Glossfero and Popo: Safe Repository Cartography and a Grammar for Naming Under Exception

Flyxion
Independent Researcher

July 2026

Abstract

A personal or organizational corpus of tens of thousands of Git repositories accumulates faster than any human curator can track it, and it accumulates in a form – ad hoc directory boundaries, undocumented dependencies, duplicate and near-duplicate projects, filenames scarred by `(1)`, `(2)`, and `-final-final` – that resists both manual cleanup and naive automation. Handing an unconstrained large language model free rein to reorganize such a corpus is not a safe alternative: a single confident but wrong semantic judgment, executed directly against real repositories, is exactly the kind of irreversible action a probabilistic system should never be trusted to take unsupervised. This essay describes Glossfero, a system designed around the opposite discipline – the model interprets, measurable facts constrain, and a human authorizes every action that cannot be undone – and Popo, the naming grammar Glossfero's own object-naming problem forced into existence. Glossfero's architecture, its formal scoring and fingerprinting mathematics, and its N-version conformance methodology are presented first, followed by Popo's escalation hierarchy, formal grammar, evidence ladder, and safety properties. The essay closes with an honest account of what has actually been built and verified as of this writing, including an empirical result the N-version methodology produced rather than merely promised: four independently written implementations, in four different languages, converged on byte-identical output for the same input, and the one point on which they initially disagreed exposed a real ambiguity in the specification rather than a mere implementation bug. A final section extends both halves toward a single unifying diagnosis, provoked by examining three AI-generated diagrams of unrelated content and a companion evidence-warrant system built for photonic quantum-computing claims: a representation may not acquire stronger semantic warrant than is preserved by the constraints of its generating history, and a system safe against individual hallucination can still be unsafe against fluent, internally coherent, and entirely ungrounded semantic consensus unless that principle is enforced structurally rather than merely observed.

1 Introduction

1.1 The problem

Suppose a corpus of approximately twenty-two thousand Git repositories has accumulated over years of individual work: research code, half-finished experiments, essays converted to LaTeX, small utilities, abandoned forks, and a handful of large repositories that have quietly grown to contain several genuinely distinct subsystems bundled together because splitting them was never worth stopping to do. No single person can hold the shape of that corpus in memory. Duplicate projects exist under different names because the author forgot, eighteen months apart, that a first attempt already existed. Names have degraded into collision debris – `document (1)`, `paper-copy`, `final-final` – because the person doing the renaming, in the moment, did not have the leisure to work out what actually distinguished two similarly named things. Large repositories contain components that would be more useful, more reusable, and more comprehensible as independent repositories, but nobody has had the time to determine where the seams actually are.

This is not a problem full-text search solves. Search finds things that are already named well enough to be found; it does nothing for the repository whose name gives no hint of its contents, or the two repositories that are eighty percent the same code under different names, or the one repository that is secretly three unrelated projects. It is also not a problem an unconstrained large language model solves safely. A model can be extremely good at recognizing that a directory called `src/parser` is thematically distinct from the rest of a repository, and extremely bad at knowing, with the calibrated humility the situation demands, when it is wrong – and the action being contemplated (splitting a repository, renaming it, merging it into another) is not one that tolerates being silently wrong. Somewhere between "a human manually curates twenty-two thousand repositories" (impossible) and "a language model autonomously restructures twenty-two thousand repositories" (reckless) is the actual design space Glossfero occupies.

1.2 The governing discipline

Glossfero's answer is architectural, not merely procedural: separate the parts of the pipeline that produce a semantic judgment from the parts that produce an irreversible action, and make it structurally impossible for the former to become the latter without an explicit human decision in between. Every stage of the pipeline re-examines the current state of the corpus; nothing is cached across a decision that changes it. By default nothing is created, deleted, split, renamed, pushed, or rewritten – the system proposes and records, and *migration*, the one stage that touches real repositories, is a distinct, later, explicitly authorized phase. This discipline is stated once, precisely, and treated as the measure against which every other design decision in this essay is judged:

> The model never gets to turn a semantic guess directly into an irreversible namespace operation.

Concretely: a language model may propose a decomposition, estimate which structural type a candidate resembles, and suggest a name. It may never itself write an assignment of that name, never itself execute a migration, and never itself invoke Git. Every irreversible

action in the system is either a deterministic function of measured, reproducible facts, or it requires a recorded human decision.

1.3 Three problems, one document

This governing discipline turned out to require solving three problems that at first looked unrelated and turned out to be the same problem at three different scales. The first is the subject of Sections 2 through 4: how does a system safely propose decompositions of a large repository, verify a proposed new repository does not already exist somewhere in a twenty-two-thousand-repository catalogue, and do all of this in a way that is independently reproducible rather than dependent on the idiosyncrasies of one implementation. The second is the subject of Section 5: once a genuinely new object has been identified, what should it be *called* – and it turns out that naming, pursued seriously, is not a classification problem with five answers, but an anti-collision and exception-handling language whose vocabulary happens to include five familiar words as members, not as an exhaustive ontology. The third, the subject of Section 6, arrived from outside the system entirely, from examining why certain generated diagrams of unrelated technical content fail in a way that is fluent rather than merely wrong, and turned out to expose a gap in Glossfero's own scoring function that the first two problems had not yet forced into view: measurable structure and model-estimated semantics can be combined into a single score in a way that hides exactly the kind of disagreement between them that matters most. None of the three sections is a system followed by an appendix about naming conventions followed by a second appendix about a loosely related failure mode. Each later section is what the previous ones, examined honestly and under real pressure from a companion project, actually required.

2 The Glossfero pipeline

2.1 Stages

Glossfero's pipeline is a sequence of ten stages, each of which re-reads the current global registry state before acting, because a proposal accepted at one stage changes what counts as a collision at a later stage for every other proposal under consideration, including proposals against entirely unrelated repositories:

$$\texttt{DISCOVER} \to \texttt{MEASURE} \to \texttt{SUMMARIZE} \to \texttt{PROPOSE PARTITION} \to \texttt{GLOBAL COLLISION CHECK} \to \texttt{CLASSIFY OBJECT TYPE} \to \texttt{NAME} \to \texttt{VERIFY DEPENDENCIES} \to \texttt{SCORE PROPOSAL} \to \texttt{WRITE PLAN + LEDGER}$$

`DISCOVER` and `MEASURE` establish a reproducible fingerprint of an existing repository from Git facts alone – commit count, tracked paths, blob sizes, language composition – without parsing any language deeply. `SUMMARIZE` produces a recursive, evidence-linked description of the repository, built bottom-up from file facts through directory and subsystem summaries to a single repository summary, so that no summary at any layer is ever an unattributed guess. `PROPOSE PARTITION` generates candidate decompositions from measurable structure first – directory boundaries, package boundaries, manifest boundaries, dependency communities, historical co-change communities – before a language model is ever asked to evaluate them semantically; the model interprets candidates a deterministic process already pro-

duced, it does not invent partitions from an unstructured tree. `GLOBAL COLLISION CHECK` is the subject of Section 2.4. `CLASSIFY OBJECT TYPE` and `NAME` are the subject of Section 5. `VERIFY DEPENDENCIES` and `SCORE PROPOSAL` apply the scoring function of Section 2.3. `WRITE PLAN + LEDGER` appends an immutable, auditable record; nothing is mutated in place.

