Governance at Scale Through Compiled Execution

Opus Experiment O-03: Structural Guarantees Across 120 Workflows and 12 Domains

Nesean Crofford

Abstract

As enterprises deploy AI-driven decision systems into regulated operational settings, the binding constraint is not accuracy but governance. When a policy changes, three questions must be answerable: what changed, what decisions are affected, and can the result be proven? Runtime inference systems do not answer these questions structurally. Their execution artifacts are generated token sequences, not formal representations amenable to complete diff, bit-identical replay, or content-addressed verification.

This paper presents an empirical evaluation of compiled execution across 120 enterprise workflows spanning 12 domains, with 2,294 test inputs and 240 compiled artifacts. The structural diff surface identifies 219 of 219 known policy changes (diff completeness = 1.0). Every historical decision is reproducible with bit-identical output and a cryptographic proof chain (replay success = 1.0). Determinism holds at 1.0 across all 120 workflows: same input, same compiled artifact, same output, always. Across the full corpus, 124 decision flips are enumerated across all 12 domains in under 20 seconds.

The contribution is threefold: (1) evidence that the execution guarantees established at small scale in O-02 are structural rather than statistical, holding identically at 30x the workflow count; (2) a demonstration that governance operates at corpus scale, with complete policy-change coverage and bit-identical replay; and (3) a clear articulation of compiled execution's position in a full pipeline from intent to governance, distinguishing it from both runtime agent systems and traditional rules engines.

1 Introduction

When an enterprise changes a policy, three questions matter: what changed, why, and can you prove it?

These questions are not theoretical. A financial institution that tightens a credit threshold needs to know which decisions are affected before the change goes live. A healthcare organization that modifies a prior authorization rule must demonstrate, under audit, that historical decisions were made under the correct policy version. A procurement team that adjusts an approval limit needs to verify that downstream processes are not disrupted.

In systems built on runtime inference, these questions have no structural answer. The execution artifact is a token sequence generated anew on each invocation. There is no formal representation to diff between policy versions. There is no deterministic function to replay. There is no structurally reliable mechanism linking a rule modification to a specific decision change. Governance is approximated through generated explanations: plausible narratives that are not themselves execution artifacts and therefore are not structurally verifiable.

The enterprise AI stack has a missing layer. Language models are increasingly capable of understanding policy intent. Operational systems (ERP, claims platforms, compliance engines) are

capable of acting on structured instructions. But between understanding and action, there is no trustable execution substrate: no layer that transforms what the model understood into something that executes correctly, deterministically, and with a governance trail. This is the infrastructure gap that compiled execution addresses.

Compiled execution fills this gap by separating intelligence from execution. Intelligence is applied at authoring time, when human intent is transformed into a formal intermediate representation. At runtime, execution is a pure function over a sealed artifact: deterministic, inspectable, and reproducible. The distinction is not between "AI" and "rules" but between systems that infer their way through execution at runtime and systems that enforce correctness by construction.

This distinction, *enforcement rather than evaluation*, is central to the present work. Compiled execution does not test whether outputs are correct after the fact. It constructs a representation in which the admissible execution surface is fixed before runtime. The decision function is sealed at compile time, and the validated policy surface guarantees a unique output path for every admissible input. Correctness is therefore enforced by construction within the compiled representation, rather than measured after the fact.

1.1 The Research Arc: O-01 → O-02 → O-03

This paper is the third in a sequence. Each experiment addresses a distinct question in the argument for a missing execution layer between AI reasoning and operational consequence.

O-01: Is the problem real? Experiment O-01 established that language models are unreliable execution systems. Across 1,260 runs on 12 deterministic policy scenarios, no LLM exceeded a 25% exact-match pass rate. The dominant failure mode was not incorrect reasoning but execution non-conformance: models that understood the task could not reliably produce exact operational artifacts. O-01 identified the gap: AI systems can reason about policies but cannot execute them with the properties that operational settings require.

O-02: Can a different architecture close the gap? Experiment O-02 compared five system architectures across 3,900 controlled runs, spanning the full spectrum from raw inference to compiled execution. The finding was structural: no amount of scaffolding (structured output, validation, retries, multi-role orchestration) closed the gap to zero. Only compiled execution (Opus) achieved 100% correctness, 0% temporal misbinding, perfect determinism, and zero runtime failures. O-02 proved that the missing layer exists and that it requires a fundamentally different execution model, not a better prompt or a more elaborate agent.

O-03: Does the missing layer work at enterprise scale, and what does it enable? O-03 shifts from diagnosis to capability. Where O-02 asked *why do inference systems fail under execution constraints?*, O-03 asks *what does a compiled execution layer enable once those constraints are satisfied?* The answer is governance. Not governance as post-hoc logging, but governance as a structural property of the execution model: complete diff coverage, bit-identical replay, causal attribution, and corpus-level impact analysis, all operating over a formal representation at 30x the scale of O-02.

1.2 Execution Model Landscape

Before presenting the experimental design, it is important to establish the architectural landscape. Table 1 distinguishes four execution models along dimensions that determine governance capability.

Model	Authoring	Intelligence Location	Execution	Governance	Failure Mode
Raw LLM / Agent	Prompt or orchestration	Runtime	Probabilistic	Generated narrative	Stochastic drift, compounding inference
Rules Engine	Manual coding by domain experts	None (authoring only)	Deterministic	Logs, audit trail	Brittleness; no intelligence at any stage
Agent-Coded Artifact	LLM-generated code, frozen	Authoring (one-shot)	Deterministic	Code diff (opaque)	Opaque representation; no structural governance
Compiled Execution (Opus)	**LLM-assisted authoring**	**Authoring time**	**Deterministic**	**Structural: diff, replay, governed change analysis**	**Specification error (pre-runtime, inspectable)**

Table 1: Four execution models distinguished by where intelligence is applied, how execution proceeds, and what governance is structurally possible. The key row is *intelligence location*: agents place it at runtime, rules engines have none, and compiled execution places it at authoring time with determinism at runtime.

