The Axiom Profile of Mathlib

Larsen James Close

2026

Abstract

Every theorem in a proof library rests on a measurable set of axioms, and a library's distribution of those sets — its axiom profile — is a fact about the formalized library, not an opinion about it. This paper measures the profile of the entire Mathlib library: 707,053 declarations at a pinned commit, each declaration's transitive axiom closure computed from a content-addressed archive of the elaborated terms, certified against the Lean kernel by full replay. The findings: 25.09% of Mathlib (177,395 declarations) depends on no axioms at all; only 23 distinct axiom profiles occur in the whole corpus, and 99.96% of declarations lie on the Boolean cube over three axioms — propositional extensionality, quotient soundness, and choice; the library contains zero incomplete proofs. Choice reaches half the corpus (50.84%), but its entry structure is extraordinarily narrow: every choice-dependent declaration reaches the axiom through a frontier of just 206 declarations that use it directly, and a structural counterfactual over the dependency graph shows 55.4% of all choice-dependence flowing through three generic classical-instance gateways jointly. The measurement separates two things the per-declaration view conflates: mathematics that needs choice, and mathematics that inherits choice from shared infrastructure. Every figure is recomputable from the query outputs deposited with this paper.

What an axiom profile is

A proof checked by a kernel is not unconditionally true; it is true relative to what it assumes. In Lean 4 (de Moura and Ullrich 2021) the assumptions are explicit objects — declarations introduced by the `axiom` keyword — and every theorem's dependence on them is a computable fact: walk the proof term, collect every constant it references, close transitively, and keep the axioms. Lean ships this facility per declaration as `#print axioms`. The set that comes back is the declaration's **axiom profile**: the exact, minimal description of the axioms the kernel was asked to take on faith.[1]

The profile is the legibility layer of formal mathematics. Two proofs of the same statement are not interchangeable if one consumes `Classical.choice` and the other does not — they license different uses, transfer to different foundations, and extract to different computational content. A library's profile distribution is therefore a structural fact about the formalized library, in the same way a codebase's dependency graph is a structural fact about the software. An axiom profile is a property of a formalization, not of a theorem in the abstract: the same result can carry a different profile under a different encoding or foundation, and that encoding-dependence is itself part of what the measurement exposes. What has been missing is the measurement at corpus scale: not *this theorem's* assumptions but *the whole library's* — the distribution of profiles, the reach of each axiom, and the topology of how the non-constructive principles actually enter.

This paper reports that measurement for Mathlib (the mathlib Community 2020), the largest unified library of formalized mathematics in existence: 707,053 declarations across 9,597 modules, at commit `8f9d9cff6bd728b17a24e163c9402775d9e6a365` on toolchain `leanprover/lean4:v4.28.0`. Three findings organize the results.

First, the profile space is tiny. A corpus of seven hundred thousand declarations could in principle exhibit thousands of distinct axiom combinations. Mathlib exhibits 23 — and 99.96% of the corpus lies on the Boolean cube over just three axioms: `propext` (propositional extensionality), `Quot.sound` (quotient soundness), and `Classical.choice`. All eight subsets of the three occur. The remainder is a thin compiler band plus the axiom `sorryAx` reaching exactly one declaration — itself. Mathlib contains zero incomplete proofs.

Second, a quarter of Mathlib is axiom-free. 177,395 declarations — 25.09% — depend on no axioms at all: fully constructive in the strictest sense the kernel can certify, computing under no assumptions whatsoever.

Third, choice is wide but narrow. Half the corpus (50.84%) reaches `Classical.choice` transitively — yet only 206 declarations in all of Mathlib reference the axiom directly, and a counterfactual analysis over the dependency graph shows the majority of all choice-dependence flowing through three generic classical-instance gateways. For most of the choice-dependent corpus, the axiom enters through shared infrastructure rather than through direct, local reference at the site itself.

The measurement substrate

A corpus-scale claim is only as good as the chain of custody behind it, so the substrate is stated before the results. Computing transitive axiom closure correctly at this scale is not automatic — a dependency graph that mis-traces how an axiom enters through shared infrastructure yields a materially different distribution — so the extraction below is certified against the kernel and cross-checked across independent implementations, not trusted to a single computation.

[1] The kernel also grants commitments it never turns into a proof obligation — definitional eta, proof irrelevance, and the conversion rules by which it judges two terms equal. These are substantive (coherent type theories omit them) yet appear in no axiom profile, because the profile tracks dependencies and a definitional grant is never depended upon. That stratum — and the assumptions implicit in how a domain is encoded, which become definitions or theorems rather than axioms — is a distinct measurement, taken up in separate work.