2.2 Event sourcing and the ledger

The registry's current state is never treated as though it appeared from nowhere. It is instead defined as the fold of an append-only sequence of events:

$$S_n = E_n\big(E_{n-1}(\cdots E_1(S_0)\cdots)\big) \tag{1}$$

where S_0 is the empty registry and each E_i is one of a fixed set of named event types – RepositoryDiscovered, RepositoryMeasured, SummaryGenerated, PartitionProposed, CollisionDetected, ExistingObjectReused, ObjectTypeAssigned, CandidateRenamed, HumanReviewRequested, HumanReviewAccepted, ProposalRejected, MigrationPlanned, MigrationExecuted, and RegistryUpdated – each carrying a payload validated against a fixed schema and a hash of that payload's canonical serialization. Any materialized view of the registry is rebuildable from this stream alone. This is not merely good bookkeeping practice; it is what makes the stage invariants of Section 2.5 enforceable at all, since several of them are properties of the *history* of an object, not merely its current state.

2.3 The scoring function

A proposed partition P is scored by

$$J(P) = \alpha\, C_{\text{cross}} + \beta\, H_{\text{cochange}} + \gamma\, S_{\text{imbalance}} + \delta\, A_{\text{ambiguity}} + \epsilon\, D_{\text{duplicate}} - \zeta\, C_{\text{semantic}} \tag{2}$$

with lower values preferred. C_{cross} measures dependency edges crossing the proposed boundary; H_{cochange} penalizes separating files that historically change together; $S_{\text{imbalance}}$ penalizes pathological partitions, such as one repository retaining nearly everything and dozens of others receiving only a handful of files each; $A_{\text{ambiguity}}$ penalizes candidates whose responsibility is unclear; $D_{\text{duplicate}}$ penalizes proposals that resemble an entry already present in the global catalogue; C_{semantic} rewards internal conceptual cohesion. The division of labor among these terms is itself a design commitment: only $A_{\text{ambiguity}}$ and C_{semantic} are estimated by a language model. Every other term is required to be reproducible from repository facts alone, independent of model version – which is precisely what makes C_{cross} and H_{cochange} candidates for exact-match conformance testing in Section 4, while $A_{\text{ambiguity}}$ and C_{semantic} are not.

Two of these measurable terms are defined precisely enough to be independently computed by unrelated implementations and still agree. Let R be the set of commits in a repository's history excluding its first commit, since an initial commit necessarily touches every seed file at once and carries no discriminating signal. For a candidate with path set S, define a co-change pair as two paths that both appear among the changed paths of some commit in R. Then

$$H_{\text{cochange}}(S) = 1 - \frac{|\{(p_1, p_2) : p_1, p_2 \in S\}|}{|\{(p_1, p_2) : p_1 \in S \text{ or } p_2 \in S\}|} \tag{3}$$

expressed in the codebase as `cochange_score`, the complement of this penalty: the fraction of co-change activity touching S that stays entirely internal to S. Separately, let $\mathrm{imports}(S)$ be the set of distinct top-level intra-repository packages imported by any file under S, and let $\mathrm{crossing}(S) \subseteq \mathrm{imports}(S)$ be the subset whose resolved target lies outside S. Then

$$C_{\mathrm{cross}}(S) = \frac{|\mathrm{crossing}(S)|}{|\mathrm{imports}(S)|} \tag{4}$$

expressed in the codebase as `cross_boundary_cost`. Both formulas were arrived at not as an abstract exercise but because four independent implementations, described in Section 4, needed to compute the identical number from the identical repository, and an informal description such as "penalizes separating files that change together" does not by itself guarantee that.

2.4 Fingerprints and the collision cascade

Every repository carries up to six independent fingerprints, because no single fingerprint answers the question of whether a candidate duplicates something already in the catalogue: a *name* fingerprint (normalized punctuation, case, and separators), a *structural* fingerprint (a hash of the sorted tracked-path list), a *manifest* fingerprint (a hash of recognized build and dependency manifests together with their blob hashes), a *history* fingerprint (commit ancestry, where repositories are related), a *semantic* fingerprint (derived from model-generated summaries and optionally embeddings), and a *content* fingerprint (sampled or full blob hashes). The structural and manifest fingerprints are defined precisely enough to be exact-match conformance fields:

$$\mathrm{structural} = \texttt{sha256}\Big(\bigcup\nolimits_{\mathrm{sorted}} \mathrm{tracked\_paths} \Big) \tag{5}$$

$$\mathrm{manifest} = \texttt{sha256}\Big(\bigcup\nolimits_{\mathrm{sorted}} \{\, \texttt{path} : \texttt{blob\_sha} \mid \texttt{path} \in \text{recognized manifests} \,\} \Big) \tag{6}$$

where the sorted sets are joined with newlines before hashing. A global collision check for a candidate proceeds through these fingerprints, and several cheaper checks besides, strictly in order of ascending cost – normalized name, typed name, token similarity, semantic-root lookup, embedding nearest-neighbor, manifest similarity, content overlap, Git-history relationship – and stops as soon as an earlier stage resolves the question. A language model is consulted only at the very end, and only against a bounded top-5-to-20 set of plausible matches surfaced by the cheaper stages, never against a description of the full catalogue. The result is never a boolean. It is one of nine classifications – New, ExactNameCollision, SemanticDuplicate, SemanticOverlap, RelatedDifferentObject, Ambiguous, ReuseExisting, MergeCandidate, and RequiresHumanReview – because at the scale of twenty-two thousand repositories, the difference between "this already exists" and "this is suspiciously similar to something that already exists for a reason worth investigating" is a difference a boolean cannot express and a human reviewer needs made explicit.