The critical distinction is in the *intelligence location* column. Agent systems place intelligence at runtime, which is where fragility also lives. Rules engines have no intelligence at all: they are authored manually and execute deterministically, but they cannot leverage language models for specification or synthesis. Compiled execution places intelligence at authoring time, where LLMs assist in transforming human intent into formal representations, and determinism at runtime, where execution is a pure function over a sealed artifact.

This separation is what enables governance. Because the execution artifact is a first-class semantic object (a formal intermediate representation, not opaque code or a token sequence), the system can structurally diff two versions, causally attribute decision changes to specific rule modifications, and replay any historical decision with bit-identical output.

1.3 Contributions

1. Evidence that execution guarantees are structural rather than statistical: determinism, diff completeness, and replay success hold at 1.0 across 120 workflows, 12 domains, and 2,294 test inputs, identical to the values observed at 4 workflows in O-02.

2. A demonstration of corpus-level governance: 219 of 219 known policy changes surfaced, 124 decision flips enumerated across all 12 domains, and bit-identical replay with cryptographic proof chains, all operating over a formal execution representation.

3. A clear articulation of compiled execution's position in the full pipeline from intent to governance,

distinguishing it from both agent systems (runtime intelligence, runtime fragility) and rules engines (no intelligence, no formal governance surface).

4. An honest assessment of explanatory limitations, explicitly framing governance around the primary pillars of diff completeness, replay success, and determinism, with attribution treated as a supplementary best-effort surface rather than a prerequisite for the governance claim.

2 Framing O-03 in the Full Opus Pipeline

O-01 identified the gap between AI reasoning and operational execution. O-02 proved that a compiled execution layer closes it. O-03 asks what that layer looks like as infrastructure: not for 4 workflows, but for an enterprise-scale decision surface with governance as a first-class requirement.

The Opus architecture positions compiled execution as the missing infrastructure between AI reasoning and operational consequence. The full pipeline runs from human intent to governance:

Intent → Structure → Compilation → Execution → Governance

O-03 proves the latter three stages: *Compilation → Execution → Governance*. The earlier stages (intent extraction and structural formalization) are where LLM intelligence lives. That intelligence is applied *once*, at authoring time, when domain knowledge is transformed into a formal intermediate representation (the NormalizedIR). After that transformation, runtime execution is deterministic and non-inferential. No language model is involved at execution time.

This architectural separation is the foundation of every claim in this paper. It is also the reason the infrastructure gap identified in O-01 and O-02 persists in practice: existing systems either place intelligence at runtime (agents) or omit it entirely (rules engines). Neither provides a layer that captures intelligence upstream and enforces deterministic, governable execution downstream.

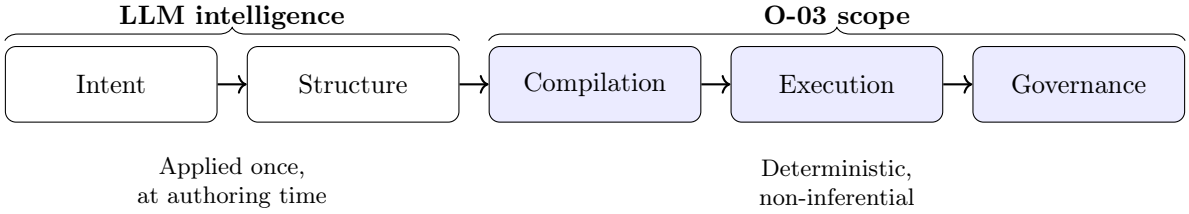

Figure 1: The full Opus pipeline. LLM intelligence operates in the Intent and Structure stages (authoring time). O-03 proves the Compilation → Execution → Governance stages, where runtime behavior is deterministic and all governance operations are structural.

2.1 What O-02 Proved: The Layer Exists

Experiment O-02 established that compiled execution produces a qualitatively different runtime profile from inference-based systems. Across 3,900 runs on 4 workflows in 3 domains, Opus achieved 100% correctness, 0% temporal misbinding, perfect determinism, and zero runtime failures. No inference-based system, including structured output with retries, multi-role agent orchestration, and agent-generated deterministic code, matched these properties on any axis.

O-02's contribution was to prove that the missing layer identified by O-01 is not hypothetical. A compiled execution architecture closes the gap between AI reasoning and operational requirements. But O-02 operated at small scale (4 workflows, 3 domains, 52 scenarios). Two questions remained: are the properties observed artifacts of careful hand-tuning, or consequences of the execution model itself? And does the layer provide governance (the ability to answer "what changed, why, and can you prove it?") at the scale enterprises require?

2.2 What O-03 Adds: The Layer Works at Scale

O-03 answers both questions at 30x the scale and shifts the thesis from failure diagnosis to governance capability:

- **120 workflows across 12 enterprise domains**, spanning finance, insurance, credit, HR, compliance, procurement, IT operations, healthcare, logistics, legal, real estate, and education.
- **Corpus-level governance**: structural diff, bit-identical replay, and causal impact analysis operating over the entire decision surface simultaneously.
- **Workflow composition**: 7 multi-step chains where determinism is preserved through sequential execution, and governance covers the full chain.
- **Double-run verification**: independent compilation runs producing identical corpus hashes, confirming reproducibility at the infrastructure level.

3 Related and Contrasting Execution Models

3.1 Agent Systems

Agent-oriented architectures place language models at runtime, using orchestration patterns such as planner–executor–validator loops, multi-role delegation, and tool-use chains. These systems are flexible and capable of complex reasoning. However, their governance surface is limited to generated explanations: narratives produced by the same inference process that made the decision. O-02 demonstrated that agent systems achieve 12.3% exact-match correctness and 70.8% temporal misbinding under controlled conditions. Their governance outputs are plausible but not structurally verifiable.

