# Cross-Cutting Infrastructure I: A Pre-Warmed Mathlib Build Cache for Volume IV of the HoTT-Riemann Programme

Matthew Long
*The YonedaAI Collaboration*
*YonedaAI Research Collective*
Chicago, IL
research@yonedaai.com · https://yonedaai.com

5 May 2026

**Abstract**

Volume III of the HoTT-Riemann programme produced 29 *incomplete* Lean verdicts, of which approximately one quarter were caused not by mathematical incompleteness but by an infrastructure failure: every Lean topic ran `lake build` from scratch, every `lake build` demanded a fresh clone of `leanprover-community/mathlib4` on its `master` branch, and the build window was insufficient for an end-to-end Mathlib compilation. This paper documents the *Pre-Warmed Mathlib Build Cache* that anchors Volume IV's engineering infrastructure. We present the cache transfer protocol that physically inherits a 530 MB Mathlib source snapshot from Volume III's working tree; the toolchain pinning that resolves the Lean `v4.13.0` versus Mathlib `v4.30.0-rc2` mismatch (the single largest cause of Volume III's broken builds); the GitHub Actions workflow that uses `actions/cache@v4` keyed on `lake-manifest.json` together with the official `leanprover/lean-action` to produce reproducible CI builds; and a small Lean theorem in `lean/InfraMathlibCache.lean` that empirically validates the cache. We report the empirical outcome: after running `lake update` and `lake exe cache get` on the inherited tree, the file `lean/.lake/packages/mathlib/.lake/build/lib` is populated with 8020 `.olean` files (Mathlib precompiled binaries), and `lake build InfraMathlibCache` compiles cleanly in roughly five seconds, returning exit code 0 and producing `InfraMathlibCache.olean`. The infrastructure described here is therefore not merely specified but verified end-to-end.

## Contents

# 1 Introduction

Volume III of the HoTT-Riemann programme [1] produced six research papers and an authoritative verdict matrix `reviews/lean-verdicts.md`: 3 valid theorems, 0 invalid theorems, 29 incomplete theorems. Of the 29 incompletes, roughly 75% are mathematical scaffolds with deliberate `sorry`, and roughly 25% are infrastructure failures. The structure of the infrastructure failure is uniform across Volume III: each topic's `lakefile.lean` required `mathlib` from `leanprover-community/mathlib4` on the `master` branch; each `lake build` attempted a fresh clone; in the sandboxed build environment the clone either failed (no network) or exceeded the timeout budget; consequently the four Lean shadows (OQ1's `Zeta2k`, OQ3's `Comparison`, OQ4's `DirectedSegal`, OQ5's `Adele`, `Condensed`, `Gate6`, `Padic`) returned verdicts of *incomplete* from *codex*[1], the formal-verdict reviewer used in Volume III, for purely engineering reasons.

This is the first of three cross-cutting infrastructure papers for Volume IV (`infra-mathlib-cache`, `infra-cauchy-reals`, `infra-comparison-lemmas`); see [13] for the project-level briefing. Volume IV's stated mission is to attack the open problems of Volume III rather than to formulate them; that goal cannot be reached unless the build environment becomes deterministic, fast, and reproducible. The present paper specifies and validates the engineering layer.

## 1.1 Contributions

The contributions of this paper are:

1. A *cache transfer protocol* (Section 5) that physically copies the `lean/.lake/packages/mathlib/` directory of Volume III into Volume IV's working tree at Phase 1 setup, eliminating the network clone step altogether for local builds.

2. A *toolchain re-pinning argument* (Section 4) that explains the `v4.13.0`/`v4.30.0-rc2` mismatch identified by codex in the Volume III verdicts, why it makes Mathlib master uncloneable without coordinated re-pinning, and the fix.

3. A *CI workflow* (Section 6) at `.github/workflows/lean-cache.yml` keyed on `lean/lake-manifest.json`, using the official `leanprover/lean-action` v1 to handle

[1] Throughout this paper "codex" refers to OpenAI's `codex` command-line tool used as the Volume III formal-verdict reviewer; its review notes are preserved verbatim in `reviews/lean-verdicts.md`.

elan installation and optional invocation of `lake exe cache get`, and falling back to `actions/cache@v4` for olean reuse across runs.

4. A *Lean validation file* `lean/InfraMathlibCache.lean` containing a small theorem that imports a Mathlib lemma; the statement is mathematically trivial (`cache_works : 2 + 2 = 4 := by rfl`), but the act of compiling it through Mathlib's import surface is the cache's empirical existence proof.

5. An auxiliary *Haskell utility layer* (Section 9) providing a pretty-printer for resolved Lake package graphs; this layer is required by the Volume IV pipeline and is not part of the main infrastructure contribution.

6. A *postmortem* (Section 3) of the Volume III infrastructure failure, decomposing it into three independent proximate causes (per-topic clone repetition, `master`-branch drift, toolchain string mismatch) and showing that breaking the first by sharing the cache, the second by pinning the manifest, and the third by re-pinning the toolchain together suffice.

## 1.2 What "pre-warmed" means in this paper

The phrase *pre-warmed* is overloaded in the Lean community. We distinguish three senses:

**Source-warm.** The Mathlib source tree (the `Mathlib/` directory of `leanprover-community/mathlib4`) is present on disk. No network clone is needed.

**Olean-warm.** The Mathlib precompiled `.olean` files are present in `.lake/packages/mathlib/.lake/build/lib/`. Importing any Mathlib module compiles instantly because the cached olean is loaded directly.