2.5 Stage invariants

Eight conditions halt the pipeline outright rather than allow it to proceed with degraded confidence. A proposal whose source commit has changed since analysis began must be regen-

erated, not patched. A collision check must refuse to run against a registry stale beyond a configured tolerance. A candidate classified SemanticDuplicate, SemanticOverlap, Ambiguous, or RequiresHumanReview may not proceed to naming until a recorded human decision resolves it – a semantic duplicate does not merely lower a confidence score, it stops the pipeline. A partition that would cut a dependency cycle spanning the proposed boundary halts before naming, since two repositories that cannot build independently are a structural defect, not a matter of taste deferrable to human judgment. A summary lacking evidence – manifests, imports, or representative files it can point to – is treated as an unattributed guess and rejected before it can ground anything downstream. A model response that fails schema validation is invalid, never partially accepted or best-effort repaired. No name may reach an assigned state without a valid type. And a migration that would sever a target repository's commit history from its source must halt unless a human has explicitly set an override on that specific event. These are not soft guidelines; each is a named, numbered stopping condition (Inv-1 through Inv-8) that conformance tests can reference directly.

3 N-version conformance: theory and result

3.1 Why five independent implementations

A specification that only one implementation has ever satisfied has not been tested; it has been transcribed. The risk is not merely bugs, which any testing methodology catches eventually, but a subtler failure: a specification with a genuine ambiguity – a formula that admits two equally defensible readings, a serialization rule that constrains a digit sequence but not its notation – will simply never surface the ambiguity if only one implementation ever encounters it, because that implementation's choice becomes, silently and by default, the specification. N-version programming as a discipline predates this project by decades; what Glossfero does with it is treat the same idea not as a fault-tolerance technique for production redundancy, but as a falsification method for the specification itself, applied once, deliberately, before any single implementation is allowed to become authoritative by accident.

Concretely, Glossfero is built as five independent realizations of one shared contract – implementations in Rust, Haskell, Python, and Clojure, plus a human-review front end in Elm – against a specification directory (`spec/`) that is sovereign: JSON Schema definitions for every object the pipeline produces, a set of numbered invariants, a protocol document fixing every algorithm precisely enough to be independently reproduced, and a set of golden fixtures with expected output derived by actually running the specified algorithms against real inputs, never invented by hand. Each backend implementation was written against `spec/` alone, with the deliberate constraint that no implementation's source was consulted while writing another, so that agreement between them would be evidence about the specification and not merely evidence that one author is consistent with themselves.

3.2 Exact-match and bounded-match conformance

Not every field an implementation produces can reasonably be expected to match another implementation byte-for-byte. A field derived purely from Git facts or deterministic computation – a path count, a byte count, a structural fingerprint, the `cochange_score` of Section 2.3 – must match exactly, because nothing about it is supposed to vary. A field that legitimately

depends on a language model's output – a generated summary's wording, a confidence score, a free-text justification – cannot be held to the same standard without conflating specification conformance with model determinism, which is a property Glossfero deliberately does not require of any model provider. The conformance harness therefore classifies every field in advance as exact-match or bounded-match, and diffs an implementation's output against a golden fixture accordingly: bounded-match fields are checked for presence, type, and range, never for a specific value.

3.3 A worked golden fixture

The first golden fixture, `repo-small-001`, is a real five-commit Git repository, not a synthetic description of one, constructed so that its commit history carries a deliberate co-change signature: two files that always change together, a third subsystem that changes independently of both, and a one-directional import dependency between them. Every expected value in the fixture – path count, byte count, language fractions, structural and manifest fingerprints, the `cochange_score` and `cross_boundary_cost` of a specific candidate decomposition – was computed by actually running the algorithms of Section 2 against the real repository, not typed in by hand, on the principle that a golden fixture whose expected output was invented rather than derived from the algorithm it exists to test is worse than no fixture at all.

3.4 The empirical result

Four backend implementations – Python, Rust, Haskell, and Clojure – were built independently against this fixture, each producing a `RepoRecord` for the same repository. All four agree byte-for-byte on every measured field: identical path and byte counts, identical language fractions, identical structural and manifest fingerprints, differing only in a `scanner_version` string that identifies which implementation produced the record, a field the specification never claims should match across implementations. This is the positive result the N-version methodology was built to produce: not merely that four programs compile and run, but that a written specification, absent any shared implementation code, actually constrains four independent authors to the same answer.

The result was not clean on the first attempt, and the disagreement that did occur is more informative than the eventual agreement. Haskell's default numeric formatter rendered a small language fraction in scientific notation (`3.0167597765363128e-2`) where Python and Rust both used plain decimal notation for the identical underlying value. The conformance harness's field-level diff did not catch this, because it compares parsed JSON values for semantic equality, not serialized bytes – and by that measure all three outputs were equal. But the specification's own canonical-serialization rule (Section 2 of the protocol document) requires byte-identical output for a reason independent of the field-diff harness: the ledger's event hashes are computed over exactly these bytes, so a notational difference that a semantic-equality check waves through would silently produce different hashes for logically identical events. The specification's rule – "shortest round-trip" constrains the digit sequence, not the notation – was genuinely ambiguous as written, permitting both readings; it was tightened to require plain decimal notation always, and Haskell's formatter was rewritten by hand to comply. The Clojure implementation, written afterward, hand-rolled its own canonical serializer from the outset rather than trusting a library default, specifically because of what

the Haskell case had just demonstrated. This is the N-version methodology functioning exactly as intended: a real specification gap, invisible to a single implementation and invisible to semantic-equality testing, made visible by disagreement and closed before it could propagate into the ledger's integrity guarantees.

4 Specification architecture

The repository is organized so that `spec/` can be read and implemented without reference to any particular backend: `spec/schemas/` holds JSON Schema definitions for every object exchanged between stages – `RepoRecord`, `Proposal`, `CollisionResult`, `GlossferoName`, `LedgerEvent`, and the request and response envelopes of the model-provider boundary – with JSON chosen as the canonical interchange format specifically because all target languages handle it cleanly and its serialization can be made deterministic, per Section 2's canonicalization rules. `spec/invariants/` holds the eight numbered stopping conditions of Section 2.5. `spec/protocol/` holds the algorithmic and grammatical detail of Sections 2 and 5. `spec/examples/` holds one schema-valid example instance per object type. `fixtures/` and `golden/` hold the real inputs and derived expected outputs of Section 3.3. `conformance/` holds the differ and runner that check a given implementation's output against the golden fixtures according to the exact-match and bounded-match rules of Section 3.2.