3.2 Rules Engines and Decision Tables

Deterministic execution itself is not new. Rule engines, decision tables, and policy DSLs have long provided reliable execution for structured decisions. These systems are deterministic and auditable. However, they lack two properties that compiled execution provides: (1) no intelligence is applied at any stage (authoring is entirely manual, making them brittle and expensive to scale); and (2) they typically lack a formal intermediate representation that supports structural diff, causal attribution, and counterfactual analysis as first-class operations. Their audit surface is logs and evaluation traces, not a semantic representation of the decision function.

3.3 Agent-Coded Artifacts

O-02 evaluated a hybrid approach: LLM-generated Python code, frozen at build time and executed deterministically at runtime. This approach recovers correctness and temporal binding but not governance. The execution representation is opaque code, not a formal intermediate representation. It can be diffed syntactically but not semantically. It cannot support structural attribution or invertible replay in the same way a formal execution representation can. Determinism is necessary but insufficient for governance.

3.4 Runtime Verification

Runtime verification (Leucker and Schallhart, 2009) monitors system behavior against formal specifications during execution. Compiled execution inverts this relationship: rather than verifying behavior at runtime, it guarantees behavior at compile time. The decision function is sealed before execution. Runtime is enforcement of a fixed representation, not monitoring of a probabilistic process.

4 Experimental Design

4.1 Corpus Construction

The O-03 corpus comprises 120 enterprise workflows across 12 domains, each compiled into two policy versions (v1 and v2) for a total of 240 compiled artifacts. Workflows are generated from 12 domain templates with parameterized variation, producing a controlled distribution across three complexity tiers.

Dimension	**Value**
Workflows	120
Domains	12
Compiled artifacts	240 (v1 + v2 per workflow)
Test inputs	2,294
Decision flips (v1 vs. v2)	124
Workflow chains	7
Unique condition primitives	6
Primitive reuses	1,070
Total rules	788
Total conditions	1,070
Corpus hash	`dcc42470...`

Table 2: O-03 corpus summary. All values are drawn from the frozen formal run artifacts.

Each workflow is a NormalizedIR, the same formal intermediate representation proven in O-02. The NormalizedIR is a constrained, typed intermediate representation of the decision function, consisting of explicit input schemas, ordered rule evaluation, and a closed set of condition primitives. It is designed to be fully enumerable, content-addressable, and verifiable prior to execution. The closed

primitive set comprises six operators, and every IR includes a fallback output for completeness. The compiler verifies that every possible input has exactly one output path. No ambiguity survives to runtime.

Compiled artifacts are executed through a deterministic runtime service that exposes a stable decision API for downstream systems. Execution is a pure function: $f(\text{artifact}, \text{input}) = \text{output}$. No inference, no network calls to language models, no stochastic behavior.

4.2 Domains and Complexity Tiers

The 12 domains were selected to represent the breadth of enterprise decision-making: finance, insurance, credit, HR, compliance, procurement, IT operations, healthcare administration, logistics, legal, real estate, and education. Each domain contributes 10 workflows spanning three complexity tiers.

Tier	**Count**	**Rules (mean)**	**Conditions (mean)**	**Inputs (mean)**	**Override Depth (mean)**
T1 (Simple)	29	2.6	3.5	8.3	1.3
T2 (Moderate)	58	6.0	8.0	17.4	4.7
T3 (Complex)	33	11.1	15.3	31.6	9.9

Table 3: Complexity distribution across tiers. T3 workflows have 4× the rules and conditions of T1 workflows, yet all tiers produce identical guarantee metrics.

The tier distribution mirrors a realistic enterprise portfolio: a mix of simple gates (T1), moderate multi-condition approvals (T2), and complex regulatory cascades (T3). This distribution is important because it enables the complexity flatness claim: guarantees do not degrade as structural complexity increases.

4.3 Regulatory-Grounded Anchor Workflows

To ensure the corpus is grounded in recognizable enterprise logic, a subset of workflows are derived from published regulatory or industry-standard criteria. These anchor workflows use field names, thresholds, and rule structures that a domain practitioner would recognize:

Anchor Workflow	Source Framework	Tier	Key Rules
BSA/AML screening	Bank Secrecy Act	T3	$10K CTR threshold, PEP flag, jurisdiction risk
CMS prior authorization	CMS/Medicare	T3	Step therapy, formulary tier, medical necessity
SOX control effectiveness	Sarbanes-Oxley §404	T3	Material weakness, deficiency escalation
OFAC sanctions screening	OFAC SDN List	T3	SDN match, 50% ownership rule, geographic risk
FCRA adverse action	Fair Credit Reporting Act	T2	Score-based denial, adverse action trigger
PCI-DSS classification	PCI DSS v4.0	T2	Cardholder data, encryption requirement
HIPAA minimum necessary	HIPAA Privacy Rule	T2	Role-based access, minimum necessary standard
DOT hazmat classification	49 CFR	T2	Hazard class, packing group, quantity threshold
FHA loan qualification	FHA guidelines	T3	DTI ratio (43%), credit score floor (580)

Table 4: Representative regulatory-grounded anchor workflows. These are not full regulatory implementations but capture decision-relevant rules at a fidelity that makes the workflow recognizable and defensible.

4.4 Compilation and Governance Pipeline

Each workflow passes through the same pipeline: template instantiation → NormalizedIR generation → 8-gate fail-closed validation → compilation into a sealed execution artifact → oracle output generation → governance batch (structural diff, replay, attribution). Content-addressing operates at every level: IR hash, compiled hash, oracle hash, workflow hash, and corpus hash.

The v2 policy variant for each workflow introduces parameterized threshold changes (e.g., tightening a credit limit, shortening a return window, raising an approval threshold). These changes produce the decision flips that the governance surface must detect and attribute.