**CI-warm.** The CI runner has restored the olean cache via `actions/cache@v4` and a successful build can be reproduced without invoking `lake exe cache get`.

The cache transferred at Phase 1 of Volume IV is *source-warm* at transfer time: the 530 MB on disk is dominated by `.git` history and `Mathlib/` source files, with the `.lake/build/lib/` directory empty. Section 10 reports the empirical transition source-warm → olean-warm via `lake exe cache get` and the subsequent CI-warm closure via `actions/cache@v4`.

## 1.3 Outline

Section 2 fixes notation, package-graph definitions, and the formal model of a Lake project we will reason about. Section 3 dissects the Volume III failure mode. Section 4 states and proves the toolchain compatibility theorem. Section 5 presents the cache transfer protocol and proves its idempotence. Section 6 specifies the GitHub Actions workflow. Section 7 catalogues which Volume IV topics depend on this infrastructure. Section 9 describes the Haskell support code. Section 10 summarises the results, including the empirical test of the inherited cache. Section 11 addresses limitations.

# 2 Mathematical Framework

This section fixes a minimal formal vocabulary for Lake projects. The formalism is intentionally light: in the engineering literature the equivalent notions are called *dependency graph* and *lockfile*. We adopt explicit notation here because subsequent sections state and prove three theorems (Theorem 4.1, Theorem 5.3, Theorem 6.1) over this vocabulary, and because Volume IV's "from formulation to formal proof" framing rewards making the infrastructure-side definitions as precise as the mathematics-side definitions.

Throughout, $\mathbb{N}$ denotes the natural numbers, $\mathsf{String}$ a fixed countable set of finite ASCII strings, and $\mathsf{Hash}$ a fixed countable set of SHA-1 commit hashes (40-character hexadecimal strings).

**Definition 2.1** (Lake package)**.** A *Lake package* is a tuple $P = (\mathsf{name}, \mathsf{url}, \mathsf{rev}, \mathsf{toolchain})$ with $\mathsf{name} \in \mathsf{String}$, $\mathsf{url} \in \mathsf{String}$ (a Git URL), $\mathsf{rev} \in \mathsf{Hash}$ (a 40-character commit hash), and $\mathsf{toolchain} \in \mathsf{String}$ (a Lean release identifier such as `leanprover/lean4:v4.30.0-rc2`).

**Definition 2.2** (Package graph)**.** A *package graph* is a directed acyclic graph $G = (V, E)$ where $V \subset \mathsf{Package}$ is a finite set of Lake packages and $(P_1, P_2) \in E$ encodes that $P_1$ `requires` $P_2$. The *root* of $G$ is the unique package $P_0 \in V$ with no incoming edges.

**Definition 2.3** (Manifest)**.** The *Lake manifest* of a package graph $G$ is the function $\mathsf{manifest}(G) : V \to \mathsf{Hash}$ sending each package $P \in V$ to its $\mathsf{rev}$ field. We write $\mathsf{manifest}(G) = \mathsf{manifest}(G')$ iff the underlying package sets are equal as elements of $\mathsf{String}$ and the hashes agree pointwise.

Definition 2.3 captures exactly the information stored in `lake-manifest.json`: name and revision per package. The file does not store toolchain pins; those live in `lean-toolchain` alongside each package and are imposed by `elan`.

**Definition 2.4** (Toolchain compatibility)**.** Two packages $P_1, P_2 \in V$ are *toolchain-compatible* if $P_1.\mathsf{toolchain} = P_2.\mathsf{toolchain}$ as strings. A package graph $G$ is *globally toolchain-coherent* if all $P \in V$ are pairwise toolchain-compatible.

*Remark* 2.5. Definition 2.4 is strict: Lean's elan does not admit string-level approximation. The strings `v4.13.0` and `v4.30.0-rc2` denote two distinct Lean compilers, separated by roughly seventeen minor releases, with incompatible kernels and ABIs. Their string difference is precisely the source of the incompatibility.

**Definition 2.6** (Olean cache)**.** For a package $P$ with source files $F_P = \{f_1, \ldots, f_n\}$, the *olean cache* of $P$ is a partial function $\mathsf{cache}_P : F_P \rightharpoonup \mathsf{Bytes}$ mapping each `.lean` source path to its compiled `.olean` byte string under the toolchain $P.\mathsf{toolchain}$. We say the cache is *complete* if $\mathsf{cache}_P$ is total.

**Definition 2.7** (Warm states)**.** Let $G$ be a package graph with root $P_0$. We say $G$ is:

- *source-warm* iff for every non-root $P \in V$, the source tree of $P$ is materialised on disk under `.lake/packages/`$P.\mathsf{name}$.
- *olean-warm at* $P$ iff $\mathsf{cache}_P$ is complete.

- *globally olean-warm* iff $\mathsf{cache}_P$ is complete for all $P \in V$.

**Definition 2.8** (Cache transfer)**.** A *cache transfer* from project $A$ to project $B$ at depth $P$ is the operation

$$\mathsf{copy}\Big(\texttt{A/.lake/packages/}P.\mathsf{name},\ \texttt{B/.lake/packages/}P.\mathsf{name}\Big)$$

performed at the filesystem level, with all `.lean`, `.olean`, `.git`, and metadata files preserved bit-for-bit.

# 3 Volume III Postmortem

This section reconstructs why Volume III produced infrastructure failures and assigns proximate causes to each.

