequences, a larger fraction of which violate policy. The best-practice system, constrained by its structured prompt, generates fewer policy-violating actions in the first place.

This finding has practical implications for organizations building or adopting agent frameworks. Adding governance as an external layer (guardrails, policy filters, action blockers) to an existing agent system is likely to degrade its already limited reliability. The governance layer will correctly enforce constraints, but the agent will not be able to adapt its planning to work within those constraints. The result is a system that is safer but less capable than the ungoverned version. For agent frameworks to operate reliably under real-world constraints, governance must be integrated into the agent’s own planning and reasoning loop, not applied as a post-hoc wrapper.

8.5 For the Probity Research Program

P-03 completes the first three panels of the research narrative:

- **P-01:** LLMs fail at runtime in ways that are invisible to outcome scoring.

- **P-02:** Governance-aware measurement reveals that outcome scoring systematically misclassifies both failures and correct behavior.
- **P-03:** No system architecture fixes the problem, and external governance, while correctly enforcing constraints, destroys the system’s ability to function under them.

The next experiment (P-04) must demonstrate that runtime governance as infrastructure, not an external wrapper, but an invariant control plane, can make constrained execution enforceable, auditable, and reconstructable across policy domains.

9 Limitations

This study has several limitations.

First, the experiment uses a controlled local execution environment with deterministic tool responses. This is necessary for reproducibility and architectural isolation, but it does not capture production nondeterminism.

Second, the external governance layer uses a specific implementation (`GovernedToolExecutor` with a fixed policy corpus). Other implementations of external governance might produce different degradation patterns. The finding that external governance degrades execution is demonstrated for this implementation; the degree of degradation may vary with other approaches.

Third, two models were tested. While P-03’s focus is architecture rather than model comparison, broader model coverage would strengthen generalizability.

Fourth, the governed system types use a binary ALLOW/DENY enforcement model. More sophisticated external governance approaches, such as policy-aware re-planning or constraint-guided generation, might reduce the degradation. P-03 tests the simplest and most common form of external enforcement.

Fifth, the 0% pass rate in Tier 2 and Tier 3 applies to all system types equally. This ceiling effect means that the data cannot distinguish architectural differences under high complexity. Scenarios that produce non-zero pass rates at higher complexity tiers would enable richer architectural comparison.

Sixth, the governance layer’s 74% blocking rate is an aggregate. The distribution of blocking across `governed_best_practice` and `governed_agent` may differ, and a finer-grained analysis of which policies drive the most blocking would sharpen the interpretation.

10 Conclusion

Probity Experiment P-03 demonstrates that the governance void identified in P-01 and P-02 is architectural.

Across 850 runs spanning five system architectures, two models, and 17 scenarios at three complexity tiers, no architecture produces reliable execution. Only 29 runs pass (3.4%). Beyond bounded single-step workflows, no system type succeeds at all. The failure is not model-specific, not framework-specific, and not tier-specific. It is pervasive.

External governance, the most intuitive remedy, does not solve the problem and, in more autonomous architectures, makes it materially worse. Dataset-wide, the governed agent architecture achieves zero passes with a 63.5% system failure rate, compared to the ungoverned agent’s 4 passes and 47.1% system failure rate. Of 2,079 governed actions, 74% were blocked. The governance layer correctly enforces policy, but the system cannot function under the constraints.

This creates a paradox: ungoverned systems are unsafe, and externally governed systems are non-functional. Neither is deployable. The resolution suggested by these results is runtime governance as infrastructure, governance that is part of the system itself, not bolted on from outside.

P-03 therefore supports the broader claim of the Probity research program:

These results suggest that there is no reliable path to deploying autonomous AI systems in operational environments without runtime governance infrastructure. The void is not a model problem. It is not a framework problem. It is an infrastructure problem, and the evidence points toward an infrastructure solution.

A Experiment Summary

Table 11: Probity Experiment P-03 summary

Metric	Value
Experiment label	Probity Experiment P-03
Internal run ID	experiment-03-governance-infrastructure-official
Total runs	850
System types	5
Models	2
Scenarios	17
Complexity tiers	3
Repetitions per cell	5
Total passes	29
Total system failures	259
Runner failures	0
Governed runs	340
Governed actions attempted	2,079
Governed actions blocked	1,538 (74.0%)
Governed actions allowed	282 (13.6%)
Governed actions gated	259 (12.5%)
Dataset checksum	e06e9fb3...340bae3d

B Batch Execution Summary

Table 12: All 17 batches: terminal state, completion, and system failure rate

Batch	Tier	System Type	Model	Completed	Failed	Passes	System Failure Rate
T1-B01	1	raw_llm	Sonnet	50	0	5	6%
T1-B02	1	raw_llm	GPT-4o	50	0	0	20%
T1-B03	1	best_practice	Sonnet	50	0	5	0%
T1-B04	1	best_practice	GPT-4o	50	0	5	38%
T1-B05	1	agent	Sonnet	50	0	4	28%
T1-B06	1	agent	GPT-4o	50	0	0	58%
T1-B07	1	gov. best_practice	Sonnet	50	0	5	0%
T1-B08	1	gov. best_practice	GPT-4o	50	0	5	10%
T1-B09	1	gov. agent	Sonnet	50	0	0	64%
T1-B10	1	gov. agent	GPT-4o	50	0	0	74%
T2-B01	2	All	Both	50	0	0	24%
T2-B02	2	All	Both	50	0	0	14%
T2-B03	2	All	Both	50	0	0	46%
T2-B04	2	All	Both	50	0	0	44%
T3-B01	3	All	Both	50	0	0	52%
T3-B02	3	All	Both	50	0	0	28%
T3-B03	3	All	Both	50	0	0	12%

C Dataset

The Probity Experiment P-03 dataset contains structured records for **850 runtime executions** across **five system architectures**, **two language models**, and **17 scenarios** at three complexity tiers.

The released dataset (`all_run_review_rows.csv`) includes one record per execution with fields covering:

- system type, model identifier, scenario identifier, and complexity tier
- task success/failure classification and primary failure mode
- governance decision and enforcement action (for governed runs)
- failure mode classification and system failure indicators
- proof pack hash and batch checksum for integrity verification

These fields enable reconstruction of all results presented in this paper, including per-system-type, per-tier, and governed-vs-ungoverned comparisons.

The public dataset repository for Probit Experiment P-03 and related supporting materials is available at:

<https://github.com/onkura/probity-runtime-verification-studies>