4.5 Double-Run Verification

To confirm reproducibility at the infrastructure level, the full corpus was compiled independently twice. The double-run verification produced:

- Zero compiled artifact diffs between runs
- Zero workflow hash mismatches
- Identical corpus hash: `dcc424701151eebd078c231b7c339b54a7a7767af27d6ca38e9247886e63bb24`

This confirms that the compilation pipeline is itself deterministic. The same inputs to the compiler produce bit-identical outputs.

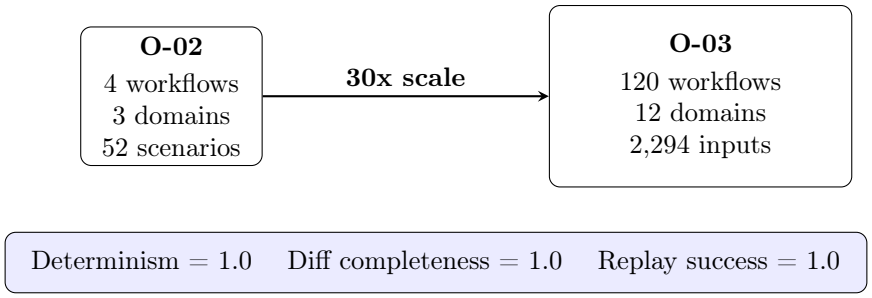

Figure 2: Scale progression from O-02 to O-03. The guarantee surface is flat across both experiments because the properties are structural consequences of the compilation model, not statistical outcomes.

5 Results

The results are organized around three primary governance pillars (diff completeness, replay success, and determinism), followed by supporting metrics that confirm the structural nature of the guarantees.

5.1 Primary Governance Pillars

5.1.1 Diff Completeness: Every Change Is Visible

The structural diff surface identifies **219 of 219** known policy changes across the corpus (diff completeness = 1.0). Every parameter override introduced in a v2 policy variant that produces a behavioral change is surfaced by the diff operation. No policy change is invisible to governance.

The diff surface operates as a high-recall enumeration: it identifies 377 total structural differences between v1 and v2 representations to guarantee complete coverage of the 219 known policy changes. The "excess" entries are surfaced because the governance surface is designed to miss nothing at the structural layer. Behavioral impact is then resolved by signal-dependent boundary probes and replay. In this sense, the diff answers *what changed structurally*; replay answers *which of those changes alter decisions on the tested surface*.

This is the headline governance result. An enterprise deploying compiled execution can change 120 policies simultaneously and receive a complete enumeration of every affected decision, before any change takes effect.

5.1.2 Replay Success: Every Decision Is Reproducible

Replay success = 1.0 across all 2,294 test inputs. Any historical decision, from any workflow, at any point in the corpus, is reproducible with bit-identical output and a cryptographic proof chain binding the input, the compiled artifact, and the output.

This is structural replay from compiled artifacts, not re-inference. The same compiled decision function, given the same input, produces the same output every time. The proof chain is: IR hash

→ compiled artifact hash → output hash. Each hash is content-addressed and verifiable.

5.1.3 Determinism: Correctness by Construction

Determinism = 1.0 across all 120 workflows. Every workflow produces identical output on every invocation for every input. This follows directly from the execution model: $f(\text{artifact}, \text{input}) = \text{output}$. Same function, same input, same output.

This is not a test result that could weaken with additional data or edge cases. It is an architectural property of the compilation model. The compiler verifies that every possible input has exactly one output path. Ambiguity is rejected at compile time, not resolved at runtime.

Pillar	Metric	Value
Visibility	Diff completeness	1.0 (219/219)
Reproducibility	Replay success rate	1.0 (2,294/2,294)
Determinism	Determinism score	1.0 (120/120)

Table 5: The three primary governance pillars. These are the metrics on which the governance claim rests.

5.2 Supporting Metrics

The following metrics confirm that the primary guarantees are structural consequences of the compilation model, not artifacts of narrow testing.

Metric	Value	Assessment
Boundary coherence	1.0	All boundary probes produce correct decision flips
Chain determinism	1.0	All 7 multi-step chains are deterministic end-to-end
Primitive consistency	1.0	All 6 condition primitives behave identically across contexts
Structural determinism	PASS	Compiler verification confirms unique output paths
Complexity flatness	FLAT	Guarantees are identical across T1, T2, and T3 tiers
Runtime failures	0	No runtime-induced failure categories observed
Supplementary / best-effort		
Attribution coverage	0.0887	See Section 8

Table 6: Supporting and supplementary metrics. Attribution coverage is a secondary, best-effort explanatory surface constrained by the `explain_change` API on boundary-exact inputs. The governance claim in O-03 rests on the three primary pillars in Table 5: complete structural visibility of policy changes, bit-identical replay, and deterministic execution.

5.3 Complexity Flatness

A critical property of compiled execution is that guarantees do not degrade with workflow complexity. T3 workflows (mean: 11.1 rules, 15.3 conditions, 9.9 override depth) produce the same

guarantee metrics as T1 workflows (mean: 2.6 rules, 3.5 conditions, 1.3 override depth). Determinism, boundary coherence, replay success, and diff completeness are 1.0 for every tier.

This is expected: the guarantees are properties of the compilation model, not of the complexity of the compiled artifact. A workflow with 15 conditions is no harder to execute deterministically than one with 3, because the compiler verifies completeness and uniqueness of output paths regardless of the number of conditions.