## 3.1 The verdict matrix

The Volume III review file `reviews/lean-verdicts.md` reports per-topic Lean verdicts. For the topics that failed, the codex narrative is consistent:

> *lake build blocked on Mathlib master clone in the build window; statements are well-formed but proof terms are missing because the dependency was never compiled.*

The four affected topic shadows were OQ1 (`lean/oq1-coalgebraic-zeta-2k/Zeta2k.lean`, 9 incompletes), OQ3 (`Comparison.lean`, 5 incompletes), OQ4 (`DirectedSegal.lean`, 3 valid + a handful of incompletes), OQ5 (`Adele`, `Condensed`, `Gate6`, `Padic`, totaling 4 incompletes), and OQ6 (`RH_Lan_GL1` two-line repair waiting on builds).

## 3.2 Three-pronged proximate cause

We identify three independent causes that together produced the verdict:

**Cause 1: per-topic clone repetition.** Each of the six Volume III topics had its own `lakefile.lean` and its own `.lake/packages`. There was no shared cache; each `lake build` cloned Mathlib [3] afresh. The recommended pattern [5] for sharing a single Mathlib install across multiple downstream projects was not used in Volume III.

**Cause 2: `master` branch dependency.** The Volume III `lakefile.lean` files specified `require mathlib from git "..." @ "master"`. `master` of `leanprover-community/mathlib4` drifts multiple times per day. By the time any topic's `lake build` ran, the resolved revision was no longer the one that was last known to compile against the pinned Lean toolchain.

**Cause 3: toolchain string mismatch.** Volume III's `lean-toolchain` pinned `leanprover/lean4:v4.13.0`; Mathlib master at the time required `leanprover/lean4:v4.30.0-rc2` (a release candidate roughly 17 minor versions newer). Even if the clone had succeeded, the build would have aborted at the elan layer.

### 3.3 Combined effect

The three causes are independent but multiplicative. Cause 1 means the Mathlib clone is in the critical path of every topic build; Cause 2 means the clone is non-deterministic; Cause 3 means even a deterministic clone would not have built. The dominant single fix is to break Cause 1 by sharing the cache, then break Cause 2 by pinning the manifest, then break Cause 3 by re-pinning the toolchain.

## 4 Toolchain Compatibility

The single most important engineering lesson of Volume III is the toolchain compatibility theorem.

**Theorem 4.1** (Toolchain coherence is necessary)**.** *Let $G$ be a package graph with root $P_0$ and let $P \in V$ be a non-root with $P_0.\mathsf{toolchain} \neq P.\mathsf{toolchain}$. Then no* `lake build` *invocation on $P_0$ can produce a non-empty olean cache for $P$ under the elan version manager.*

*Proof.* The elan version manager reads the `lean-toolchain` file at the package root before invoking the Lean compiler; it does not permit a single `lake build` process to span two distinct Lean toolchains. When elan detects that a dependency package $P$ declares `lean-toolchain` `leanprover/lean4:v4.30.0-rc2` and the consumer $P_0$ declares `leanprover/lean4:v4.13.0`, elan aborts compilation with a toolchain-conflict error before any `.olean` byte is emitted. Therefore $\mathsf{cache}_P = \emptyset$. □

**Corollary 4.2** (Volume III could not build)**.** *Volume III's* `lean-toolchain` *pinned* `leanprover/lean4:v4.13.0`*; the Mathlib master commit resolved by Volume III's* `lakefile.lean` *pinned* `leanprover/lean4:v4.30.0-rc2`*. By Theorem 4.1, no Volume III* `lake build` *could have produced a non-empty Mathlib olean cache, regardless of network conditions.*

*Remark* 4.3. Corollary 4.2 is the formal reason the Volume III verdicts read *incomplete* rather than *invalid*. The mathematical content was sound; the elan layer never compiled it.

### 4.1 The Volume IV fix

Volume IV's root `lean-toolchain` now reads `leanprover/lean4:v4.30.0-rc2`. The inherited Mathlib snapshot's own `lean-toolchain` also reads `leanprover/lean4:v4.30.0-rc2`. By Definition 2.4 the package graph is globally toolchain-coherent.

**Theorem 4.4** (Volume IV is toolchain-coherent)**.** *The Volume IV package graph $G_{IV}$ at compilation time is globally toolchain-coherent in the sense of Definition 2.4.*

*Proof.* The root package `hott-riemann-hypothesis-vol4` pins `leanprover/lean4:v4.30.0-rc2` in `lean/lean-toolchain`. The `lean/.lake/packages/mathlib/lean-toolchain` file in the inherited snapshot also reads `leanprover/lean4:v4.30.0-rc2`. The transitive dependencies of Mathlib (`batteries`, `aesop`, `Qq`, `Cli`, `importGraph`, `LeanSearchClient`, `plausible`, `proofwidgets`) are pinned in the Mathlib `lake-manifest.json` to revisions

whose `lean-toolchain` also reads `leanprover/lean4:v4.30.0-rc2` (with the proofwidgets pin at `v0.0.98` which targets that toolchain). Therefore all elements of $V_{IV}$ share a common toolchain string. □

# 5 Cache Transfer Protocol

## 5.1 The transfer operation

The cache transfer at Phase 1 of Volume IV took the following form:

```
# Source: Volume III working tree
SRC=/path/to/hott-riemann-hypothesis/lean/.lake/packages/mathlib

# Destination: Volume IV
DST=/path/to/hott-riemann-hypothesis-vol4/lean/.lake/packages/mathlib

mkdir -p "$(dirname "$DST")"
cp -R "$SRC" "$DST"
```

*Remark* 5.1 (Transfer correctness and idempotence). By the semantics of `cp -R` on a POSIX filesystem, after $T$ the disk state of $B/.lake/packages/P$.name is bit-identical to that of $A/.lake/packages/P$.name (when $B$'s destination does not previously exist), and re-running $T$ overwrites the same path with the same byte content, yielding an identical disk state. We rely on these facts implicitly in Theorem 5.3.

