Phase 0: Contracts for a Reproducible Non-Cancellation Program

Dritëro M.

Abstract

Phase 0 defines the governance layer of the Origin Axiom program. Its purpose is not to advance a physical theory, but to make later phases auditable: claims must be bounded, evidence must be canonical and file-addressable, provenance must be recorded, and falsifiers must be stated explicitly.

Phase 0 contributes four deliverables. First, it defines a project vocabulary for claims, evidence, canonical artifacts, provenance, and phase scope contracts. Second, it introduces corridor governance: global filters (notably θ corridors) are treated as schema-validated artifacts with an append-only history, preventing silent narrowing. Third, it specifies the required structure of every claim (ID, statement, evidence pointers, non-claim boundaries, falsifiers, and provenance). Fourth, it defines a reproducibility contract, including a minimal run-bundle standard and failure-handling rules that quarantine broken evidence rather than hiding it.

Phase 0 makes no physics claims. It does not assert a value of $\theta^{\star}$, does not infer cosmology, and does not treat motivation as evidence. Its role is to provide the constitution under which later phases can make defensible, peer-review-friendly claims.

Contents

1 Introduction

This repository is organized as a governed, multi-phase research program. The role of Phase 0 is to define the governance layer: how claims are written, what counts as evidence, how phases are scoped, how reproducibility is enforced, and how failure is recorded.

What Phase 0 is. Phase 0 is a constitution for the project. It specifies: (i) a claim taxonomy and required claim structure, (ii) an evidence contract based on canonical artifacts and file pointers, (iii) phase scope contracts (including explicit non-claims boundaries), (iv) a reproducibility contract (deterministic pipelines and run provenance), (v) explicit failure modes and falsifiers, and (vi) corridor bookkeeping for global filters (notably θ-corridors).

What Phase 0 is not. Phase 0 does not make physics claims. It does not assert a specific value of $\theta^{\star}$, does not claim a mechanism for vacuum energy, does not infer cosmology, and does not treat any narrative motivation as evidence. Any physical statements belong to later phases and are valid only if they satisfy the contracts defined here.

Why governance is required. Without governance, research repositories drift in three predictable ways: claims drift (text diverges from figures), artifact drift (results live in ad-hoc folders or unreproducible runs), and reproducibility drift (code or environment changes alter outputs without a clear audit trail). Phase 0 is designed to prevent these failure modes by making the project auditable end-to-end.

Project rule: no dark progress. Work is only considered "real" when it is logged and reproducible. Each change that affects claims, figures, or phase papers must be recorded in the progress log(s) and must be recoverable from versioned inputs (code, configs, and referenced artifacts). This is not bureaucracy; it is the minimal condition for defensible scientific communication.

How to read this document. Section 2 defines the governance vocabulary. Section 3 defines corridor artifacts and their append-only history. Section 4 defines what phases may claim and how claims must be structured. Section 5 defines reproducibility requirements. Section 6 defines failure modes and falsifiers. Later phases are required to satisfy these contracts as a condition of being considered "locked".

2 Axioms and definitions (governance vocabulary)

This section defines the minimal vocabulary used throughout the project. These definitions are *governance objects*: they specify how the repository communicates results and how those results are audited. They are not physics postulates.

2.1 Core objects

Claim. A *claim* is a bounded statement asserted by a phase. A claim must be written in a standardized form (Section 4) and must be supported by evidence. Claims are the only unit of scientific assertion in this project.

Non-claim boundary. A *non-claim boundary* is an explicit list of statements that a phase does *not* assert. Non-claims are mandatory: they prevent scope creep and protect later phases from inheriting unintended conclusions.

Evidence. *Evidence* is a set of canonical artifacts (figures, tables, data summaries, run bundles) that directly support a claim. Narrative text, motivation, or conceptual analogies are not evidence.

