A Failure-Resistant Scientific Comparison Pipeline

Ryan Bostic
Independent Researcher

2026-01-25

Abstract

Scientific computing pipelines frequently compute comparisons by default whenever numerical data are available, even when the epistemic prerequisites for comparison are ambiguous or absent. This practice risks silently converting uncertainty into inferred claims under automation.

We present a failure-resistant comparison pipeline architecture in which admissibility is treated as an explicit governance concern rather than an implicit assumption. Comparisons are executed only when authorized by a declared admissibility configuration; when authorization is absent, the pipeline deterministically emits a structured "blocked" outcome rather than a partial or inferred result.

The system separates intent declaration from enforcement, uses generated manifests to bind governance decisions to execution, and records both computed and blocked outcomes as canonical, auditable artifacts. Importantly, numerical proximity is never treated as evidence of admissible pairing.

We contribute a failure-resistant governance and execution pattern for scientific comparison pipelines: comparisons run only when explicitly authorized, and otherwise the system deterministically produces an auditable blocked outcome.

Keywords: scientific computing; reproducible workflows; epistemic governance; computational auditability; admissibility; policy-as-code; scientific software systems; failure-resistant pipelines

1 Introduction

A methodology and epistemic-governance pattern for scientific computing pipelines:

If a comparison is not explicitly admissible, the correct output is a structured refusal ("blocked"), not an inferred claim.

This paper is about *how* comparisons are governed and executed responsibly.

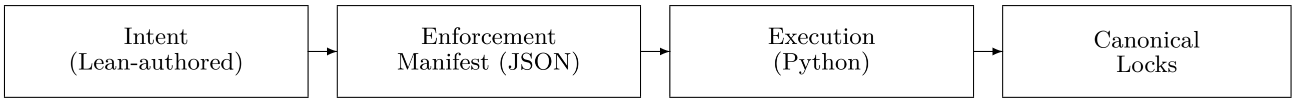
This manuscript specifies a governance and execution pattern for scientific comparison pipelines. It is evaluated by enforceable behavior under automation (authorized comparison vs blocked outcome), and it reports no new empirical results.

Contribution: a deterministic pipeline architecture that can output an auditable "no" (blocked) outcome—by design—when admissibility prerequisites are absent.

2 Positioning relative to prior work

This work sits adjacent to, but is distinct from, several existing lines of practice and research, including reproducible scientific workflows, data provenance systems, policy-as-code frameworks, and computational audit trails.

Reproducible pipelines typically emphasize determinism of execution and repeatability of results [1, 2]. Scientific workflow systems and workflow engines operationalize repeatable, automated execution in practice [3, 4]. Provenance systems focus on tracking data lineage and transformations [5]. Policy-as-code approaches encode operational constraints that govern system behavior [6]. This pipeline incorporates elements of all three, but addresses a narrower and often under-specified question: **how a scientific system should behave when a comparison is not epistemically authorized**.



Intent declares gates and defaults. Manifest is validated at runtime. Execution computes only if authorized; otherwise it blocks. Lock

Figure 1: Failure-resistant comparison pipeline: intent declaration, enforcement, execution, and auditable outcomes.

Rather than attempting to improve comparison metrics, infer equivalence, or resolve interpretive uncertainty, the system described here enforces a conservative governance posture: absent explicit admissibility, the system must refuse to compare and must record that refusal deterministically. The contribution is therefore not a new analytical method, but a failure-resistant execution pattern that **preserves epistemic boundaries even under automation**.

3 Approach overview

Many analysis pipelines will compute *something* as soon as numbers exist, even if the justification for comparing them is weak or missing.

This pipeline enforces the opposite posture:

- Comparisons are computed **only** when they are explicitly authorized by an admissibility configuration.
- If authorization is absent, the pipeline emits a **blocked** outcome with explicit blockers.
- "Numeric closeness" is never treated as a substitute for admissibility.

Governance invariant: unauthorized comparisons deterministically yield a blocked outcome with zero comparison rows.

Terminology (used consistently below): a *lane* is a data/analysis channel; a *gate* is an admissibility prerequisite; a *manifest* is the executable enforcement configuration generated from declared intent; a *lock* is a canonical, machine-generated audit artifact recording either a computed result or a blocked outcome.

4 Pipeline architecture

4.1 Architecture at a glance

Key property: the manifest is an enforcement artifact, not a policy surface.