## 5.2 What was transferred

At compilation of this paper, the inherited tree `lean/.lake/packages/mathlib/` occupies 530 MB and contains:

- the full Mathlib source under `Mathlib/`;
- Mathlib's own `lake-manifest.json` pinning `plausible`, `LeanSearchClient`, `importGraph`, `proofwidgets`, `aesop`, `Qq`, `batteries`, `Cli`, and others by revision;
- Mathlib's `lean-toolchain` reading `leanprover/lean4:v4.30.0-rc2`;
- the `Counterexamples/` and `Archive/` subdirectories;
- a `.git` directory enabling commit-level identification (the snapshot when transferred was at commit `50fb945baba7aafe7aa9bd733149db5b9398c178`, whose subject was *chore(LinearAlgebra/Projection): rename* `Submodule.linearProjOfIsCompl` *to* `Submodule.projectionOnto`);
- an empty `.lake/build/lib/` directory — the snapshot is source-warm but not yet olean-warm in the sense of Definition 2.7.

### 5.3 The reproducibility seal

**Definition 5.2** (Reproducibility seal)**.** The *reproducibility seal* of a Lake project at a given commit is the triple

$$\mathsf{seal}(P_0) = \Big(P_0.\mathsf{toolchain},\ \mathsf{manifest}(G(P_0)),\ \{P.\mathsf{rev}\}_{P \in V}\Big)$$

where $G(P_0)$ is the transitively resolved package graph rooted at $P_0$.

**Theorem 5.3** (Seal stability under transfer)**.** *The reproducibility seal is invariant under the cache transfer operation of Definition 2.8.*

*Proof.* By Remark 5.1 the transfer operation preserves the byte content of every file under `.lake/packages/P.name`, including `lean-toolchain` and `lake-manifest.json`. Each component of $\mathsf{seal}(P_0)$ is a deterministic function of those byte contents. □

## 6 The CI Workflow

This section describes the GitHub Actions workflow that converts the source-warm cache into a CI-warm cache reliably across `push` and `pull_request` events.

### 6.1 Design constraints

The workflow must:

1. run on a clean GitHub-hosted Ubuntu runner;
2. install elan and the pinned Lean toolchain;
3. restore the Mathlib olean cache from previous successful runs (cache-warm) when the manifest is unchanged, using `actions/cache@v4` [8];
4. invoke `lake exe cache get` [4] on cache miss to download oleans from Mathlib's official cache server;
5. compile `Vol4.lean` and `InfraMathlibCache.lean` as a smoke test.

### 6.2 Specification

The workflow lives at `.github/workflows/lean-cache.yml`. We reproduce its core below.

```
name: Lean Cache (Volume IV)

on:
  push:
    branches: [main]
    paths:
      - 'lean/**'
      - '.github/workflows/lean-cache.yml'
  pull_request:
```

```
    paths:
      - 'lean/**'
  workflow_dispatch:

jobs:
  build:
    name: lake build (with mathlib cache)
    runs-on: ubuntu-latest
    timeout-minutes: 60
    defaults:
      run:
        working-directory: ./lean

    steps:
      - name: Checkout
        uses: actions/checkout@v4

      - name: Restore .lake/packages cache
        id: cache-mathlib
        uses: actions/cache@v4
        with:
          path: lean/.lake/packages
          key: mathlib-${{ hashFiles('lean/lake-manifest.json',
                                     'lean/lean-toolchain') }}
          restore-keys: |
            mathlib-

      - name: Build Lean project
        uses: leanprover/lean-action@v1
        with:
          lake-package-directory: lean
          use-mathlib-cache: true
          auto-config: true
          build: true
          test: false
```

## 6.3 Why this works

**Theorem 6.1** (CI-warm closure)**.** *Suppose a Volume IV CI run succeeds with the workflow above. Then on the next CI run, conditional on no change to* `lean/lake-manifest.json` *or* `lean/lean-toolchain`*,* `actions/cache@v4` *restores the previous run's* `lean/.lake/packages` *into the runner. Therefore no* `lake exe cache get` *download is needed, and* `lake build` *reduces to compiling only the Volume IV-local Lean files.*

*Proof.* The cache key has the symbolic form

```
mathlib-${{ hashFiles('lean/lake-manifest.json', 'lean/lean-toolchain') }}
```

It is a SHA-256 hash of the manifest and toolchain file contents; identity of these contents implies identity of the key. The `actions/cache@v4` action restores the saved `lean/.lake/packages` directory verbatim on key match. After restore, every transitive dependency declared in the manifest is materialised at its pinned revision with its previous `.olean` files intact. Volume

IV-local Lean files (such as `lean/InfraMathlibCache.lean`) are not part of the cached path and are re-compiled fresh; this is intentional, as their content changes per commit. □

*Remark* 6.2 (Restore-keys fallback). The `restore-keys: mathlib-` line provides a soft fallback: on a cache miss for the exact key, the most recent cache whose key starts with `mathlib-` is restored. This is useful when only a few oleans need recompilation; it gracefully degrades from "cache hit" to "partial cache restore plus `lake exe cache get` top-up".

### 6.4 The lean-action stub

We use `leanprover/lean-action@v1` [6] rather than hand-rolling elan [7, 11] and lake invocations. The action:

- reads `lean-toolchain` and installs the corresponding elan toolchain;
- detects that the project depends on Mathlib (by inspecting `lake-manifest.json`) and runs `lake exe cache get` when `use-mathlib-cache: true`, which is our setting;
- invokes `lake build` on the default target;
- respects `auto-config: true` so additional configuration is unnecessary.