The model-provider boundary deserves particular emphasis because it is where the governing discipline of Section 1.2 is most directly at stake. Every backend wraps its language-model provider behind an interface that speaks only a fixed request and response schema; no code outside that boundary is permitted to know which provider is in use. A response that fails schema validation is invalid, full stop – never coerced, truncated, or best-effort parsed into something usable, per Inv-6 of Section 2.5. This is not defensive programming for its own sake. It is the concrete mechanism by which "the model never gets to turn a semantic guess directly into an irreversible namespace operation" is enforced rather than merely asserted: a model's output is data validated against a contract, and only a deterministic downstream process, or an explicit human decision, can act on it.

5 Popo: a grammar for naming under exception

5.1 From ontology to language

An early draft of this project treated `.bubble`, `.sphere`, `.object`, `.text`, and `.cloak` as an exhaustive ontology of repository kinds: every object gets classified into exactly one of five boxes. This framing did not survive contact with how naming actually happens in practice, and abandoning it produced something more useful than the ontology it replaced. What Glossfero's naming stage actually needs is not a taxonomy but a small, open-ended vocabulary belonging to a naming and control language – *Popo* – whose purpose is to keep ordinary names ordinary for as long as possible, and to introduce marked forms only once ordinary naming has genuinely stopped being sufficient. Popo is, at its core, an anti-collision and exception-handling language, not a type system, and the five familiar words above are members of its vocabulary rather than its complete enumeration.

5.2 The escalation hierarchy

Naming a new object, or resolving a collision with an existing one, proceeds through an ordered sequence of increasingly marked forms, and skipping ahead in this sequence is a failure of the naming procedure rather than a stylistic choice:

$$\text{(anonymous)} \prec \text{ordinary name} \prec \text{ordinary disambiguation} \prec \text{semantic synonym} \prec \text{descriptive affix} \prec \text{I} \tag{7}$$

An object begins *anonymous*: Section 5.9 argues at length that most objects should remain unnamed until something external actually requires a stable reference to them. Once naming is genuinely required, the first attempt is an *ordinary name* – a single unmarked word, `garden`. If that name is already taken by something genuinely distinct, and a compositional qualifier accurately describes the distinction, *ordinary disambiguation* appends it: `constraint-closure-note`, `constraint-closure-slides`. These qualifiers are ordinary words in a dash-composed chain and are part of the object's identity, not machinery for handling the collision – a grammatical distinction formalized in Section 5.4. If no compositional qualifier fits because the new object is not a variant of the existing one but simply a different thing that wants the same word, *semantic synonym* substitution tries another accurate ordinary word: `grove`, `orchard`, `nursery` in place of a taken `garden`. Only once ordinary semantic discrimination is genuinely exhausted does a *descriptive affix* from a small closed vocabulary get appended – `constraint-closure.bubble` – and only when an object needs to say something about how it currently participates in an operation, rather than what it is, does a *Popo operator* apply.

The ordering is a safety property. A numeric suffix such as `document (1)` does not appear anywhere in this hierarchy, because it resolves nothing: see Section 5.7.

5.3 Formal grammar

$$N ::= w \mid N\text{-}w \qquad \text{(ordinary semantic names)} \tag{8}$$

$$F ::= N\,(\,.D\,)^{*}\,(\,.A\,)^{?}\,(\,.O\,)^{*} \qquad \text{(full names)} \tag{9}$$

where w is an ordinary lowercase word; D (discriminator) is zero or more chained ordinary-word segments, open vocabulary, each further narrowing identity (`foo.parser.compat` has two); A (descriptive affix) is at most one terminal marker from the closed vocabulary of Section 5.5; and O (Popo operator) is zero or more terminal markers from an open, curated vocabulary, applied after any descriptive affix. Ordinary, non-Glossfero file-format extensions such as `.txt` or `.py` compose into this grammar the same way any other segment would; Glossfero does not own or reinterpret them.

5.4 Two grammatical positions

N and D are *noun modifiers*: they contribute to the object's ordinary identity, exactly as `-note` and `-slides` do. O is an *operator*: a predicate applied to the object that governs its participation in some presently relevant operation and asserts nothing about its content. This distinction is why a single token cannot mean both "this is a compatibility layer" – a noun-modifier claim about identity – and "exclude this from the current scan" – an operator claim about participation – at the same time, even though both readings can occupy the identical textual slot. Section 5.5 resolves exactly this conflict.

5.5 Resolving `cloak`

An earlier schema defined `cloak` as a descriptive object type meaning "overlay, adapter, projection, compatibility, or interface layer." That definition is superseded. `cloak` is a Popo *operator*, defined precisely as

$$\text{cloak}(x) \;\Rightarrow\; x \notin Q \tag{10}$$

for whatever quantified operation domain Q is presently in force – "all recognized text documents this script processes," "all repositories in this scan." Marking an object `.cloak` asserts nothing about what x is; it asserts only that x is presently excluded from Q. The canonical illustration is temporary: renaming `chapter.tex` to `chapter.tex.cloak` to exclude it from a batch operation, then removing the suffix afterward to restore it. Anything that genuinely needs to describe an object as a compatibility layer can do so through ordinary discriminator vocabulary (`foo-compat`, `foo.compat`) without needing a reserved terminal affix at all.

5.6 Operator vocabulary

Popo operators are open-ended by design. Two sub-kinds have appeared so far. *Domain-exclusion* operators remove x from a quantified operation domain without asserting anything about content, as `cloak` does above. *Escalation* operators mark x for immediate human attention independent of any quantified operation:

$$\text{escalate}(x, \text{urgency}) \;\Rightarrow\; \text{requires\_immediate\_human\_attention}(x) \tag{11}$$

with `project.pleasecallmenow` and `project.thisisanemergency` as the operative examples. Modern filesystems impose no meaningful length limit on an extension – a fact worth stating plainly, since the three- and four-character convention (`.txt`, `.exe`, `.com`) is a DOS-era 8.3-filename artifact with no analogue on ext4, NTFS, or any Git hosting backend built on them – so an escalation operator being a full, legible English phrase is a feature rather than friction.

5.7 Five independent filename signals

A single mechanism – appending a terminal marker – turns out to carry at least five signals that this project initially ran together:

$$\text{content identity} \;\neq\; \text{dispatch hint} \;\neq\; \text{operational treatment} \;\neq\; \text{preservation status} \;\neq\; \text{succession status} \tag{12}$$

Content identity is what the object actually is; nothing in a filename changes it. *Dispatch hint* claims which handler or affordance to route the object through in a particular environment, independent of content identity, and can run in either direction: `notebook.ipynb.txt` widens dispatch, exposing a Jupyter notebook through the generic text affordances of a platform with no registered `.ipynb` handler, while `chapter.tex.cloak` narrows it. *Operational treatment* is `cloak`'s job specifically, as formalized in Section 5.5. *Preservation status* constrains permissible transitions on the object rather than its interpretation, formalized in Section 5.10. *Succession status* claims which of several related objects presently holds the canonical, unqualified name and how the others relate to it in time, formalized in Section 5.12. Conflating any two of these is

where naming confusion actually originates; Windows' and mobile platforms' file-association models, which assume a file has exactly one name and that name determines exactly one owning application, answer the question "what application owns this filename" – Popo needs to answer a different question, "what affordance do I want for this object, in this operation, right now."

