Is Math Curvy? Is Math Fat?

The Intrinsic Geometry of the Lean 4 Mathlib Library

Ashani Dasgupta
Cheenta Academy
`ashani.dasgupta@cheenta.com`

March 2026

Abstract

I have always been curious about the shape of mathematics. In this paper I construct the full dependency graph of the Lean 4 Mathlib library—619,518 declarations and 12.5 million logical dependencies—and ask two questions about its geometry. Is it *curvy*? Is it *fat*?

The answer to both is yes, but at different scales. Globally the graph is tree-like: stochastic sampling of 10,000 quadruples yields a Gromov hyperbolicity constant $\delta = 1.50$ ($\delta = 1.00$ after pruning high-degree utility nodes). Locally it is dense: a Jaccard-overlap proxy for Ollivier–Ricci curvature measured on 20,000 edges returns a mean $\kappa \approx 0.022 > 0$. I call this dual architecture a *tree of cliques*.

Three null-model controls, Erdős–Rényi, Barabási–Albert, and a configuration model with Mathlib's exact degree sequenc; all exhibit *negative* local curvature ($\kappa \approx -0.01$ to -0.02) after identical pruning. The positive curvature of Mathlib is not a statistical artifact. It is a property of mathematical knowledge itself.

I embed the graph into a 16-dimensional Poincaré ball and use the geometry to hunt for logical voids. Two results have been formally verified in Lean 4: a scale-invariance lemma for τ-packings in the Besicovitch covering theorem, and a construction of the cycle matroid bridging graph theory and matroid theory—a foundational definition absent from the current Mathlib library.

1 Introduction

I have always been curious about the shape of mathematics. Usually a mathematician proves new theorems by gluing other previously known theorems. In this paper we imagine theorems, definitions and axioms as nodes in a graph. If theorem A is used directly in the proof of theorem B, we add an edge from A to B. This produces a directed graph of mathematics. For now we disregard the directions and study the properties of this graph $\mathcal{M}$.

There exists a database of theorems and formalized proofs called Mathlib, written in the Lean 4 proof assistant. We used this library to construct $\mathcal{M}$ with 619,518 nodes and 12,506,540 edges. Some vertices are attached to many, many other vertices. For example `Eq` (Lean's definition of equality) is connected to 273,547 other vertices.

To remove the noise created by these high-degree vertices, we prune the top 1% by degree. Call the new graph $\mathcal{M}'$. We then ask the following questions:

- **Global topology.** Is $\mathcal{M}'$ Gromov hyperbolic?
- **Local geometry.** What is the neighborhood shape of $\mathcal{M}'$?
- **Null models.** Is the geometry special to mathematics, or would any large graph look the same?
- **Embedding.** Can we map $\mathcal{M}'$ into a Poincaré ball?
- **Synthesis.** Can we extrapolate the manifold to predict new theorems?

2 Gromov Hyperbolicity

We wish to know whether $\mathcal{M}'$ is Gromov hyperbolic. The Gromov product of x, y with respect to w is

$$(x, y)_w := \tfrac{1}{2}\big(d(x, w) + d(y, w) - d(x, y)\big).$$

$\mathcal{M}'$ is δ-hyperbolic if for all x, y, z, w,

$$(x, y)_w \geq \min\{(x, z)_w,\ (y, z)_w\} - \delta.$$

The smallest such δ is the Gromov hyperbolicity constant. Trees have $\delta = 0$. It would be interesting if we find a small δ for $\mathcal{M}'$.

Computing δ exactly requires checking all $\binom{n}{4}$ quadruples, which for $n > 600{,}000$ would take millions of years. We estimate δ using 10,000 randomly chosen quadruples. For each quadruple x, y, z, w we compute the six pairwise distances, form the three sums $d(x, y) + d(z, w)$, $d(x, z) + d(y, w)$, $d(x, w) + d(y, z)$, sort them, and take $\delta = \frac{1}{2}(S_{\text{max}} - S_{\text{mid}})$.

The result is seems interesting. Over 10,000 quadruples, the maximum observed δ is **1.50** for the full graph and **1.00** for the pruned graph. For a library containing over twelve million edges, this is extraordinarily small. It tells us that the global structure of Mathlib is nearly as tree-like as a classical hyperbolic plane.