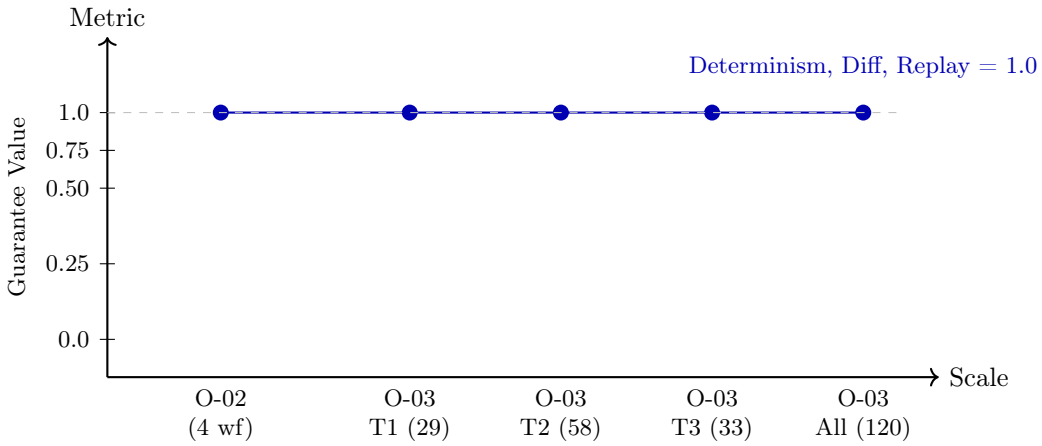

Figure 3: Guarantees vs. scale. All three primary governance pillar metrics hold at 1.0 across O-02 (4 workflows) and every O-03 tier. The flat line is the central evidence: properties are structural consequences of the compilation model.

5.4 Composition

The O-03 corpus uses 6 unique condition primitives (`eq`, `gt`, `lte`, `in`, `not_in`, `IS_NULL`) reused 1,070 times across 120 workflows. Enterprise decisions share structure: a threshold check in finance (`amount > 500`) is structurally identical to one in healthcare (`copay > 50`) or procurement (`po_value > 10000`). The compilation model exploits this through shared primitives, not through 120 independent implementations.

Seven multi-step workflow chains demonstrate that determinism is preserved through sequential composition. Each chain comprises 2–3 independently compiled workflows executed in sequence, where the output of one step maps into the input of the next. Chain determinism = 1.0: the composition of deterministic functions is deterministic.

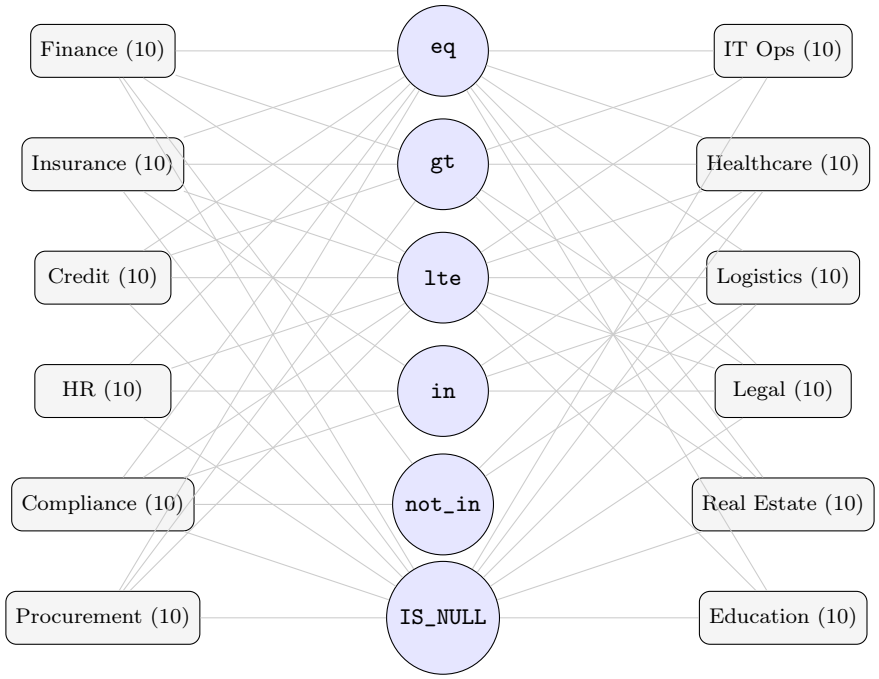

Figure 4: Primitive composition across the O-03 corpus. Six condition primitives are shared across all 12 domains. The corpus is a system with a shared execution substrate, not 120 independent implementations.

5.5 Reproducibility

Double-run verification confirms that the compilation pipeline itself is deterministic. Two independent compilation runs produce identical corpus hashes (`dcc424701151eebd...`), with zero compiled artifact diffs and zero workflow hash mismatches.

It is important to note the distinction between frozen logical surfaces and environment-local envelope metadata. Certain hash fields (e.g., `compiled_hash`) embed the output directory path and therefore vary across runs. These are classified as *environment-local envelope* metadata. The logical surfaces that constitute the governance claim, IR hashes, output hashes, and the corpus hash, are content-addressed and identical across runs. This separation strengthens the infrastructure claim: corpus identity and logical reproducibility are independent of the environment in which compilation occurs.

6 Flagship Governance Scenarios

The governance surface is not an abstract property. It answers specific operational questions that enterprises face when managing decision portfolios. This section presents three scenarios, each framed around the perspective of the person who would use it.

6.1 Scenario A: "What Happens If We Tighten These Policies?"

Perspective: Policy owner / CFO.

A policy owner is considering a set of parameter changes across the enterprise decision surface, tighter approval thresholds, shorter return windows, lower auto-approve limits. Before signing off, they need a complete answer to: *which decisions change, and how?*

The O-03 governance surface provides this answer in under 20 seconds for all 120 workflows simultaneously:

- **219 of 219** known policy changes are surfaced by the structural diff (100% coverage)
- **85 of 120** workflows are affected by the v2 parameter changes
- **124 decision flips** across all 12 domains

Domain	Affected Workflows	Decision Flips
Healthcare	10	20
Finance	10	18
IT Operations	9	13
Compliance	7	11
Credit	7	10
Logistics	8	10
Procurement	7	9
Real Estate	7	8
Education	7	8
Legal	4	7
Insurance	5	5
HR	4	5
Total	**85**	**124**

Table 7: Corpus-level policy impact by domain. Every affected workflow and every decision flip is enumerated, not estimated.