Canonical artifact. A *canonical artifact* is an evidence object with a stable location in the repository and a reproducible provenance. Canonical artifacts are either versioned directly (e.g., PDF figures) or generated deterministically from versioned inputs under the reproducibility contract (Section 5).

Provenance. *Provenance* is the information required to reproduce an artifact: the code version, configuration/parameters, environment snapshot, and an execution identifier (run ID). Provenance must be sufficient for an independent reader to regenerate the artifact.

2.2 Phase governance

Phase. A *phase* is a governed unit of work with a declared scope, allowed claim types, and required artifacts. A phase is considered *locked* only when it satisfies the Phase 0 compliance checklist (Section 4.5).

Scope contract. A *scope contract* states what a phase is allowed to claim and what it forbids itself from claiming. Scope is binding: statements outside scope are invalid by definition.

Claims ledger. A *claims ledger* is the authoritative list of claims for a phase (or for the project), including IDs, statements, evidence pointers, and non-claim boundaries. The ledger is the primary map from text to artifacts.

2.3 Corridor governance

Filter. A *filter* is a structured, machine-readable restriction applied to a space of possibilities (e.g., parameter ranges). Filters may be introduced by phases as part of their evidence-based narrowing process.

Corridor. A *corridor* is the current authoritative allowed region for a global filter (notably the θ corridor). Corridors are represented as structured artifacts and must be updated only through an append-only history.

Corridor history. The *corridor history* is an append-only log of corridor updates. Each update must record: (i) the new corridor definition, (ii) the phase that produced it, (iii) the evidence artifacts justifying it, and (iv) the code/config provenance (run IDs and version information).

2.4 Failure and falsifiability

Failure mode. A *failure mode* is a known way a phase can fail: conceptual invalidity, numerical instability, irreproducibility, artifact drift, or evidence insufficiency. Failure modes must be stated explicitly and revisited when results change.

Falsifier. A *falsifier* is an explicit condition under which a claim should be rejected. Falsifiers may be numerical (e.g., stability breaks under parameter sweeps), logical (a contradiction with the phase contract), or empirical (in later phases, disagreement with measured quantities).

2.5 Project axiom (governance, not physics)

The only "axiom" of Phase 0 is governance:

> Work is only considered complete when it is stated as bounded claims, supported by canonical artifacts, and reproducible from versioned inputs with recorded provenance.

All physics hypotheses, models, and interpretations are deferred to later phases under these contracts.

3 Corridor method: filters, corridors, and append-only history

This project introduces *corridor governance* to prevent uncontrolled parameter drift. The corridor method treats global restrictions (notably θ-ranges) as first-class, machine-readable artifacts with append-only provenance.

3.1 Motivation (governance)

In multi-step research programs, a key parameter may be implicitly narrowed over time by narrative convention, selective figure choices, or untracked tuning. Corridor governance prevents this: any narrowing must be explicit, justified by evidence, and recorded with provenance.

3.2 Objects

Filter artifact. A *filter* is a structured, machine-readable restriction applied by a phase. Filters are saved as JSON and validated against a schema. A filter may restrict one or more variables (e.g., θ intervals), and may include a justification pointer.

Corridor artifact. A *corridor* is the authoritative, current restriction for a global variable maintained at the project level (e.g., the current θ corridor). The corridor is also a schema-validated JSON artifact.

Corridor history. The *corridor history* is an append-only log of corridor updates. The history is the source of truth for "how we got here" and must be sufficient to audit every narrowing step.

3.3 Required fields and schemas

All filter and corridor artifacts must be schema-validated. At minimum, a corridor update entry must record:

- **Target:** which variable is being restricted (e.g., `theta`).
- **Intervals:** the allowed range(s) after the update (possibly disjoint).
- **Phase provenance:** which phase produced the update.
- **Evidence pointers:** canonical artifact paths that justify the update (figures, tables, summaries).
- **Run provenance:** run ID(s) for the generating pipeline execution.
- **Code provenance:** commit hash (or equivalent version identifier) for the code state.
- **Timestamp:** when the update was made.