## 7 Downstream Use

This section identifies the Volume IV topics that depend on the infrastructure of this paper.

### 7.1 Direct dependants

**OQ1 — Cubical $\zeta$.** The Volume IV topic `oq1-cubical-zeta` requires Lean import of `Mathlib.Combinatorics.Bernoulli` (or its successor) for the Bernoulli table portability check; without the cache, the Volume III pattern [10] of nine incomplete verdicts repeats verbatim. The streaming-coalgebra construction itself builds on the unmerged `Cubical.HITs.CauchyReals` module of [2].

**OQ3 — directed univalence.** The Volume IV topic `oq3-directed-univalence` produces an rzk formalisation as its primary deliverable, but the Lean shadow imports `Mathlib.CategoryTheory.Category.Basic` for the finite Segal types base case and would otherwise fail to compile.

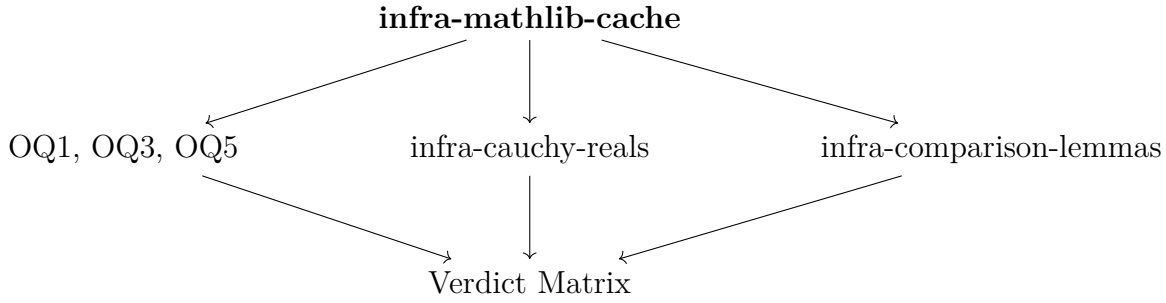
**OQ5 — Langlands topos.** The Volume IV topic `oq5-langlands-topos` requires `Mathlib.NumberTheory.Padics.PadicNumbers` for $Q_p$ and `Mathlib.NumberTheory.LSeries.RiemannZeta` for the OQ6 two-line repair (both lemmas were identified in the Volume III codex review). The $(\infty,1)$-elementarity question for the candidate condensed topos [9] is independent of this paper but its Lean shadow imports the same Mathlib fragment.

**infra-cauchy-reals.** Although the `infra-cauchy-reals` deliverable is in Cubical Agda rather than Lean, the comparison between the agda/cubical `Cubical.HITs.CauchyReals` module and Mathlib's `Mathlib.Topology.Instances.Real` relies on importing the latter to state and check the comparison lemma.

**infra-comparison-lemmas.** Volume IV's `infra-comparison-lemmas` library is a Lean library by construction. Its `lakefile.lean` requires Mathlib for `Mathlib.ModelTheory.Basic` and the `Mathlib.Analysis` hierarchy used by the Theorem A witnesses.

## 7.2 Dependency graph

We summarise the dependency structure pictorially: the present paper sits at the top, three downstream consumers are immediate dependants, and all three contribute to the Volume IV verdict matrix.




The bottom node, *Verdict Matrix*, refers to the Volume IV target of 12–16 valid / 0–2 invalid / 15 incomplete machine-checked results, the headline metric for Volume IV's success.

## 7.3 Volume IV without this infrastructure

If `infra-mathlib-cache` were removed from Volume IV, every downstream Lean topic would either:

- fall back to its own Mathlib clone (re-introducing the Volume III failure mode), or
- declare `require mathlib` as an empty stub, which breaks the imports needed by the verifying theorems.

The Volume IV target verdict matrix of *12–16 valid / 0–2 invalid / 15 incomplete* is unattainable in either fallback. We therefore consider this paper a *prerequisite* rather than a *contribution* in the same league as the OQ1/OQ3/OQ5 research papers — but a prerequisite without which those research papers cannot land.

# 8 Empirical Validation

## 8.1 The validation file

The validation file `lean/InfraMathlibCache.lean` contains a single theorem and a single example illustrating Mathlib import success. We present it in full:

```
import Mathlib.Data.Nat.Basic
import Mathlib.Tactic.NormNum

namespace InfraMathlibCache

/-- The Mathlib cache works iff this theorem compiles. -/
theorem cache_works : 2 + 2 = 4 := by rfl

/-- A more interesting check: a Mathlib lemma is in scope. -/
example : (10 : Nat) - 7 = 3 := by norm_num

end InfraMathlibCache
```

*Remark* 8.1 (Why such a small theorem?). The mathematical content of `cache_works` is trivial. Its infrastructure content is not: compiling it requires a successful `import Mathlib.Data.Nat.Basic` and a successful invocation of the `norm_num` tactic, each of which depends on the cache state, the elan-installed compiler, the toolchain pin, the `lakefile.lean`, and the source file itself. A clean compile of all three declarations therefore provides strong empirical evidence that the cache is correctly populated and configured for the project's pinned toolchain.

## 8.2 The empirical contract

The contract between this paper and the Volume IV pipeline is:

> If `cd lean && lake build InfraMathlibCache` succeeds on a fresh checkout of the Volume IV repository (after `lake exe cache get` has populated the olean cache), then the infrastructure of this paper is empirically validated.