The system does not report that "healthcare might be affected." It reports that exactly 10 healthcare workflows produce 20 decision flips on specific boundary inputs under the v2 policy surface. For example, a $200 petty cash limit tightened to $150 flips one decision from APPROVE to DENY. The system identifies the governing threshold change, the exact input that crosses it, and the exact output change.

Critically, this analysis occurs *before deployment*. Governance is not reactive monitoring but pre-execution enumeration of all affected decisions under the new policy surface.

6.2 Scenario B: "Can You Prove This Decision Was Correct?"

Perspective: Auditor / compliance officer.

A regulator asks for proof that a specific historical decision was made correctly under the policy version that was in effect at the time. The enterprise must produce not just the decision, but evidence that it was the *only possible* decision given the input and the policy.

The O-03 governance surface provides this through structural replay with a cryptographic proof chain:

Element	**Value**
Workflow	`finance.petty_cash`
Input	`{department: "", receipt_present: "yes", request_amount: 0}`
Original output	APPROVE (reason: standard)
Replayed output	APPROVE (reason: standard)
Output hash match	True
Frozen proof surfaces	
IR hash	`201a08f2...`
Output hash	`e9136b35...`

Table 8: Bit-identical replay proof for `finance.petty_cash`. The auditor receives the exact input, the exact output, and a frozen proof chain binding the replay to the formal policy representation. Environment-local envelope metadata used during compilation is excluded from the paper-level replay claim.

This is structural replay from compiled artifacts, not re-inference. The same compiled decision function, given the same input, produces bit-identical output every time. The proof chain binds the intermediate representation to the compiled artifact to the output. The auditor does not receive a generated explanation of why the decision was made. They receive cryptographic proof that the decision is uniquely determined by the compiled policy representation and input, under the validated execution model.

6.3 Scenario C: "Does This Change Break Downstream Processes?"

Perspective: Operations team / pipeline owner.

An operations team is tightening a policy parameter in one workflow that feeds into a multi-step pipeline. They need to know: does this change propagate to downstream decisions? Does it break anything upstream?

The O-03 governance surface answers this through chain-level governance analysis:

Step	**Workflow**	**v1 Decision**	**v2 Decision**	**Status**
0	`tenant_screening`	APPROVE	APPROVE	Unchanged
1	`commercial_lease_approval`	DENY	DENY	Unchanged
2	`rent_adjustment`	APPROVE	DENY	**Changed**

Table 9: Chain governance for `real_estate_lease_pipeline`. A policy tightening in step 2 (`auto_approve_pct`: 3 → 2) flips the rent adjustment decision. Steps 0 and 1 are provably unaffected.

The policy change is a tightening of `auto_approve_pct` from 3 to 2 in the rent adjustment workflow. A rent increase of 3% was previously auto-approved under v1 (threshold = 3); under v2 (threshold = 2), the same increase is denied. The governance surface shows:

- **Upstream unchanged**: Steps 0 and 1 produce identical decisions under v1 and v2.
- **Downstream changed**: Step 2 flips from APPROVE to DENY.
- **Cause**: The flip is isolated to the downstream step and causally attributable to a single policy parameter change.

The operations team receives proof that the change is contained. No upstream decision is affected. The business impact is a single, identifiable flip in a single step, caused by a single parameter change. This is verifiable before deployment.

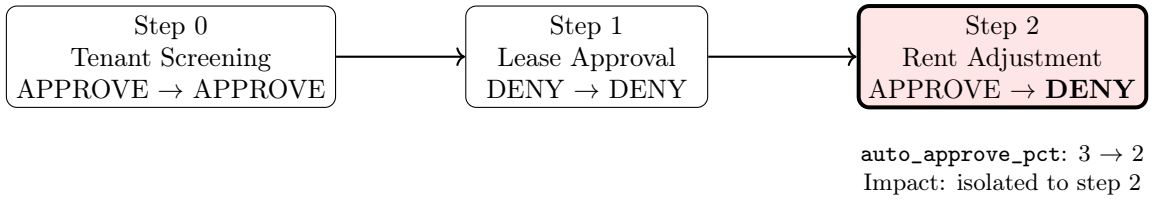

Figure 5: Chain governance for `real_estate_lease_pipeline`. Policy tightening in step 2 is isolated, causally attributed, and provably does not affect upstream steps.

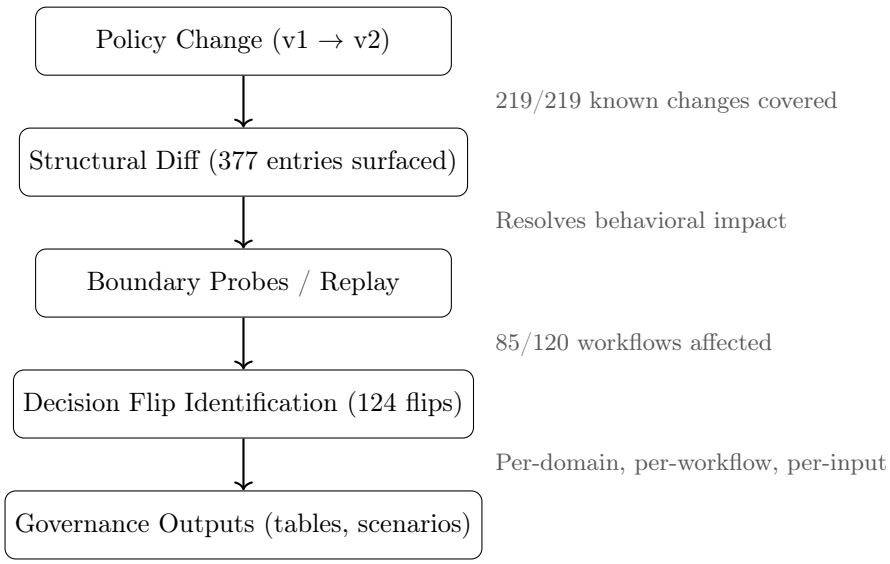

Figure 6: Governance pipeline from policy change to operational output. The structural diff provides high-recall enumeration; boundary probes and replay resolve which changes cause behavioral impact; governance outputs enumerate every affected decision.