- **Rationale:** a short statement of the rule used for narrowing.

Schemas define the exact field names and types. A corridor update that fails schema validation is invalid.

3.4 Update rules (what is allowed)

Rule 1: No silent narrowing. A phase may not narrow a corridor by narrative convention. If a paper or claim relies on a restricted range, that restriction must exist as a filter artifact and must be recorded in the corridor history.

Rule 2: Narrowing requires evidence. A corridor may be narrowed only if the narrowing rule is supported by a canonical artifact that: (i) is reproducible under the phase's pipeline, and (ii) directly implements the stated narrowing criterion.

Examples of valid narrowing criteria include:

- a parameter sweep showing instability outside a range (numerical falsifier),
- a fit objective producing a bounded confidence region (statistical constraint),
- an explicit pass/fail test (viability filter) applied uniformly across the domain.

Rule 3: Narrowing must preserve auditability. For any corridor update, an independent reader must be able to:

1. locate the exact evidence artifacts cited,
2. locate the run bundle(s) that generated them,
3. reproduce the artifacts from versioned inputs,
4. confirm that the narrowing rule matches what the artifacts show.

Rule 4: Append-only history. The corridor history is append-only. Past corridor states are never overwritten. Reversals are allowed (widening is allowed) but must also be recorded as explicit history entries with provenance.

3.5 Artifacts in this repository

The repository maintains, at minimum:

- **Schemas:** JSON schemas for filters and corridors.
- **Current corridor:** the authoritative current corridor JSON (e.g., the current θ corridor).
- **History log:** an append-only JSONL file recording each update entry.
- **Dashboard:** a human-readable summary pointing to the current corridor and the latest update entry.

These objects exist to make phase-to-phase narrowing explicit and defensible. Later phases must treat corridor artifacts as binding constraints when a corridor exists.

3.6 Relationship to falsifiability

Corridor governance links governance to falsifiability: a narrowing is a *testable commitment*. If later work shows that a narrowed corridor fails a claim's own falsifiers, the corridor update can be reversed (widened) via an explicit history entry. This preserves scientific honesty while maintaining auditability.

4 Phase contracts: scope, claims, and evidence binding

A phase is a governed unit of work. This section defines what a phase must declare, what it is allowed to claim, and how claims bind to evidence. These contracts are mandatory: statements that violate a phase contract are invalid by definition.

4.1 Scope contract (required)

Every phase must provide a scope contract with:

1. **Scope:** what the phase is intended to establish (and at what claim strength).
2. **Non-claims:** a list of statements the phase explicitly does not assert.
3. **Primary artifacts:** the canonical outputs used as evidence (figures, tables, summaries, run bundles).

Binding rule. A phase may only assert claims that are consistent with its declared scope and non-claims. If a text statement exceeds scope or contradicts a non-claim boundary, it must be removed or moved to a later phase.

4.2 Claim taxonomy (governance types)

This project uses a small taxonomy of claim types. A phase must label the type of each claim, because type determines what evidence is required.

Existence claim Asserts that a defined effect or quantity is non-zero / present under stated conditions.

Robustness claim Asserts that an effect persists under controlled variations (parameter sweeps, numerical settings).

Bounded viability claim Asserts consistency with a comparator or constraint *within a limited test*, without claiming a full physical explanation.

Mechanism claim Asserts that a specific causal mechanism is responsible (high burden; typically deferred).

Prediction claim Asserts a falsifiable quantitative prediction for external comparison (highest burden; typically deferred).

Phases must not label claims more strongly than their evidence supports.

4.3 Required claim structure

Every claim must appear in a claims ledger and must have the following fields.

Claim ID. A stable identifier of the form `P{phase}-C{number}` (e.g., `P2-C03`). IDs are stable across edits.