5.8 The evidence ladder

Filename evidence is cheap and defeasible, never authoritative, generalizing the Unix hashbang line's role as corroborating rather than exhaustive evidence about an executable:

Layer	Evidence
E_0	name
E_1	extension chain
E_2	metadata
E_3	magic bytes / hashbang / header
E_4	structural parse
E_5	documentation
E_6	whole-object interpretation

`foo.py` (E_1) is evidence that something should be treated as Python; a Python hashbang (E_3) corroborates it independently; actually parsing the file (E_4) is stronger evidence still; understanding what the program does (E_6) requires more still, and nothing below E_6 is entitled to claim it. Glossfero's own fingerprint cascade of Section 2.4 climbs exactly this ladder – name, then metadata, then structural and manifest facts, then semantic summary, then content – cheapest layer first, without having previously named the principle it was following.

5.9 Objects as predicate sets

Taken together, Sections 5.7 and 5.8 argue against modeling a Glossfero object as having one type. It should be modeled as a set of independently established predicates, each traceable to the evidence layer that grounds it:

$$x : \{\, \text{Text}, \text{LaTeX}, \text{Editable}, \text{Versioned}, \text{Summarizable}, \text{Cloaked}_Q \,\} \tag{13}$$

Apparent conflicts dissolve under this model rather than requiring arbitration. `experiment.json.txt` is not a contradiction between `.json` and `.txt`; it is an embedded-format predicate (JSON) and a requested-treatment predicate (TEXT) coexisting, neither needing to yield. This means the terminal component of a name retains precedence only for *cheap dispatch* – what a lightweight tool consults first, per E_0/E_1 – and stronger inspection at E_2 and beyond may refine or outright contradict it without that constituting an error. Polymorphism is not an edge case Popo tolerates; under this model it is the ordinary case.

An immediate consequence is that Glossfero's own pipeline stage `CLASSIFY OBJECT TYPE`, and the single `object_type` field it feeds, are now known to understate what the stage should actually produce – closer to a predicate set with per-predicate evidence provenance than one classification. This essay records that gap rather than resolving it: redesigning `CollisionResult` and `Proposal` around a predicate-set model is a materially larger schema change than the

`cloak` migration of Section 5.5, and deserves its own deliberate, separately confirmed pass rather than being folded into this one.

5.10 Preservation and the transition relation

A read-only flag makes a claim of a different kind than everything above: not what x is, but which transitions on x are admissible.

$$\text{readonly}(x) \;\Rightarrow\; x_t \not\to x_{t+1} \tag{14}$$

for unauthorized content mutation. This is independent of predicate-set membership – an object can be $\{\text{Text}, \text{LaTeX}, \text{Cloaked}_Q\}$ and separately `readonly`, and neither fact is derivable from the other – and it composes usefully with the rest of this system: a repository-wide operation can claim not merely that an object has been classified outside a mutation domain at the Popo level, but that the filesystem independently enforces the same constraint, which is a materially stronger guarantee than either representation of intent offers alone.

5.11 Collision resolution versus collision postponement

Two fundamentally different responses to a duplicate name exist, and only one of them resolves anything. *Postponement* records that a collision happened without understanding it: `document (1)`, `paper-copy-2`, `final-final`. The numeral in `document (1)` carries no information about what distinguishes the two objects, only that a naming system once failed to work it out. *Resolution* replaces postponement with an answer, using the escalation hierarchy of Section 5.2: `paper-slides` resolves what `paper (1)` merely postponed, because the actual distinction – this one is the slides – has become legible in the name itself.

This distinction yields a concrete maintenance procedure, the *number-pop pass*: discover postponement debris by pattern (`(1)`, `-copy`, `-final-final`, and similar); classify each instance as either actually redundant, checked against the fingerprint cascade of Section 2.4, or genuinely distinct; merge or delete the redundant case; and for the genuinely distinct case, discover the real distinction and rename through the escalation hierarchy, never reintroducing a numeric suffix as the resolution. Detecting the debris requires no semantic understanding of content, only a naming pattern; resolving what is found requires the fuller pipeline.

5.12 Succession

A naive versioning convention qualifies the *new* object when it supersedes an old one – `report.pdf` becomes `report-v2.pdf` – leaving anything that already refers to `report.pdf` silently resolving to the stale version. Popo's convention runs the other way: when x' supersedes x, the qualifier moves onto the *outgoing* object, and the unqualified name always tracks currency rather than a fixed artifact. This is a different relation from ordinary disambiguation (Section 5.2), which resolves collisions between simultaneously existing, genuinely distinct objects, neither more canonical than the other. Succession resolves a collision between sequential versions of what is conceptually the same object, where one side is canonical by construction, and that asymmetry is what determines which side of the rename receives the qualifier. Whether real Glossfero repository successions on a Git host should apply this same discipline – demoting a superseded repository's name rather than qualifying its replacement – is an open design question this essay records without resolving; the ledger already reserves a CandidateRenamed

event for exactly this purpose, but the protocol document does not yet specify its semantics fully.

5.13 Deferred naming and naming as capability grant

Every tier of Section 5.2 assumes naming is already necessary, and most of the time, for most objects, it is not yet. A Python lambda is not a function that happens to lack a name; the entire point of the construct is the ability to remain anonymous, to be used once, locally, without the commitment a name represents – naming a function is presenting an interface, an implicit promise of stability worth referring to elsewhere, which a single-use local transformation has no need of. The trigger for crossing from anonymous into the named tiers of Section 5.2 is *export*: the moment an object needs a stable identity outside the context that produced it, not creation, not first use, not even repeated use as long as it stays inside that originating context.

This has a sharper consequence than convenience. An unnamed, untyped object should default to the most conservative predicate set available – $\{\mathrm{Text}\}$ and nothing more, in the terms of Section 5.9 – until an explicit naming or typing commitment grants it anything richer, because an anonymous script could contain destructive instructions, and treating it as $\{\mathrm{Executable}\}$ by default before any explicit claim has been made about it is a capability escalation nobody authorized. Naming, and the fuller commitment of documentation or a type signature, is the act that *grants* capability; it is not a label applied after the fact to capability an object already had. This gives the evidence ladder of Section 5.8 a companion default-deny principle: absent evidence, assume the least capable interpretation, never the most useful one.