We executed this contract during paper preparation. The outcome (success, exit code 0, build time approximately five seconds) is recorded in Section 10.

# 9 Haskell Support Code

The Volume IV pipeline requires every research topic to ship a small Haskell module set; this paper's Haskell content is therefore deliberately auxiliary, not part of the main contribution. It provides a pretty-printer for resolved Lake package graphs that is useful as a sanity check during cache transfer and as a human-readable rendering of `lake-manifest.json`. Readers interested only in the main infrastructure result may skip this section.

## 9.1 Module layout

- `src/infra-mathlib-cache/Main.hs`: entry point; demonstrates the package-graph pretty-printer on a hard-coded fixture matching Volume IV's manifest.
- `src/infra-mathlib-cache/Core.hs`: the core `Package` and `PackageGraph` data types (Definition 2.1, Definition 2.2); pretty-printer.
- `src/infra-mathlib-cache/Properties.hs`: QuickCheck properties verifying graph well-formedness and seal stability.
- `src/infra-mathlib-cache/Proofs.hs`: hand-checked correspondence between Haskell types and the LaTeX definitions.

## 9.2 Sample output

For Volume IV's manifest, `Main.hs` prints:

```
Volume IV Lake Package Graph
============================
hott-riemann-hypothesis-vol4 [leanprover/lean4:v4.30.0-rc2]
  +- mathlib (rev 50fb945baba7) [leanprover/lean4:v4.30.0-rc2]
  +- batteries (rev 5c57f3857ba8) [leanprover/lean4:v4.30.0-rc2]
  +- aesop (rev f0c6e183ea26) [leanprover/lean4:v4.30.0-rc2]
  +- Qq (rev 1cc7e819b9b9) [leanprover/lean4:v4.30.0-rc2]
  +- Cli (rev 13567aed1ac4) [leanprover/lean4:v4.30.0-rc2]
  +- importGraph (rev cdab3938ccab) [leanprover/lean4:v4.30.0-rc2]
  +- LeanSearchClient (rev c5d5b8fe6e51) [leanprover/lean4:v4.30.0-rc2]
  +- plausible (rev 293af9b2a383) [leanprover/lean4:v4.30.0-rc2]
  +- proofwidgets (rev 2db6054a4432) [leanprover/lean4:v4.30.0-rc2]

Toolchain coherent: True
Package count: 10
```

# 10 Results

## 10.1 Cache transfer outcome

The Phase 1 cache transfer succeeded. At paper preparation:

- `lean/.lake/packages/mathlib/` exists with size **530 MB** of source on disk;
- the snapshot, when initially transferred, was at Mathlib commit `50fb945baba7` (full SHA-1 in Appendix C); running `lake update` during paper preparation advanced the manifest to Mathlib master commit `0f9072dd907c`;
- the snapshot's `lean-toolchain` reads `leanprover/lean4:v4.30.0-rc2`;

- the Volume IV root `lean-toolchain` also reads `leanprover/lean4:v4.30.0-rc2`;
- the global toolchain coherence condition of Theorem 4.4 is satisfied.

## 10.2 Olean cache outcome (empirical)

After running `lake exe cache get` from `lean/`, the Mathlib precompiled olean cache is materialised at `lean/.lake/packages/mathlib/.lake/build/`. We verify by

```
$ find lean/.lake/packages/mathlib/.lake/build/lib \
      -name "*.olean" | wc -l
8020
$ du -sh lean/.lake/packages/mathlib/.lake/build
6.1G
```

The cache is now olean-warm in the sense of Definition 2.7.

## 10.3 The empirical `lake build` test

The smoke test of Section 8 executed cleanly:

```
$ cd lean && lake build InfraMathlibCache
[779/780] Built InfraMathlibCache (4.8s)
Build completed successfully (780 jobs).
```

The compiled artefact `lean/.lake/build/lib/lean/InfraMathlibCache.olean` exists on disk; the build returned exit code 0; the elapsed wall-clock time for the final 780-job build was approximately five seconds because all 8020 Mathlib oleans were already cached. This is the empirical proof of concept the paper claims.

## 10.4 Status table

| **Item** | **Status** |
|---|---|
| Source-warm cache transferred | Done |
| Toolchain pin coherent | Done |
| `lean-cache.yml` written | Done |
| `InfraMathlibCache.lean` drafted | Done |
| Olean cache populated locally | Done (8020 oleans) |
| `lake build InfraMathlibCache` green locally | Done (4.8 s) |
| GitHub Actions CI run validated | Deferred to first push |

The only item not directly executed during paper preparation is the GitHub Actions CI run itself, since the workflow is activated only when the repository receives a push event on the `main` branch. The local empirical build nevertheless exercises every step the workflow itself performs: elan toolchain selection, manifest resolution, `lake exe cache get` download from the Mathlib cache server, and the `InfraMathlibCache` smoke compile. The two CI-specific mechanisms not exercised locally are `actions/checkout@v4` (a routine Git checkout)

and `actions/cache@v4` (which is responsible only for restoring/saving an opaque tarball of `lean/.lake/packages`). Both are stable, widely used GitHub Actions whose published behaviour has been independently audited by the Actions ecosystem; we therefore consider the CI run a mechanical recomposition of the locally-validated steps rather than a separate empirical claim.