Enforcement binding: at execution time, comparison jobs consult the manifest to determine whether a computation is authorized and record the decision (authorized vs blocked) in a canonical lock, alongside a content fingerprint for auditability.

At execution time, the pipeline validates the manifest prior to any comparison computation and deterministically aborts the comparison path if required admissibility conditions are not satisfied.

Here, "validate" means structural and consistency validation (schema and referenced fingerprints), not a cryptographic security guarantee.

Lean's role: Lean is used as a declarative intent authoring mechanism for admissibility configuration.

Why Lean: it provides a declarative, type-checked surface for intent that is amenable to structured review and deterministic downstream manifest generation.

Admissibility predicates are left venue- and domain-independent; the contribution is the enforcement pattern ("do not compare unless authorized"), not the semantics of any particular gate.

Blocked outcomes are recorded as canonical locks. Each blocked lock records a comparison identifier, the gating conditions evaluated, explicit blockers (failed gates), zero comparison rows, and a content fingerprint for auditability. This structure distinguishes refusal from missing data and supports deterministic reproduction of the refusal behavior.

4.2 Evidence artifacts (what you can inspect)

The artifacts below are machine-generated audit records and configuration surfaces produced by the pipeline; they are cited to demonstrate inspectability, not to require repository familiarity.

Examples:

- A machine-generated audit record documenting a blocked cross-lane comparison.
- A generated admissibility enforcement manifest (JSON) that binds declared intent to execution.
- A manifest updater tool that regenerates the enforcement manifest from declared intent.
- An explicit pairing-evidence mapping surface (mapping tuples); it may be empty without forcing a comparison.

5 Governance outcomes

Under this governance model, the system either computes an authorized comparison or emits a structured blocked outcome with explicit blockers.

6 Reproducibility statement (reproduces the refusal)

Goal: deterministically reproduce the **blocked** outcome (not compute a cross-lane score).

Reproduction procedure (conceptual):

1. Update the admissibility enforcement manifest from declared intent.
2. Regenerate the canonical audit locks.
3. Verify the invariant that unauthorized comparisons remain blocked and emit zero rows.

Repository-specific commands are provided in supplementary material.

Expected result:

- A representative audit record remains blocked and emits zero rows.

7 Supplementary material

Supplementary materials provide runnable code and canonical audit artifacts to support inspection and deterministic reproduction.

8 Technical specification (supplementary appendix)

Appendix A (Supplementary; frozen): detailed governance specification, claim registry, and constrained case study outline.

Appendix A is provided for deeper inspection; the journal narrative is intended to be self-sufficient for first-pass evaluation.

9 Scope and limits

This manuscript specifies governance behavior under automation rather than introducing domain results: admissibility predicates remain intentionally abstract; the pipeline records both computed and blocked outcomes as canonical locks; and unauthorized comparisons remain blocked (zero-row) rather than being inferred from numerical proximity.

10 Conclusion

This paper contributes a conservative, failure-resistant execution pattern for scientific comparison pipelines: comparisons are computed only when explicitly authorized, and otherwise the pipeline deterministically produces an auditable refusal. The emphasis is on governance behavior under automation, not on analytical novelty or domain claims.

References

[1] G. K. Sandve, A. Nekrutenko, J. Taylor, and E. Hovig, “Ten simple rules for reproducible computational research,” *PLOS Computational Biology*, vol. 9, no. 10, p. e1003285, 2013.

[2] V. Stodden, J. Seiler, and Z. Ma, “Enhancing reproducibility for computational methods,” *Science*, vol. 354, no. 6317, pp. 1240–1241, 2016.

[3] P. Di Tommaso, M. Chatzou, E. W. Floden, P. P. Barja, E. Palumbo, and C. Notredame, “Nextflow enables reproducible computational workflows,” *Nature Biotechnology*, vol. 35, no. 4, pp. 316–319, 2017.

[4] J. K"oster and S. Rahmann, “Snakemake—a scalable bioinformatics workflow engine,” *Bioinformatics*, vol. 28, no. 19, pp. 2520–2522, 2012.

[5] L. Moreau, P. Missier *et al.*, “PROV-DM: The PROV data model,” W3C Recommendation, 2013, https://www.w3.org/TR/prov-dm/.

[6] “Open policy agent (opa),” Project documentation, https://www.openpolicyagent.org/.