5.14 The formal safety property

The reason any of the preceding sections matters beyond taxonomy is a single closure property. Popo operators exist to make broad, universally quantified operations expressible and safe under local exception, without per-operation special-case logic written into every script that walks a repository:

$$\forall x \in R, \quad \neg\,\mathrm{cloak}(x) \;\Rightarrow\; \mathrm{process}(x) \tag{15}$$

Instead of abandoning a broad operation because one exceptional object would break it, or writing bespoke exclusion logic into every script that needs it, the exception is marked locally on the object itself, once, and every operation that respects the convention is automatically safe. This is not a taxonomy problem solved by clever suffixes. It is a uniform-quantification-under-exception problem solved by a lightweight, filesystem-native marking convention, and it is the actual design point of the entire naming layer.

6 Anti-signal, evidential warrant, and non-inflation

6.1 Why this matters beyond Glossfero

Recursive feedback is the reason a technological or scientific process can look, on a logarithmic time axis, like a straight line accelerating toward vertical: each new substrate does not merely store the previous layer's output, it changes the rate and criteria by which future output gets

selected. A membrane lets heritable variation persist long enough to be tested against an environment; a ribosome lets that variation express as protein complexity cheaply; a population of independent enthusiasts, in Moore's-law-era computing, searches a solution space in parallel and shares back only what survives its own informal filtering. None of this depends on the raw quantity of material produced. It depends on a filtering process that reliably separates what is worth keeping from what is not, and on that filtered residue being fed back in as the next layer's substrate.

This is precisely the mechanism a corpus of recursively generated material threatens to break, and the threat is not the one filtering was built to handle. A continuous, unfiltered record – every household broadcasting constantly, a hypothetical camera trained on an empty field for a thousand years – is not richer signal for having captured everything; it is lower signal-to-noise, because nothing has yet done the compression work of deciding what was salient. Human literature and art are already the output of that compression, which is why a corpus built from them trains something useful and an unfiltered feed would not. The danger this section addresses is a category of generated material that is not low-salience in this sense at all. It is high-salience by every heuristic a filter has learned to use, while having quietly stopped tracking the thing salience was ever supposed to be a proxy for. The three images examined below are small, legible instances of exactly that category, chosen because their failure is diagnosable in a way a subtler instance would not be.

6.2 Three failures and a recoverability progression

Three AI-generated diagrams of quantum-computational concepts – a circuit diagram, a Bloch-sphere state-vector visualization, and a phase-angle color-mapping diagram – were examined side by side, and they turn out to occupy three qualitatively different positions on an axis of recoverability, not merely three points of increasing error rate.

The circuit diagram contains a real error: a two-qubit CNOT gate is drawn as two independent single-wire boxes, each separately labeled `CNOT GATE`, rather than one gate with a control wire and a target wire. A laser wavelength annotation, $\lambda = 532\,\mathrm{nm}$, is attached to a qubit phase-shift gate, where no wavelength has any referent at all. Both errors are diagnosable because the surrounding representation still supplies enough constraints to diagnose them: "CNOT," the wire topology, and the gate-sequence grammar jointly determine a small and well-defined repair neighborhood, and the wavelength annotation can simply be rejected as inadmissible to the representation it is embedded in. The representation is wrong, but an independently grounded reference class remains available from which a correction is justified.

The Bloch-sphere diagram is more dangerous for containing no comparably alien intrusion. Its central equation, $|\psi\rangle = \cos(\theta/2)|0\rangle + e^{i\varphi}\sin(\theta/2)|1\rangle$, is correct. What has gone wrong is that genuine facts and genuine visualization conventions have been composed under the wrong equivalence relations. Phase is sometimes represented by hue as a plotting convention; the diagram treats hue as though it were constitutive of phase. The parameter θ genuinely parameterizes amplitude; the diagram describes θ as though it directly were a probability, silently dropping the squaring that relates them. Every local association in the image has genuine provenance. What has disappeared is the distinction between the represented structure and the interface chosen to expose it – and once that distinction is gone, an increasing number of locally plausible statements can be generated that are internally coherent relative to the wrong ontology, without any single statement supplying an obvious foothold for correction.

The phase-angle wheel completes the progression by assigning each computational basis state, $|00\rangle$, $|01\rangle$, $|10\rangle$, $|11\rangle$, a fixed individual phase value. This is not merely one more locally plausible error. Phase, in the quantum-mechanical formalism the diagram is nominally illustrating, is a relationship between the terms of a superposition, not a property any single basis state carries in isolation – the diagram's own superposition equation, $|\Psi\rangle = \sum_i c_i|i\rangle$, sits directly beside the error and does not repair it, because the coefficients c_i are now being read through the very ontology that needs correcting. The representation has become self-supporting under a false semantics. This yields a three-stage progression sharper than a simple error count: an error within a representation, which a surrounding grounded reference class still permits diagnosing; corruption of the distinction between a representation and the structure it represents, which allows locally plausible but ungrounded statements to proliferate; and a coherent representation of the wrong ontology, in which the visual and notational apparatus of rigor is fully present and the object it was meant to describe is simply gone.

6.3 Anti-signal

Ordinary noise tends to destroy correlations, which is exactly why it is comparatively easy for a filter to detect and discard. The failure mode illustrated above does the opposite: it preserves the surface correlates by which signal has historically been recognized – mathematical notation, polished diagrammatic conventions, explanatory labeling, terminological density, visual regularity – while severing those correlates from the generating constraints that made them trustworthy indicators in the first place. *Anti-signal* is the appropriate name for material with this property, and it is a substantially more dangerous category for a recursively trained system than ordinary noise, because a selection mechanism trained to recognize the external marks of expertise can positively select it. The generated phase wheel is not merely mistaken for a real explanation; by the filter's own learned salience criteria it can look more like a real explanation than an actual expert's hastily sketched, notationally inconsistent, but substantively correct diagram does. Where the original difficulty was distinguishing a gazelle from empty grass, the difficulty this raises is a generated object that resembles a gazelle more closely, by the filter's own criteria, than many actual gazelles do.