Claim statement (one sentence). A single sentence that states exactly what is asserted, including the conditions under which it holds.

Evidence pointers (file paths). Explicit repository paths to canonical artifacts supporting the claim (figures, tables, data summaries). Narrative discussion does not count as evidence.

Non-claim boundary. A bullet list of statements that the claim does *not* imply. This prevents accidental over-interpretation.

Falsifiers / failure conditions. A bullet list of conditions under which the claim should be rejected (numerical instability, violation under parameter sweep, comparator failure, irreproducibility, etc.).

Provenance pointers. Where applicable, claims must reference run IDs and configuration identifiers sufficient to regenerate the evidence artifacts.

4.4 Evidence binding rule

A claim is valid only if its evidence artifacts can be located and reproduced:

1. the artifact exists at the referenced path (or is generated deterministically from versioned inputs);
2. the artifact is generated by a declared pipeline or script under a declared configuration;
3. provenance (run ID, config snapshot, code version) is recorded to reproduce it.

If any of these conditions fails, the claim must be marked as unproven (or removed) until evidence is repaired.

4.5 Phase 0 compliance checklist (reviewer-facing)

A phase is considered *locked* only if it satisfies the following checklist:

1. **Scope contract present:** scope + non-claims + primary artifacts are declared.
2. **Claims ledger complete:** every claim has an ID, one-sentence statement, evidence pointers, non-claim boundary, falsifiers.
3. **Evidence is canonical:** evidence artifacts are versioned or reproducible from versioned inputs.
4. **Reproducibility path exists:** scripts/workflow + config + provenance are sufficient to regenerate evidence.
5. **Assumptions/approximations explicit:** documented and linked to sensitivity/limitations.
6. **Failure modes explicit:** falsifiers defined; failures are logged, not hidden.
7. **Corridor governance obeyed:** any narrowing is recorded as a filter artifact and appended to corridor history with provenance.

Phase 0 satisfies this checklist by construction. Later phases must satisfy it as a condition for publication-ready claims.

5 Reproducibility contract: deterministic pipelines and run provenance

A claim is only as strong as its reproducibility. This section defines the minimum reproducibility requirements for any phase that produces evidence artifacts. These requirements are binding: if an artifact cannot be regenerated under the stated procedure, the corresponding claim must be marked as unproven until reproducibility is restored.

5.1 Principles

Determinism over convenience. Evidence artifacts must be generated by deterministic scripts or workflows using versioned inputs and explicit parameters. Interactive or manual steps are allowed only if they are fully specified and reproducible (and should be avoided for canonical artifacts).

One-click (or one-command) regeneration. Each phase must provide a single documented command (or short sequence) that regenerates all canonical artifacts for that phase from versioned inputs.

No dark progress. Evidence is not "real" until it is reproducible and logged. Cached outputs, screenshots, and ad-hoc runs are not acceptable as canonical evidence.

5.2 Canonical artifacts vs. regenerable outputs

Canonical artifacts. Canonical artifacts are the evidence objects referenced by claims (e.g., figures included in phase papers, summary tables, corridor/filter JSON artifacts). Canonical artifacts must have stable paths and must be traceable to a run provenance record.

Regenerable outputs. Large intermediate files, caches, and raw simulation outputs may be excluded from version control if they can be regenerated deterministically. If excluded, the phase must specify how regeneration occurs and how equivalence is checked (hashes, summary statistics, or figure reproduction).

5.3 Run provenance (minimum required)

Each phase must define a *run provenance record* sufficient to regenerate its canonical artifacts. At minimum, provenance must include:

- **Run ID:** a unique identifier for the execution that produced the artifacts.
- **Code version:** commit hash (or tag) identifying the exact repository state.
- **Configuration snapshot:** the full parameter/config file(s) used (YAML/JSON), stored or copied into the run bundle.
- **Environment snapshot:** package versions and runtime metadata (e.g., `pip freeze`, OS/Python version).
- **Command/workflow:** the exact command(s) used (or workflow target name), so the run can be repeated.
- **Outputs index:** a list of generated canonical artifacts and their paths.