3 Local Curvature: Is Math Fat?

The Ollivier–Ricci curvature measures local fatness. Suppose x and y are adjacent vertices. We place masses on each vertex and its neighbors using a lazy random walk: half the mass stays at the vertex, the other half is distributed equally among its neighbors. Then we compute the Wasserstein distance W_1 between these two mass distributions. The curvature is

$$\kappa(x, y) = 1 - \frac{W_1(\mu_x, \mu_y)}{d(x, y)}.$$

In a tree, the neighborhoods are spread far apart, so W_1 is large and $\kappa < 0$. When shortcuts exist (triangles, shared neighbors), transport is cheaper and κ becomes positive.

Computing exact κ requires solving a linear program for each edge, which is too expensive at our scale. Instead we use a Jaccard-overlap proxy on 20,000 randomly sampled edges:

$$\hat{\kappa}(u, v) = \frac{|N(u) \cap N(v)|}{|N(u) \cup N(v)|} - \frac{1}{|N(u)| + |N(v)|}.$$

The mean is $\hat{\kappa} = 0.022$. This positive curvature tells us that locally, the graph is *fat*: adjacent theorems share many common neighbors, forming dense clusters within specific mathematical domains.

4 Tree of Cliques

So we have a duality. Globally, $\mathcal{M}'$ is hyperbolic ($\delta \leq 1.50$): the macro-structure follows the exponential expansion of a tree. Locally, the neighborhoods are positively curved ($\kappa > 0$): within any single mathematical domain, theorems are densely interlinked.

I describe this as a **tree of cliques**. Inside a topic like Linear Algebra or Group Theory, the logic is redundant and interconnected. The cost of moving between related ideas is very low. But between topics, these dense clusters are connected by thin, branching bridges. At the global scale, the shortest path between a theorem in Topology and a theorem in Number Theory must travel through a common logical ancestor.

This hybrid geometry suggests that the Poincaré ball, with its infinite room near the boundary and its crowded center, is a natural home for this structure.

Table 1: Geometric signatures of Mathlib versus three null models. All graphs pruned identically (top 1% by degree removed). Mathlib is the only graph with positive local curvature.

Graph	**Nodes (pruned)**	$\boldsymbol{\delta}$	**Mean** $\boldsymbol{\kappa}$
Mathlib	613,348	1.50	$+\mathbf{0.022}$
Erdős–Rényi	614,696	1.00	-0.012
Barabási–Albert	613,361	1.00	-0.013
Configuration model	613,336	1.00	-0.021

5 Is the Tree of Cliques Real?

A natural worry, raised by Johan Commelin on the Lean Zulip, is that our top-1% pruning might *force* the graph into a tree of cliques. Perhaps any large graph, pruned the same way, would look like this.

To test this, we generated three null-model graphs with the same basic statistics as Mathlib and applied identical pruning:

- **Erdős–Rényi**: 619,518 nodes and 12,506,540 edges placed uniformly at random. No hierarchy, no hubs, no community structure.
- **Barabási–Albert**: 619,518 nodes grown by preferential attachment ($m = 20$ edges per new node). This produces hubs and a power-law degree distribution, mimicking Mathlib's high-degree utility nodes.
- **Configuration model**: Mathlib's *exact* degree sequence, but with all edges rewired randomly. Same number of connections per node, but the mathematical content of those connections is destroyed.

The result (Table 1) is interesting. All three null models exhibit *negative* local curvature after identical pruning. Mathlib is the only graph with positive κ.

The configuration model is particularly telling. It preserves every node's degree—the same hubs, the same distribution of connections—but destroys which theorems depend on which. This alone flips κ from $+0.022$ to -0.021. The sign reverses; the magnitude is nearly identical.

The tree of cliques seems not a statistical artifact. It is not a consequence of having hubs. It is not created by pruning. It is a property of *which theorems depend on which other theorems.* It appears to me that this might be a property of mathematical knowledge itself.

6 Autoformalization Does Not Trim Fat

Yakov Pechersky (Lean Zulip, March 2026) made an interesting hypothesis: that local curvature κ might measure human API design choices rather than logical structure. Human mathematicians write lemmas that are dense locally because they optimize for readability and reuse. Autoformalized code, optimizing instead for proof completion, might produce a flatter, leaner graph.

The sphere packing formalization exists in two versions: a human-written corpus by `thefundamentaltheor3m` and an autoformalized version by Math Inc Gauss. We embedded both in the Poincaré ball and measured identically.