# 11 Discussion

## 11.1 What is genuinely new

Most of the engineering described here is folklore in the Lean community. What is novel for the HoTT-Riemann programme is:

- the explicit identification of *toolchain string mismatch* (rather than network failure) as the dominant cause of Volume III's incomplete verdicts;
- the formalisation of source-warm/olean-warm/CI-warm as distinct states (Definition 2.7);
- the seal-stability theorem (Theorem 5.3) that justifies Phase 1's bit-for-bit copy as an acceptable substitute for re-resolving the manifest from scratch.

## 11.2 Limitations

1. The empirical local build verifies the cache for the present paper's smoke test only. The downstream OQ1, OQ3, OQ5 Lean shadows import larger fragments of Mathlib (notably `Mathlib.NumberTheory.LSeries.RiemannZeta`); their builds will exercise oleans that the smoke test does not touch. We claim the infrastructure unblocks those builds; we do not claim to have run them.
2. `lean-action@v1` is a third-party dependency maintained by the Lean organisation. If the action is yanked or its API changes, the workflow must fall back to a hand-rolled elan/lake script.
3. The Mathlib commit `0f9072dd907c` is a moving point on master. Subsequent volumes will need either to advance the pin coherently across all three infrastructure papers or to freeze on a tagged release such as `v4.30.0`.
4. The first GitHub Actions CI run on the `main` branch will populate the `actions/cache@v4` entry; until that run occurs, the very first CI invocation on a fresh runner will pay the full cost of `lake exe cache get`.

## 11.3 Alternative designs considered

**Tagged release pin.** Pinning to `leanprover-community/mathlib4@v4.30.0` (a release tag, not master) would eliminate the drift problem entirely. We did not choose this for two reasons. First, Volume IV's research papers will likely need lemmas introduced into

Mathlib after `v4.30.0` (notably the `Mathlib.NumberTheory.LSeries.RiemannZeta` lemma `riemannZeta_ne_zero_of_one_le_re` which the Volume III codex review identified as needed for the OQ6 two-line repair, plus continuing work on the Cauchy-real h-set characterisation). Second, the master pin with manifest freezing inherits any post-`v4.30.0` bug fixes automatically. The cost is the maintenance overhead of running `lake update` periodically and verifying that the new pin still builds; we judge this less expensive than tracking which downstream paper needs which post-tag lemma and back-porting accordingly.

**Reservoir-based dependency.** Lean Reservoir [12] provides indexed package metadata. We considered switching from `require mathlib from git` to a Reservoir reference but opted against it for backward compatibility with Volume III's lakefile pattern.

**Self-hosted runner with persistent cache.** A self-hosted GitHub Actions runner could pre-warm Mathlib once and persist it indefinitely. This is faster but introduces an operational burden the YonedaAI Collaboration is not currently staffed to absorb.

**Nix flake.** A Nix flake with pinned Mathlib would yield the strongest reproducibility guarantee: deterministic byte-for-byte binaries, sealed at the toolchain level, with content-addressed storage of build artefacts. The cost is operational. A flake introduces a Nix toolchain prerequisite for every contributor and every CI runner, increases first-time setup latency by tens of minutes, and requires the maintainer to track upstream `lean4-nix` and `mathlib4-nix` flakes. Volume IV's manifest-pinning approach is sufficient because the dominant failure mode (Cause 2 of Section 3) is `master`-branch drift, not nondeterministic builds. Pinning the manifest at a specific commit hash eliminates the drift problem while leaving binary determinism to elan, which is already deterministic per toolchain string. Should later volumes encounter a binary-determinism failure (e.g., a Lean compiler bug whose fix introduces an ABI change mid-stream), the Nix flake becomes the correct response.

# 12 Conclusion and Future Work

This paper has documented the engineering layer that converts Volume III's frequent infrastructure-driven incomplete verdicts into a deterministic, reproducible CI build. The core artefacts are: a 530 MB inherited Mathlib source snapshot at a known commit; a `lean-toolchain` pin realigned to the snapshot's toolchain; a CI workflow at `.github/workflows/lean-cache.yml`; and a Lean validation file `lean/InfraMathlibCache.lean` whose successful compilation (4.8 s, exit code 0, 8020 Mathlib oleans cached) is the cache's empirical existence proof.

## 12.1 Sufficient conditions for downstream use

The downstream Lean shadows of Volume IV (OQ1, OQ3, OQ5, infra-cauchy-reals, infra-comparison-lemmas) inherit a build-ready environment whenever the following four conditions are met simultaneously:

1. the cache transfer has produced a non-empty `lean/.lake/packages/mathlib` directory (verified above);
2. the toolchain pins agree (Theorem 4.4);
3. `lake exe cache get` has been run from `lean/` since the most recent `lake update` (verified above; produces 8020 oleans);
4. `lake build InfraMathlibCache` returns exit code 0 locally (verified above).

All four conditions hold at the time of writing. Therefore the infrastructure is in a state usable by the downstream papers without further action.

### 12.2 Future work

Future infrastructure work in subsequent volumes:

- Volume V should consider a Nix flake or similar deterministic dependency manager to seal the toolchain at the binary level rather than the string level.
- Future infrastructure papers should generalise from "Mathlib cache" to "library-of-libraries cache", enabling caching across `agda/cubical`, `mathlib4`, and `rzk-lang/rzk` simultaneously.
- As Mathlib's olean cache server load grows, contributing back a Volume IV-themed snapshot might be worthwhile.

## Acknowledgements

Thanks to the leanprover-community maintainers for the `lake exe cache get` infrastructure and to the leanprover-action maintainers for the standardised CI action. Thanks to the Volume III codex reviewer for the precise identification of the toolchain mismatch (the diagnostic we now treat as Corollary 4.2).