The measurement does not run over source text or over a live elaborator session. It runs over a **content-addressed archive** of the fully elaborated corpus: every declaration's type and value terms stored as hash-consed nodes in a frozen binary grammar, 180.2 million unique nodes for the full library, with per-module manifests recording exactly which declarations each module contributes. The store's identity is a single content hash; the figures below are deterministic functions of that hash and the manifest set, in the sense that re-running any query against the same store produces byte-identical reports.

The archive was certified faithful against the Lean kernel three ways before any query ran:

- **Kernel replay.** Every declaration — all 707,053 — was decoded from the stored bytes alone and re-submitted to the kernel, which accepted it. The stored terms are not a transcription of the corpus; they are the corpus, as the kernel sees it.
- **Exact reconstruction.** Every declaration re-emitted from the store matches the original elaborated form byte-exactly, down to binder names.
- **Block re-entry.** Every inductive block re-enters the kernel from the store and the regenerated recursors match the stored ones.

The dependency and axiom-closure views — per-declaration direct references and per-declaration transitive axiom closure — were then derived from the store and certified by **three independent implementations agreeing**: a Python walk over the stored bytes; a Lean walk over the live kernel environment, byte-identical at a 167,447-declaration sample; and a third, name-graph recomputation that reproduces the axiom view from the dependency view exactly, on all 707,053 rows. The headline aggregations below are each additionally cross-derived by an independent second pass. Nothing reported here depends on trusting a single program.

The axiom profile

177,395 of 707,053 declarations — 25.09% of Mathlib — depend on no axioms at all.

There are only **23 distinct axiom profiles** in the entire corpus. Eight are the subsets of {`propext`, `Quot.sound`, `Classical.choice`}, and all eight occur. Together those eight cover 706,794 declarations: **99.96% of Mathlib lives on the Boolean cube over three axioms.** The remaining 259 declarations are a compiler band — `lcProof` and relatives, `Lean.trustCompiler`, `Lean.ofReduceBool`/`ofReduceNat`, fourteen profiles in all — plus `sorryAx` reaching exactly one declaration: itself. **Mathlib contains zero sorried declarations.**

The full distribution:

axiom profile	declarations	share
{Classical.choice, Quot.sound, propext}	355,361	50.26%
∅ (axiom-free)	177,395	25.09%
{Quot.sound, propext}	82,331	11.64%
{propext}	78,896	11.16%
{Quot.sound}	8,744	1.24%
{Classical.choice}	2,590	0.37%
{Classical.choice, propext}	1,208	0.17%
{Classical.choice, Quot.sound}	269	0.04%
compiler band (14 profiles)	258	0.04%
{sorryAx} (the axiom's own row)	1	—

And the per-axiom reach — declarations whose transitive closure contains the axiom:

axiom	reach	share of corpus
`propext`	517,880	73.24%
`Quot.sound`	446,774	63.19%
`Classical.choice`	359,470	50.84%
`lcProof`	247	0.03%
`Lean.trustCompiler`	5	—
each of `ofReduceBool`, `ofReduceNat`, `Quot.lcInv`, `lcAny`, `lcCast`, `lcErased`, `lcUnreachable`, `lcVoid`, `sorryAx`	1	—

Reach counts every declaration whose transitive closure contains the axiom, including the few compiler-band declarations off the cube, so each reach slightly exceeds the sum of its cube vertices — choice 359,470 against 359,428, propext 517,880 against 517,796, Quot.sound 446,774 against 446,705. The gaps are exactly the compiler-band declarations whose closures also carry a core axiom.

Two readings of this table deserve emphasis. The first is hygiene: a corpus built by thousands of contributors over a decade exhibits twenty-three assumption-sets, no stray domain axioms in general circulation, and not one incomplete proof. This is a remarkable fact about Mathlib's engineering discipline, and the measurement makes it checkable rather than reputational.

The second is structure: the three-axiom cube is not an arbitrary cutoff but the actual shape of the library. The fully classical vertex carries half the corpus (50.26%); the fully constructive vertex carries a quarter (25.09%); the remaining mass distributes over the partial combinations, dominated by the choice-free pairs. The constructive fraction of Mathlib is not a rounding error — it is the second-largest vertex of the cube.

Where choice enters

Half the corpus reaches `Classical.choice`. The per-declaration view stops there, and read naively it suggests that half of formalized mathematics needs the axiom of choice. The dependency graph supports a sharper question: *how* does the axiom reach all those declarations?

Only 206 declarations in all of Mathlib reference `Classical.choice` directly. Every one of the 359,470 choice-dependent declarations reaches the axiom through this frontier. The frontier is the complete set of entry points; everything else is inheritance.

The frontier's internal structure is measured by a **knockout**: a structural counterfactual on the dependency graph. Knocking out a declaration h means recomputing every axiom closure with h's outgoing edges cut — the graph model of "h reproven without its current assumptions" — and reporting the freed set: declarations choice-dependent at baseline whose recomputed closure is choice-free.