This connects directly to the diagnosis that generated corpora do not fail merely by containing mistakes – human corpora have always contained mistakes, and a filtering process that has always had to tolerate some error rate is not newly threatened by more of the same kind. What changes is that recursively generated material can preserve symbolic explicitness while progressively losing *constraint diversity*: each generation inherits the visible conventions of the material before it without being independently forced back against an external constraint surface – the world, an experiment, embodiment, a mathematical derivation, measured repository history – that could catch a drift the notation itself no longer reveals. The result is material that becomes simultaneously easier to recognize as expert-coded and harder to correct, because correction requires exactly the grounding the generation process has quietly stopped requiring of itself. Fluency rising while repairability falls is the compact statement of the whole problem.

6.4 From divergence detection to claim-specific licensing

A first response to this problem, adapted to Glossfero's own scoring function of Section 2.3, is to notice that $J(P)$ combines a model-estimated term, C_{semantic}, with measured terms, H_{cochange} and C_{cross}, as a single weighted sum – and a weighted sum is exactly the aggregation under

which a confidently wrong semantic estimate can be numerically compensated by otherwise-unremarkable measured structure, with the disagreement between the two channels never becoming visible. A natural first proposal is therefore a divergence detector,

$$\Delta(P) = \left| \, C_{\text{semantic}}(P) - f\big(H_{\text{cochange}}(P), C_{\text{cross}}(P)\big) \, \right|, \tag{16}$$

triggering mandatory human review whenever the semantic and structural channels disagree beyond some bound, on the same footing as Inv-3's existing treatment of an unresolved semantic duplicate: a hard stop, not a lowered confidence score.

This detector is useful but insufficient, and the reason it is insufficient is the more important result. $\Delta(P)$ is a distance between two channels, and a distance can only fire when both channels are present and disagree. It cannot detect the case in which one channel has been silently discarded, leaving nothing left to disagree with anything – which is precisely the structure of the more dangerous failure examined next.

6.5 Value and generating history

A companion formal system, developed independently as *Attestal*, a small Rust warrant-checking module for photonic quantum-computing claims, states the needed correction directly. Attestal's central move is to refuse to treat a computational result as a bare value:

$$\text{result} = \text{value} + \text{generating history}, \tag{17}$$

rather than result equals value alone. Its motivating case is a Hong–Ou–Mandel photonic interference simulation that produces an entirely correct output distribution,

$$P(|2,0\rangle) = \tfrac{1}{2}, \qquad P(|0,2\rangle) = \tfrac{1}{2}, \qquad P(|1,1\rangle) = 0, \tag{18}$$

for which the further claim "verified on real photonic hardware" is nonetheless false. The numerical channel and the provenance channel answer different questions – the result is right; this interpretation of the result is not licensed – and Attestal's implementation pattern refuses to average those two facts into a scalar confidence. A hardware-verification gate is checked as a strict predicate,

$$\text{HardwareClaim}(V, G) \text{ is admissible only when } G.\text{evidence} = \textsf{Hardware}, \tag{19}$$

and a perfectly correct simulated value receives exactly zero warrant for the hardware claim regardless of how closely it coincides with what real hardware would have produced. This is the logical structure the divergence detector of the previous section was reaching for without quite arriving at: not a comparison of two channels' scalar outputs, but a predicate over the pair of a claimed value and its generating history, $\text{warrant}(q \mid V, G)$, where a specific kind of claim q is licensed only by a specific, named kind of evidence in G – no accumulation of a different kind of evidence, however large, substitutes.

6.6 Compression, equivalence classes, and illicit inversion

Attestal's design is sharpened further by considering what happens when its output is compressed to retain only the numerical result. Several distinct generating histories can map to the same observed value,

$$g_1, g_2, \ldots, g_n \xrightarrow{\pi} v, \tag{20}$$

and once only v is retained, recovering which history actually occurred becomes set-valued, $\pi^{-1}(v) = \{g_1, \ldots, g_n\}$, rather than merely difficult. Nothing in the retained value is false; a later reader encountering the Hong–Ou–Mandel probabilities and inferring "this must have come from a real photonic processor" may even be constructing a statistically reasonable history. It is nevertheless a fabrication, and no local numerical check exposes it, because compression has removed exactly the information that would distinguish the true member of $\pi^{-1}(v)$ from the other, false but extensionally equivalent, candidates.

This is the same operation the phase-wheel diagram performs on quantum notation. Phase is sometimes visualized using hue; the diagram silently inverts that surface convention into a claim that hue-like angular position is intrinsic to phase, in exactly the way a compressed Attestal record could be silently upgraded from $P(|1,1\rangle) = 0$ to $P(|1,1\rangle) = 0$ measured on a photonic QPU. The visible convention survives; its historical relation to the object it was a convention *for* does not. This is why Attestal's negative `exclusions` field – explicit statements such as *not a Quandela QPU execution*, *not experimental detector data*, *not cryptographically signed* – is not decorative. It preserves a boundary on the space of future interpretations that the retained positive facts alone cannot preserve, which is the same protective role Glossfero's own collision classifications and stage invariants play for repository claims: not merely recording what is true, but recording what the available evidence explicitly does not license anyone to conclude next.

6.7 A concrete consequence for Glossfero

Applied back to the pipeline of Section 2, the lesson is that Glossfero's six fingerprint channels already constitute a typed evidence taxonomy in which different kinds of claim ought to require licensing by different, specific channels, rather than by the presence of evidence in general. A claim that "these files presently exhibit pattern X" is licensed by structural or co-change measurement. A claim that "these files were deliberately designed as a single unit" is a claim about generating history, licensable only by the `history` fingerprint or by documentation-layer evidence – E_5 on the evidence ladder of Section 5.8 – never by co-change regularity alone, no matter how clean that regularity looks; a model readily elides "these objects exhibit pattern X" into "these objects were produced because of X," and the two statements require radically different warrants despite identical surface evidence. This suggests a concrete schema addition once `SUMMARIZE` and `PROPOSE PARTITION` are implemented: a `claim_type` field on `ModelResponse`, checked against a fixed table mapping each admissible claim type to the one or more evidence channels permitted to license it, with an unlicensed claim rejected outright rather than accepted at reduced confidence – `require_intentional_history()` beside Attestal's own `require_hardware()`, identical shape, identical default-deny.