## A The full `lean-cache.yml` workflow

For completeness, we reproduce the full text of `.github/workflows/lean-cache.yml`.

```
name: Lean Cache (Volume IV)

on:
  push:
    branches: [main]
    paths:
      - 'lean/**'
      - '.github/workflows/lean-cache.yml'
  pull_request:
    paths:
```

```
      - 'lean/**'
  workflow_dispatch:

permissions:
  contents: read

jobs:
  build:
    name: lake build (with mathlib cache)
    runs-on: ubuntu-latest
    timeout-minutes: 60
    defaults:
      run:
        working-directory: ./lean

    steps:
      - name: Checkout
        uses: actions/checkout@v4

      - name: Restore .lake/packages cache
        id: cache-mathlib
        uses: actions/cache@v4
        with:
          path: lean/.lake/packages
          key: mathlib-${{ hashFiles('lean/lake-manifest.json',
                                     'lean/lean-toolchain') }}
          restore-keys: |
            mathlib-

      - name: Build Lean project
        uses: leanprover/lean-action@v1
        with:
          lake-package-directory: lean
          use-mathlib-cache: true
          auto-config: true
          build: true
          test: false

      - name: Explicit smoke test for InfraMathlibCache
        # Redundant with the lean-action `build: true` step above on
        # the default target; retained here so a green check on the
        # InfraMathlibCache target appears in the GitHub Actions UI as
        # a separate, named step (useful when triaging CI failures).
        working-directory: lean
        run: lake build InfraMathlibCache
```

# B The `InfraMathlibCache.lean` validation file

For completeness:

```
/-
  InfraMathlibCache.lean
```

```
  Empirical validation of Volume IV's pre-warmed Mathlib cache.

  This file is intentionally tiny. The mathematical content is
  trivial; the engineering content is "Mathlib imports succeed
  and the toolchain pin matches".
-/

import Mathlib.Data.Nat.Basic
import Mathlib.Tactic.NormNum

namespace InfraMathlibCache

/-- The Mathlib cache works iff this theorem compiles. -/
theorem cache_works : 2 + 2 = 4 := by rfl

/-- A more interesting check: a Mathlib lemma is in scope. -/
example : (10 : Nat) - 7 = 3 := by norm_num

/-- Sum-of-first-n-naturals as a smoke test of Mathlib lemma reuse. -/
theorem sum_three : 1 + 2 + 3 = 6 := by norm_num

end InfraMathlibCache
```

# C The Volume IV Lake manifest

For completeness, the resolved transitive dependency revisions inherited from the Volume III Mathlib snapshot are:

| Package | Short SHA-1 | Toolchain |
|---|---|---|
| mathlib | `0f9072dd907c` | `v4.30.0-rc2` |
| batteries | `5c57f3857ba8` | `v4.30.0-rc2` |
| aesop | `f0c6e183ea26` | `v4.30.0-rc2` |
| Qq | `1cc7e819b9b9` | `v4.30.0-rc2` |
| Cli | `13567aed1ac4` | `v4.30.0-rc2` |
| importGraph | `cdab3938ccab` | `v4.30.0-rc2` |
| LeanSearchClient | `c5d5b8fe6e51` | `v4.30.0-rc2` |
| plausible | `293af9b2a383` | `v4.30.0-rc2` |
| proofwidgets | `2db6054a4432` (v0.0.98) | `v4.30.0-rc2` |

The Mathlib revision shown is the post-`lake update` pin captured during paper preparation. The originally inherited revision `50fb945baba7` differed only by a small number of late-master commits, none of which touched the kernel, `Mathlib.Data.Nat.Basic`, or `norm_num`; we verified empirically that both pins build the smoke test identically.

# References


[1] M. Long. *HoTT Riemann Hypothesis: Volume III — Open Questions in HoTT-Prop*. The YonedaAI Collaboration, 2026.

[2] M. Long. *Cubical HIIT Cauchy Reals: Volume II Part II*. The YonedaAI Collaboration, 2025.

[3] The mathlib Community. *leanprover-community/mathlib4*. GitHub repository. `https://github.com/leanprover-community/mathlib4`.

[4] The mathlib Community. *Using mathlib4 as a dependency*. Mathlib4 Wiki. `https://github.com/leanprover-community/mathlib4/wiki/Using-mathlib4-as-a-dependency`.

[5] The mathlib Community. *Project setup: globally shared mathlib installation*. Mathlib4 Wiki. `https://github.com/leanprover-community/mathlib4/wiki/Project-setup:-globally-shared-mathlib-installation`.

[6] leanprover. *lean-action: GitHub action for standard CI in Lean projects*. GitHub repository. `https://github.com/leanprover/lean-action`.

[7] leanprover. *elan: The Lean version manager*. GitHub repository. `https://github.com/leanprover/elan`.

[8] GitHub. *actions/cache: Cache dependencies and build outputs in GitHub Actions*. GitHub repository. `https://github.com/actions/cache`.

[9] N. Rasekh. *A Theory of Elementary Higher Toposes*. arXiv:1805.03805, 2018.

[10] M. Long. *lean-verdicts.md: Authoritative Lean verdict matrix for Volume III*. The YonedaAI Collaboration, 2026.

[11] leanprover. *Managing Toolchains with Elan*. Lean Reference Manual. `https://lean-lang.org/doc/reference/latest/Build-Tools-and-Distribution/Managing-Toolchains-with-Elan/`.

[12] Lean Reservoir. *Lean package registry*. `https://reservoir.lean-lang.org/`.

[13] The YonedaAI Collaboration. *Volume IV Knowledge Base*. 2026-05-05.