The semantics must be stated precisely, because the numbers are easy to over-read. A knockout measures how much dependence *flows through* a declaration — infrastructure-mediated inheritance. It does **not** say the declaration is replaceable: `Classical.propDecidable`, the instance making every proposition classically decidable, is not provable without choice, and no rewrite will make it so. What the freed count says is how many declarations reach choice *only* through it.

Measured:

- `Classical.propDecidable` has 8,049 direct dependents. Knocking it out frees **91,188 declarations — 25.4% of the entire choice-dependent set.**

- The top-three joint knockout — `propDecidable`, `Classical.indefiniteDescription`, `Classical.ofNonempty` — frees **199,160 declarations: 55.4% of all choice-dependence in Mathlib.**
- The joint effect is strongly superadditive: 199,160 jointly, versus 97,477 as the sum of the three individual knockouts. Dependence flows through the gateway triple jointly — cutting one reroutes through the others.

The reading: for the majority of the choice-dependent corpus, the axiom enters through generic classical instances — decidability supplied classically because it is convenient and uniform — rather than through direct, local reference to choice at the site of the theorem. The 206-declaration frontier, and within it the three-gateway structure, is the actual interface between Mathlib and the axiom of choice. A library that wanted to offer constructive variants would not face 359,470 declarations of work; it would face a frontier whose dominant members are already enumerated, with the impact of each candidate priced in advance by exactly this arithmetic.

The graph the measurement runs on

The dependency structure underneath these results is of independent interest, and three of its facts bear directly on how a corpus like this can be maintained and re-verified.

The name-level graph has 707,053 declarations and **18,419,389 direct-dependency edges** (mean out-degree 26.1, median 17; median in-degree 1). `Eq` — propositional equality itself — is referenced directly by 430,768 declarations, 61% of the corpus, and its reverse dependency cone covers **617,711 declarations, 87.4% of Mathlib**: the maximal blast radius any change to a foundational type could have.

The reverse-cone distribution is extremely heavy-tailed. In a deterministic structured sample of the corpus, **half of all declarations are used, transitively, by at most 11 others**, while the foundational hubs are used by hundreds of thousands. For maintaining a moving corpus the consequence is concrete: a change to a typical declaration invalidates almost nothing and is nearly free to re-verify, while the expensive tail is small and exactly enumerable in advance. Verification cost under drift is not uniform — it is priced per declaration by the graph, before any work is spent.

The longest dependency chain in Mathlib has depth 338, its summit in the completely-positive-maps layer of the C*-algebra tower; the mean depth is 52.2. And the module quotient of the graph — 9,597 modules, 474,559 cross-module edge classes — contains exactly **one cycle**, which turns out to be a finding rather than an error: Lean realizes certain auxiliary declarations lazily (equation lemmas, match auxiliaries, private realizations), and a realized auxiliary lands in whichever module happens to force it first. Two import-incomparable modules can therefore each hold realizations of a shared upstream definition that reference each other — here, `Nat.factorial`'s realizations, split across `Mathlib.Data.List.Permutation` and `Mathlib.Data.Nat.Factorial.DoubleFactorial`. Module membership records where a declaration's winning realization landed, not a reference-locality boundary; tooling that assumes module-level acyclicity of declaration references will, at exactly one place in current Mathlib, be wrong.

Reproduction

Every figure above is recomputable from the query outputs deposited with this paper. The measurement is pinned to Mathlib commit `8f9d9cff6bd728b17a24e163c9402775d9e6a365` on toolchain `leanprover/lean4:v4.28.0`. The outputs are produced by extracting every declaration's elaborated terms into a content-addressed store — certified against the Lean kernel as described

above — and running three query passes over the derived views: the axiom profile, the choice-inheritance topology, and the dependency structure. The store is identified by the content hash `bbc45318d876958ae2e7a21f12c1b33ca78b5cb621b40849f4b5d7b632cc8f26` over 180,211,645 nodes, and the deposited outputs are a deterministic function of it.

Availability

This paper and the certified query outputs its figures derive from are deposited together at Zenodo: **https://doi.org/10.5281/zenodo.20710921**. The deposit contains the per-declaration axiom-skeleton, choice-inheritance, and dependency-structure outputs, a record of the Mathlib commit, toolchain, and store identity that produced them, and a script that recomputes every headline figure in this paper from those outputs.

References

de Moura, Leonardo, and Sebastian Ullrich. 2021. "The Lean 4 Theorem Prover and Programming Language." In *Automated Deduction — CADE 28*, Lecture Notes in Computer Science 12699, 625–635. Springer. https://doi.org/10.1007/978-3-030-79876-5_37

The mathlib Community. 2020. "The Lean Mathematical Library." In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs (CPP 2020)*, 367–381. ACM. https://doi.org/10.1145/3372885.3373824