6.8 A non-inflation principle, stated once and instantiated three times

The pattern connecting this section to the rest of the essay is not an analogy repeated for effect. It is one structural stance appearing at three different scales of the same research program. Popo's treatment of an unnamed object (Section 5.13) defaults it to the most conservative predicate available, $\{\text{Text}\}$, until an explicit naming or typing commitment grants anything

richer – absence of a name is not neutral, it is a refusal to grant capability. Attestal's hardware gate defaults every claim to unwarranted until the one specific tagged evidence channel that can license it is present, regardless of how much other evidence has accumulated. The claim-admissibility gate proposed above for Glossfero's own `SUMMARIZE` stage defaults every semantic assertion to inadmissible until the matching evidence channel licenses it, treating inter-channel disagreement, and more importantly inter-channel *silence*, as a finding to be escalated rather than a gap to be smoothed over by aggregation. All three are the same claim:

A representation may not acquire stronger semantic warrant than is preserved by the constraints of its genera (21)

Absence of positive license is refusal, not neutrality, at the level of a filename, a physics simulation, and a repository-decomposition claim alike. This is a stronger requirement than provenance preservation on its own, because provenance preservation only guarantees that a history is recorded somewhere; the non-inflation principle additionally requires that every downstream claim be checked against the specific, named part of that history capable of licensing it, and that no quantity of the wrong kind of evidence, and no fluency in presenting it, be allowed to substitute.

This is not a principle native to Glossfero, and it should not be presented as though it were discovered here first. It is a special case, arrived at independently through repository cartography and photonic-simulation provenance rather than derived from the general theory, of what a companion essay in the same research program treats as the central object: the requirement that the apparent epistemic strength of a claim, representation, or intervention never exceed the warrant actually supplied by its supporting structure, formalized there as $R_{\text{weak}}(x, y) \not\Rightarrow R_{\text{strong}}(x, y)$ absent an independent bridge principle, and named *structural honesty* [2]. That essay's "epistemic laundering" is this essay's anti-signal considered at the scale of a single unlicensed inference rather than a whole corpus, and its promotion/externalization distinction – overstating an output's warrant versus discarding the structure by which warrant could be recovered – maps directly onto the divergence-versus-silence distinction of Sections 6.4 and 6.8. Readers who want the general theory rather than these three instances of it should look there.

7 Status and open questions

Stated plainly rather than aspirationally: as of this writing, the specification directory is complete for the scope of a first milestone – schemas for every object exchanged through `PROPOSE PARTITION`, the eight stage invariants, and a protocol document precise enough that four independent implementations converged on it, per Section 3.4. That milestone is scanning a bounded corpus of repository metadata, inspecting one repository in depth, generating a structural summary, proposing several candidate decompositions, checking them globally, assigning typed names, and writing an immutable proposal ledger, with no repository actually created or modified. Python, Rust, Haskell, and Clojure each implement `DISCOVER` and `MEASURE` against the golden fixture and pass conformance; none yet implements `SUMMARIZE` or anything requiring the model-provider boundary. The Elm review interface's types and decoders are written against the schemas of Section 4 and its actual review screen is implemented, but it has not been compile-verified, for a reason worth recording rather than glossing over: the devel-

opment environment in which it was written lacked network access to Elm's package registry, with no viable substitute route of the kind that existed for the other four languages' package ecosystems. It should be treated as unverified until built successfully.

Four migrations are consequently queued and explicitly not yet executed, on the principle, stated in the naming resolution rule of Section 2.5, that a deliberate exception to normal handling should be a recorded, deliberate step rather than a side effect of documenting the decision that motivates it: the `cloak` schema migration of Section 5.5 (removing `cloak` from the closed `object_type` enumeration, generalizing a single discriminator field into the chained sequence Section 5.3's grammar requires, and adding an open-vocabulary operator field); a materially larger redesign of `CollisionResult` and `Proposal` around the predicate-set model of Section 5.9, should that model be adopted structurally rather than left as a documented gap; a decision, not yet made, about whether real repository succession on a Git host should follow the demote-the-outgoing-name discipline of Section 5.12; and the claim-admissibility licensing table of Section 6.7, which does not yet exist in any schema and cannot be retrofitted casually once `SUMMARIZE` implementations exist independently across four languages, for the same N-version reason the naming grammar itself was written down before any backend reached `NAME`. None of these has been folded into unrelated work, and none should be.

8 Conclusion

This essay describes one discipline applied at three scales. At the scale of a whole repository corpus, the discipline is architectural: separate measurement from judgment from action, let a model produce only the middle term, and make every irreversible step require either a deterministic function of reproducible facts or an explicit human decision recorded in an append-only ledger. At the scale of a single filename, the same discipline reappears as a grammar: keep an object's identity, its dispatch treatment, its participation in the operation presently at hand, its mutability, and its succession status as five separable claims rather than one overloaded token, default every unproven claim to the least capable interpretation available, and reserve marked, exceptional vocabulary for exactly the cases ordinary naming has genuinely exhausted. At the scale of a single claim about a single piece of evidence, the discipline reappears once more as a non-inflation principle: a value is never enough on its own to license an assertion about it, only a specific, named channel of generating history can do that, and no quantity of a different kind of evidence, however fluently presented, may substitute.

None of these three was designed first and imposed on the others. The naming grammar emerged directly from taking the repository-cartography system's own naming stage seriously enough to notice that a fixed five-type ontology could not survive contact with how an author actually renames files under real collision pressure. The non-inflation principle emerged from noticing that a scoring function combining measured and model-estimated terms as a weighted sum cannot represent the difference between genuine agreement and a fluent semantic claim quietly compensating for structure that does not support it – and from a companion warrant-checking system, developed independently for an unrelated domain, that had already solved the sharper version of the same problem by refusing to scalarize a value against its generating history at all. Each time, the more general principle was discovered by taking a concrete engineering failure inside this project seriously enough to formalize, not decided on first and then illustrated.

What has actually been verified, as opposed to designed, is considerably smaller than what

this essay describes, and that gap has been stated explicitly throughout rather than allowed to blur: four independent implementations agreeing on measured facts, one specification ambiguity caught and closed by their disagreement, a naming grammar that is presently a specification document rather than a schema, and a non-inflation principle that is presently a diagnosis and a proposed schema field rather than a running invariant. The next honest step is not a larger document. It is executing the four migrations queued in Section 7 and letting a fifth implementation, or a model's own confidently wrong summary of a repository it was never entitled to characterize that way, disagree with the rest of the system about something none of it anticipated.

References

[1] Algirdas Avižienis. *The N-Version Approach to Fault-Tolerant Software*. IEEE Transactions on Software Engineering, SE-11(12), 1985.

[2] Flyxion. *Structural Honesty: The Preservation of Recoverable Warrant Across Transformations*. Independent Researcher, 2026.

[3] Martin Fowler. *Event Sourcing*. martinfowler.com, 2005.

[4] JSON Schema Organization. *JSON Schema: A Media Type for Describing JSON Documents*. Internet-Draft, 2020-12 specification.

[5] Dennis M. Ritchie. *On the Security of UNIX*. Bell Laboratories internal memorandum, 1979.