The result (Table 2): $\kappa = 0.0199$ (human) versus $\kappa = 0.0201$ (Gauss). The geometry did not change. In this example at least, autoformalization does not trim fat. The local curvature appears intrinsic to the mathematics, not to the formalization style.

Table 2: Human versus autoformalized sphere packing. The geometry is identical.

Metric	**Mathlib**	**Mathlib Pruned**	**SP Human**	**SP Gauss**
Nodes	619,518	∼613,323	381,073	408,660
Edges	12,506,540	2,424,185	7,122,578	7,683,420
δ	1.50	1.00	2.00	1.50
Mean κ	0.0217	0.0220	0.0199	0.0201

7 Hyperbolic Neural Network

We embed the graph into a 16-dimensional Poincaré ball using a hyperbolic neural network. Each node gets a trainable coordinate $\mathbf{e}_i \in \mathbb{B}^{16}$. The distance between two points is:

$$d_{\text{hyp}}(\mathbf{x}, \mathbf{y}) = \operatorname{arcosh}\Big(1 + 2\,\frac{\|\mathbf{x} - \mathbf{y}\|^2}{(1 - \|\mathbf{x}\|^2)(1 - \|\mathbf{y}\|^2)}\Big).$$

We train with a contrastive loss: for each real edge (u, v), sample 5 random non-edges (u, v') and push $d_{\text{hyp}}(\mathbf{e}_u, \mathbf{e}_{v'})$ to be larger than $d_{\text{hyp}}(\mathbf{e}_u, \mathbf{e}_v)$. Optimization uses Riemannian Adam via the `geoopt` library (learning rate 0.3, batch size 2048). Embeddings are initialized near the origin ($\|\mathbf{e}_i\| \approx 0.01$).

The loss begins at 0.693 (the random baseline, since $-\log \sigma(0) = \log 2$) and drops to 0.182 after a single pass through all 12.5 million edges. The model has organized the library into a celestial hierarchy: fundamental logic occupies the core, and the leaves of modern research flare outward toward the boundary.

8 Pipeline: From Geometry to Verified Proof

Can the geometric structure of a formalization library guide the generation of new mathematical results? We searched for constellations of theorems in the hyperbolic sky. Imagine a few stars centered around a location indicating the existence of an undiscovered star at their center.

The pipeline proceeds in five steps:

1. **Dependency graph.** Extract the import/dependency graph.
2. **Poincaré embeddings.** Train the hyperbolic neural network.
3. **Constellation detection.** Search for tight clusters in hyperbolic space.
4. **LLM conjecture.** Present the constellation to a large language model.
5. **Lean 4 verification.** Formalize and verify against Mathlib.

We employ two search strategies.

Strategy 1: Within-namespace void filling. Filter by a mathematical namespace, find the tightest cluster of $K = 5$ theorems, and ask the LLM: "What abstraction sits between these and could unify their proofs?"

Strategy 2: Inter-domain geodesic search. Given two namespaces A and B, sample cross-domain pairs and compute points along the Poincaré geodesic between them using Möbius addition. The geodesic point most distant from any existing theorem is the void. Find the K nearest theorems from $A \cup B$, requiring at least 2 from each domain.

9 Case Study 1: Besicovitch Namespace

Applying Strategy 1 to the Besicovitch namespace of the sphere packing formalization, the tightest constellation consisted of: `TauPackage.index`, `TauPackage.toBallPackage`, `TauPackage.R`, `SatelliteConfig.exists_normalized`, and `BallPackage.r`.

A large language model proposed a new abstraction, `NormalizedBallFamily`—a structure that is, in a sense, "folklore" in the covering-theorem literature but has never been isolated as a standalone definition (as far as I could check). The LLM also produced a lemma establishing that τ-packing is scale-invariant under uniform normalization.