5.4 Run bundle (recommended structure)

A *run bundle* is a directory that packages provenance and outputs for a run. A recommended minimal structure is:

```
outputs/runs/<run_id>/
config_snapshot.yaml
env_snapshot.txt
command.txt
outputs_index.json
logs/
data/ (optional, if not too large)
figures/ (canonical figures for this run)
```

Phases may extend this structure, but must preserve the minimum provenance fields.

5.5 Workflow contract

Workflow engines (example). Workflow engines may be used to enforce deterministic artifact regeneration (e.g., Snakemake), but the workflow specification and invocation must be treated as part of the evidence contract.[1, 2]

Pipeline runner. If a phase uses a workflow system (e.g., Snakemake), the workflow specification is part of the evidence contract. The phase must document how to invoke the workflow and how the workflow maps to canonical artifacts.

Parameter centralization. Phases should centralize parameters into a small number of config files (preferably one). Parameters must not be silently embedded only in code without being surfaced in a config snapshot.

Deterministic randomness. If stochastic components exist, phases must fix seeds and record them in the configuration snapshot. If deterministic operation is not possible, the phase must define a reproducibility criterion (e.g., statistical envelope, fixed summary metrics).

5.6 Reproducibility failure handling

If a canonical artifact cannot be regenerated:

1. record the failure in the phase progress log with a minimal error summary;
2. mark any dependent claims as “evidence broken” until repaired;
3. repair reproducibility by restoring missing inputs, fixing the pipeline, or updating provenance artifacts;
4. record the repair (and any changes to outputs) explicitly in the log.

This ensures the repository remains auditable even when failures occur.

6 Falsifiability and failure modes

Phase 0 requires that every phase state what would falsify its claims and how failures are handled. This is not optional: without falsifiers, a claim is not scientifically meaningful. This section defines the minimum falsifiability and failure-reporting standards.

6.1 Falsifiers (required)

A *falsifier* is an explicit condition under which a claim should be rejected. Each claim in a phase claims ledger must include falsifiers appropriate to its claim type:

- **Existence claims:** falsified if the defined effect disappears under stated conditions or collapses to numerical noise across verified precision settings.
- **Robustness claims:** falsified if the effect fails under reasonable parameter sweeps, discretization changes, tolerance changes, or seed variations (when applicable).
- **Bounded viability claims:** falsified if the comparator test fails under the declared test conditions (e.g., the claimed bounded consistency does not hold).
- **Mechanism/prediction claims:** falsified by direct contradiction with the stated mechanism tests or with external empirical comparators.

Falsifiers must be concrete. "Could be wrong" is not a falsifier. Each falsifier must correspond to a test, a diagnostic, or a comparator that can be executed or checked.

6.2 Failure modes (required)

A *failure mode* is a predictable way a phase can fail. Each phase must list its primary failure modes, and each run bundle should record any failure encountered. Common failure modes include:

1. **Artifact drift:** figures in papers do not match current pipeline outputs.
2. **Reproducibility drift:** the pipeline no longer regenerates canonical artifacts from versioned inputs.
3. **Numerical instability:** results depend discontinuously on discretization, cutoff, tolerances, or precision settings.
4. **Parameter overfitting:** results depend on untracked tuning or hidden parameter choices.
5. **Scope violation:** phase text asserts claims outside the scope contract or contradicts declared non-claims.
6. **Evidence insufficiency:** claims are asserted without direct canonical artifacts.