7 Discussion

7.1 Enforcement, Not Evaluation

The central architectural claim of this paper is that compiled execution replaces evaluation with enforcement. In inference-based systems, correctness is a measured property: the system produces an output, and downstream processes evaluate whether that output is acceptable. In compiled execution, correctness is an architectural guarantee: the compiler constructs a representation in which every input maps to exactly one output, verified before runtime. There is no evaluation step because there is nothing to evaluate.

This distinction has a direct governance consequence. In evaluation-based systems, governance requires monitoring: observing outputs, logging decisions, and generating explanations after the fact. In enforcement-based systems, governance is structural: the execution representation itself supports diff, replay, and attribution as first-class operations. The governance surface is not bolted on; it is a direct consequence of the execution model.

7.2 Intelligence at Authoring Time

The argument of this paper is not anti-LLM. Language models are powerful tools for understanding intent, synthesizing policy, and generating formal specifications. The argument is about *where in the stack* that intelligence is most productively applied.

In agent systems, LLM intelligence operates at runtime. This is where its flexibility is most valuable, and where its non-determinism is most costly. Every runtime invocation is a new inference, subject to stochastic variation, temporal ambiguity, and governance opacity.

In compiled execution, LLM intelligence operates at authoring time. It assists in the transformation from human intent to formal specification. Once that transformation is complete, the formal specification is compiled into a sealed execution artifact. Runtime is deterministic and non-inferential. The LLM's contribution is preserved in the artifact's structure, not re-invoked on each execution.

This separation is what makes governance possible. You cannot structurally diff two token sequences. You can structurally diff two formal intermediate representations.

7.3 The Research Arc: From Gap to Layer to Infrastructure

The three experiments form a progressive argument. O-01 identified the gap: AI systems can reason about policies but cannot execute them reliably. O-02 proved that a compiled execution layer closes the gap with a qualitatively different runtime profile. O-03 demonstrates that this layer functions as infrastructure: scale-invariant, composable, and governance-native.

The scale-invariance is the key evidence for the infrastructure claim. The same guarantee metrics observed at 4 workflows hold identically at 120. The same boundary coherence observed across 52 scenarios holds across 2,294 test inputs. The same zero-failure surface observed in 780 cells holds across all 240 compiled artifacts. These guarantees did not degrade because they were never statistical in the first place. They are structural consequences of the compilation model. A compiled function is deterministic regardless of how many compiled functions exist in the corpus. A structural diff is complete regardless of how many workflows it covers. Replay is bit-identical regardless of

how many historical decisions exist.

The progression from O-01 to O-03 therefore moves from diagnosis (the gap exists) to proof of concept (a new layer closes it) to infrastructure validation (the layer scales with governance). Each step answers the natural objection to the previous one.

7.4 The Composition Argument

The O-03 corpus is not 120 independent scripts. It is a system built on 6 shared primitives reused 1,070 times. This composition is important for two reasons:

1. **Scaling**: Template-driven generation from shared primitives means the marginal cost of the next workflow approaches zero. Infrastructure cost is paid once; each subsequent domain template takes approximately one hour.
2. **Consistency**: Primitive consistency = 1.0 means that the same condition primitive behaves identically across all 120 workflows, regardless of domain context. A threshold check in finance operates the same way as a threshold check in healthcare.

7.5 What O-03 Does Not Prove

O-03 proves the Compilation → Execution → Governance stages of the Opus pipeline. It does not prove the Intent → Structure stages, where human intent is formalized into a specification suitable for compilation. Those stages remain future work and are where the most significant integration with LLM reasoning systems will occur.

O-03 also does not re-run the five-system comparison from O-02. It takes O-02's diagnostic finding as given and focuses on what the compiled execution layer enables at scale. The contribution is governance capability, not a repeated demonstration of inference failure.

8 Limitations

Attribution coverage. Attribution coverage is 0.0887 (11 of 124 attribution attempts succeed). This is the most important limitation to state explicitly. The current `explain_change` API produces integrity mismatches on boundary-exact inputs (inputs that sit precisely at decision thresholds), causing 113 of 124 attribution attempts to fail. This is an upstream API-level engineering limitation in the explanatory layer, not a limitation of compiled execution, replay, corpus integrity, or policy-change visibility.

The governance claim in O-03 does not rest on attribution. It rests on the three primary pillars: diff completeness (1.0, every policy change is visible), replay success (1.0, every decision is reproducible), and determinism (1.0, every output is correct by construction). Attribution is a secondary, best-effort explanatory surface. When it succeeds, it provides a human-readable causal chain linking a decision change to a specific rule modification. When it fails, the change is still visible in the diff and reproducible through replay. The limitation is in the explanation API, not in the compiled execution, not in replay, and not in corpus integrity.

Diff surface design. The structural diff surface is intentionally high-recall: it surfaces 377 entries to guarantee 100% coverage of 219 known policy changes. This means the diff is complete but over-inclusive. The "excess" entries are structural differences that do not cause decision flips on the tested boundary inputs. This is by design: a governance surface should never miss a change, even at the cost of surfacing changes that turn out to be behaviorally neutral. Resolution of which surfaced changes cause behavioral impact is performed by signal-dependent boundary probes and replay, not by the diff surface itself.