Definition 1 (Normalized Ball Family)**.** *Let (X, d) be a metric space. A* normalized ball family *indexed by a type ι is a tuple (c, r, R) where $c : \iota \to X$ assigns a centre, $r : \iota \to \mathbb{R}$ assigns a positive radius, and $R \in \mathbb{R}$ satisfies $R \geq 1$ and $r_i \leq R$ for all i. The* normalized radius *is $r'_i := r_i/R$.*

Theorem 2 (Scale-invariance of τ-packing)**.** *Let $\{B(c_i, r_i)\}_{i\in\iota}$ be a normalized ball family with bound $R \geq 1$, and let $\tau > 1$. If $S \subseteq \iota$ satisfies*

$$\forall\, i, j \in S,\ i \neq j \implies d(c_j, c_i) > \tau\, r_i,$$

then

$$\forall\, i, j \in S,\ i \neq j \implies d(c_j, c_i) > \tau\, r'_i.$$

Proof. Since $R \geq 1$, $r'_i = r_i/R \leq r_i$, hence $\tau r'_i \leq \tau r_i < d(c_j, c_i)$. □

The Lean 4 proof compiles against Mathlib v4.28.0 with zero errors. To our knowledge, neither the standalone structure nor the factored-out lemma exists in the current Mathlib library; the normalization argument is instead performed inline within `SatelliteConfig.exists_normalized` and its auxiliaries. The full source is in Appendix A.

This result is mathematically elementary. Its significance is as a proof-engineering contribution: factoring out the normalization step as a reusable abstraction.

10 Case Study 2: The Cycle Matroid

Applying Strategy 2 with $A =$ `Matroid` (1,420 nodes) and $B =$ `SimpleGraph` (3,440 nodes), the tightest mixed constellation consisted of:

Domain	**Theorem**
Matroid	`Matroid.Spanning.closure_eq`
Graph	`SimpleGraph.Reachable.dist_anti`
Matroid	`Matroid.Indep.mem_fundCircuit_iff`
Graph	`SimpleGraph.hasseDualIso`
Graph	`SimpleGraph.IsTree.exists_vert_degree_one_of_nontrivial`

These theorems are all about trees, spanning structures, circuits, and independence—exactly the ingredients of graphic matroids. The LLM proposed the *cycle matroid* construction.

Definition 3 (Cycle Matroid)**.** *Let $G = (V, E)$ be a simple graph. A subset $I \subseteq E$ is* independent *in $M(G)$ if (V, I) is acyclic (a forest). The resulting matroid $M(G) = (E, \mathcal{I})$ is called the* cycle matroid *of G.*

This is the most fundamental construction connecting graph theory and matroid theory, introduced by Whitney in 1935. Its matroid axioms hold because: (1) the empty set is acyclic; (2) subsets of forests are forests; (3) if $|I| < |J|$ and both are forests, then J has fewer connected components than I, so some edge of J connects two I-components without creating a cycle.

The Lean 4 formalization constructs `SimpleGraph.cycleMatroid` via Mathlib's `IndepMatroid.ofFinitaryC` and compiles against Mathlib v4.28.0 with three `sorry`s on the substantive proof obligations (exchange property, compactness, edge-set containment) and zero errors elsewhere. The definition, the hereditary property, and the characterization lemma `cycleMatroid_indep_iff` all typecheck completely. The full source is in Appendix B.

This is a qualitatively different result from Case Study 1. The Besicovitch example produced a helper lemma; the cycle matroid produced a *foundational definition*, the canonical functor from graphs to matroids. It seems to fills a genuine structural gap: the `Matroid` and `SimpleGraph` namespaces in Mathlib currently have no connecting construction, despite being the two largest combinatorial components of the library.

What the geometry contributed. The LLM could have proposed the cycle matroid without the embedding. Any combinatorialist would identify it as the missing bridge. What the geometry provided was *attention allocation*: it selected this pair from among all $\binom{18}{2} = 153$ possible namespace pairs, without domain expertise. The void between `Matroid` and `SimpleGraph` in the Poincaré ball was a measurable, geometric signal that corresponded to a real mathematical gap. The geometry is a telescope, not a theorem prover. It shows where to point the LLM.

11 What Did Not Work: Amalgamated Constellations

We also explored a third strategy, inspired by amalgamated free products in group theory. If two constellations in different domains have nearly identical internal geometry, matching pairwise distance patterns, they might represent structurally analogous clusters. The shared dependency set C would serve as the common subgroup, and a new theorem in the "amalgamation" $A *_C B$ might use concepts from both domains.

This approach encountered two obstacles. First, with $K = 5$ members, the pairwise distance signature has only 10 values. This is too low-dimensional to distinguish genuine structural isomorphism from coincidence. Second, when we introduced dependency-graph analysis to identify C concretely, the shared dependencies were dominated by typeclass infrastructure (`AddCommGroup`, `DivisionSemiring`, etc.) the algebraic axiom accessors that Lean autogenerates. Even TF-IDF weighting did not eliminate this problem, because the auto-generated axiom *accessors* are individually rare (high IDF) even though the mathematical content they encode is ubiquitous.