6.3 Failure handling rules

When a failure occurs, the correct response is governed:

1. **Record:** log the failure in the relevant progress log with a minimal error summary and the affected artifacts/claims.
2. **Quarantine:** mark any dependent claims as "broken evidence" (or remove them from the locked set) until repaired.
3. **Repair:** restore reproducibility (fix pipeline, restore missing inputs, update provenance, or revert code).
4. **Re-validate:** regenerate canonical artifacts and verify the claim-to-evidence mapping.
5. **Disclose changes:** if repaired outputs differ materially, record that difference explicitly and update claims/paper text as needed.

This procedure ensures scientific honesty while maintaining an auditable record.

6.4 Speculation policy (scope protection)

This project allows speculation only under strict labeling:

- Speculation must be explicitly labeled as *non-claim* or *future work*.
- Speculation must not be cited as evidence for any claim.
- Speculation must not weaken phase scope contracts (a phase may not "smuggle" a mechanism claim via discussion text).

A simple rule applies: if a statement is important enough to influence interpretation, it must be promoted to a bounded claim with evidence and falsifiers, or it must be demoted to clearly labeled speculation.

6.5 Relationship to corridor governance

Corridor narrowing is a scientific commitment. If a corridor update is based on a viability filter, then the filter itself must have falsifiers (e.g., demonstrate that the viability criterion is applied uniformly and reproducibly). If later work shows the narrowing rule fails its own falsifiers, the corridor must be widened via an explicit append-only history entry rather than silently rewritten.

7 Non-normative note: candidate origins (hypotheses, not claims)

This section is *non-normative*. It exists to record candidate research directions without allowing them to function as evidence or implied conclusions. No statement here is a project claim. No later phase may cite this section as evidence.

7.1 Purpose

A governance phase benefits from stating what it refuses to do. Phase 0 does not argue for a specific physical mechanism behind non-cancellation constraints. Instead, it records a small set of plausible origin routes that later phases may treat as hypotheses to test under the Phase 0 contracts.

7.2 Candidate routes (hypotheses)

1. **Symmetry / selection route.** A constraint emerges as an effective selection rule from a deeper symmetry principle, appearing as a hard floor or forbidden cancellation condition in a low-energy description.
2. **Coarse-graining / renormalization route.** Apparent cancellation at one scale fails to survive consistent coarse-graining across scales, leaving a controlled residue after integrating out degrees of freedom.
3. **Global consistency / holonomy route.** A global phase-consistency constraint restricts admissible configurations; locally canceling contributions cannot be made globally consistent, producing an irreducible remainder.
4. **Information-theoretic route.** Exact cancellation is prohibited by an encoding constraint (finite specification cost, irreducible precision, or computability limits), producing a controlled residue that behaves as an effective offset.

7.3 Governance rule

Future phases may explore any route above only if: (i) statements are written as bounded claims in the phase claims ledger, (ii) evidence is provided via canonical artifacts with run provenance, and (iii) falsifiers are stated explicitly. If a later phase cannot supply evidence, the route remains speculation and must be labeled as such.

8 Conclusion

Phase 0 defines the governance layer of this repository: how claims are stated, how evidence is bound to canonical artifacts, how phases are scoped, how reproducibility is enforced, and how failure is handled.

The central rule is simple: work is only considered complete when it is expressed as bounded claims, supported by canonical artifacts with explicit file pointers, and reproducible from versioned inputs with recorded provenance. Corridor governance prevents silent parameter narrowing by requiring schema-validated filter artifacts and an append-only history.

Later phases may propose physical models and interpretations only under these contracts. If a phase cannot provide evidence, provenance, and falsifiers, then its statements are not claims and must be labeled as speculation or future work. This is the minimal condition for an auditable, peer-review-friendly research program.

Appendices. Non-normative appendices (notation, schema templates, and worked examples) are provided in a separate companion document: `appendices.tex`.

References

[1] Johannes Köster and Sven Rahmann. Snakemake—a scalable bioinformatics workflow engine. *Bioinformatics*, 28(19):2520–2522, 2012.

[2] Johannes Köster et al. Sustainable data analysis with snakemake. *F1000Research*, 10:33, 2018.