Template-driven corpus. All 120 workflows are generated from 12 domain templates with parameterized variation. This demonstrates scaling velocity but means the corpus does not include fully hand-authored edge-case workflows. The structural guarantee argument is unaffected: the guarantees depend on the compilation model, not on how workflows are authored. A hand-authored NormalizedIR passes through the same compiler and produces the same guarantee profile as a template-generated one.

No inference comparison in O-03. O-03 does not re-run the five-system comparison from O-02. It references O-02's findings and focuses on scaling and governance. The contribution is orthogonal to the failure-surface analysis.

Single system evaluation. Only compiled execution (Opus) is evaluated. No alternative governance frameworks (e.g., traditional audit systems, compliance platforms) are compared. The claim is not that compiled execution is the only way to achieve governance, but that it provides a governance surface that is a structural consequence of the execution model itself.

9 Conclusion

O-01 identified the gap between AI reasoning and operational execution. O-02 proved that a compiled execution layer closes it. O-03 demonstrates that this layer functions as governance-native infrastructure at enterprise scale.

Across 120 workflows spanning 12 domains, with 2,294 test inputs and 240 compiled artifacts, the three primary governance pillars (diff completeness 1.0, replay success 1.0, determinism 1.0) hold identically at 120 workflows as they did at 4 workflows in O-02. The guarantees are scale-invariant because they are structural, not statistical.

The governance surface answers the questions that matter to enterprises deploying decision systems at scale: *what changed* (219 of 219 policy changes surfaced), *what is affected* (85 of 120 workflows, 124 decision flips enumerated across all 12 domains), and *can you prove it* (bit-identical replay with cryptographic proof chains for every decision in the corpus). Explanatory attribution is available as a supplementary surface when the current API path succeeds, but the core governance claim rests on structural visibility and replay rather than on generated explanations.

The architectural contribution is the distinction between enforcement and evaluation. Compiled execution does not test for correctness; it enforces it. The execution representation is a sealed artifact in which every input has exactly one output path, verified at compile time. Governance is not bolted on after the fact but emerges directly from the execution model: because the artifact is a formal intermediate representation, it supports structural diff, replay, and attribution as first-class operations.

The broader pipeline from Intent to Governance remains partially proven. O-03 establishes the

Compilation → Execution → Governance stages. The Intent → Structure stages (where LLM intelligence is applied to formalize human intent) are the natural next step, and where the deepest integration between language models and compiled execution will occur. The missing infrastructure between AI reasoning and operational consequence is no longer missing in the stages where it has been tested. The remaining work is to extend it to the full pipeline.

10 References

Hendrycks, D., Burns, C., Basart, S., et al. (2021).
Measuring Massive Multitask Language Understanding.
arXiv:2009.03300.

Srivastava, A., Rastogi, A., Rao, A., et al. (2022).
Beyond the Imitation Game: Quantifying and Extrapolating the Capabilities of Language Models.
arXiv:2206.04615.

Liu, X., Yu, Y., Zhang, J., et al. (2023).
AgentBench: Evaluating LLMs as Agents.
arXiv:2308.03688.

Leucker, M., & Schallhart, C. (2009).
A Brief Account of Runtime Verification.
Journal of Logic and Algebraic Programming.

A Supplementary Tables

Appendix Table A1: Domain Summary

#	Domain	Tier Mix	Signature Logic
1	Finance	3 T1 / 5 T2 / 2 T3	Threshold stacking, currency gates, multi-condition denials
2	Insurance	2 T1 / 5 T2 / 3 T3	Waiver logic, compound eligibility, tiered evaluation
3	Credit	2 T1 / 5 T2 / 3 T3	Score thresholds, risk tiers, multi-factor assessment
4	HR	3 T1 / 5 T2 / 2 T3	Tenure gates, band compliance, eligibility windows
5	Compliance	1 T1 / 4 T2 / 5 T3	Regulatory thresholds, jurisdiction gates, escalation chains
6	Procurement	3 T1 / 5 T2 / 2 T3	Multi-level approval ladders, tolerance matching
7	IT Operations	3 T1 / 5 T2 / 2 T3	Risk classification, severity tiering, gate/auto patterns
8	Healthcare	1 T1 / 4 T2 / 5 T3	Step therapy, formulary gates, coordination rules
9	Logistics	3 T1 / 5 T2 / 2 T3	Classification logic, multi-constraint compliance
10	Legal	2 T1 / 5 T2 / 3 T3	Jurisdiction gates, temporal limits, liability math
11	Real Estate	3 T1 / 5 T2 / 2 T3	Occupancy rules, lease math, regulatory compliance
12	Education	3 T1 / 5 T2 / 2 T3	Prerequisite chains, GPA gates, credit arithmetic

Table 10: The 12 O-03 domains with tier distribution and characteristic decision logic.

Appendix Table A2: Workflow Chain Summary

Chain ID	Representative Domains	Steps	Chain Determinism
kyc_credit_payment	Compliance / Credit / Finance	3	1.0
vendor_po_payment	Procurement / Finance	3	1.0
insurance_claim_pipeline	Insurance	3	1.0
hr_onboarding_pipeline	HR / IT Operations	3	1.0
legal_merger_pipeline	Legal	3	1.0
logistics_hazmat_pipeline	Logistics	3	1.0
real_estate_lease_pipeline	Real Estate	3	1.0

Table 11: The seven workflow chains used in O-03. All chain IDs and step structures align with the frozen formal run artifacts.

Appendix Table A3: Timing Breakdown

Operation	Value
Formal run wall time	145.84 s
Corpus build time	70.91 s
Compilation latency (mean)	295.5 ms
Build throughput	1.7 workflows/s
Governance total time	19.85 s
Governance throughput	6.05 workflows/s
Average diff time	0.64 ms
Average replay time	111.45 ms
Average attribution time	3.77 ms

Table 12: Timing decomposition for the formal run. Governance operates at 6+ workflows per second across the full corpus.