A productive resolution would require a *type-aware* dependency graph that distinguishes typeclass resolution from mathematical content. We leave this as future work.

12 Discussion

The findings suggest that formalized mathematics has a specific, measurable shape. It is a tree of cliques: globally hyperbolic, locally fat. This shape is not a statistical artifact (the null models confirm it), not a human organizational choice (autoformalization preserves it), and not a consequence of hub structure (the configuration model disproves it).

The manifold extrapolation pipeline is a prototype, not a product. The verified results are mathematically elementary. The method has not yet produced a genuinely surprising theorem. But the geometry does real work: it selects candidates that a flat keyword search would never find, precisely because their proximity is encoded in the *shape* of the library rather than its surface syntax.

Several questions remain open. Does a theorem's position in the Poincaré ball predict how hard it was to prove? Can the tree-of-cliques structure improve proof search? As Mathlib grows, does δ remain stable? Is the architecture universal across proof assistants? And can the geometry ever find something that surprises a domain expert—a void that corresponds not to a missing formalization but to genuinely unknown mathematics?

We leave these questions open, and hope that students reading this will find them exciting.

Acknowledgements

Jeremy Avigad suggested sharing this work on the Lean Zulip. Anne Baanen moved the post to its own thread. Yakov Pechersky proposed the autoformalization hypothesis that led to our strongest empirical comparison. Johan Commelin raised the pruning concern that motivated the null-model experiments. Patrick Massot shared his experience with math-cognizant pruning for the perfectoid spaces project. Siddhartha Gadgil and Matthew Coke provided encouragement. I thank the entire Mathlib community for maintaining the library that makes this investigation possible.

Data and Code Availability

The Mathlib dependency graph, trained Poincaré ball embeddings, all analysis scripts (including null-model tests), and the verified Lean 4 proofs are available at `https://github.com/ashanidasgupta1/mathlib-geometry`.

A Lean 4 Source: NormalizedBallFamily

The following compiles against Mathlib v4.28.0 with zero errors.

```
import Mathlib.Topology.MetricSpace.Basic
import Mathlib.Order.ConditionallyCompleteLattice.Basic

structure NormalizedBallFamily
    (ι : Type*) (X : Type*)
    [MetricSpace X] where
  c : ι → X
  r : ι → ℝ
  r_pos : ∀ i, 0 < r i
  rMax : ℝ
  rMax_pos : 0 < rMax
  rMax_ge_one : 1 ≤ rMax
  r_le : ∀ i, r i ≤ rMax

namespace NormalizedBallFamily
variable {ι : Type*} {X : Type*}
         [MetricSpace X]
         (F : NormalizedBallFamily ι X)

noncomputable def r' (i : ι) : ℝ :=
  F.r i / F.rMax

lemma r'_pos (i : ι) : 0 < F.r' i :=
  div_pos (F.r_pos i) F.rMax_pos

lemma r'_le_one (i : ι) :
    F.r' i ≤ 1 :=
  div_le_one_of_le₀ (F.r_le i)
    (le_of_lt F.rMax_pos)

lemma tau_packing_invariant
    (τ : ℝ) (hτ : 1 < τ)
    (S : Set ι)
    (hpack :
      ∀ i ∈ S, ∀ j ∈ S, i ≠ j →
        dist (F.c j) (F.c i) > τ * F.r i) :
    ∀ i ∈ S, ∀ j ∈ S, i ≠ j →
      dist (F.c j) (F.c i) > τ * F.r' i
    := by
  intro i hi j hj hij
  have h := hpack i hi j hj hij
  have hr : τ * F.r' i ≤ τ * F.r i := by
    apply mul_le_mul_of_nonneg_left _
      (by linarith)
    exact div_le_self
      (le_of_lt (F.r_pos i))
      F.rMax_ge_one
  linarith

end NormalizedBallFamily
```

B Lean 4 Source: Cycle Matroid

The following compiles against Mathlib v4.28.0 with three `sorry`s and zero errors.

```
import Mathlib.Combinatorics.SimpleGraph.Acyclic
import Mathlib.Combinatorics.SimpleGraph.Connectivity.Subgraph
import Mathlib.Combinatorics.Matroid.IndepAxioms
```

```
open Set SimpleGraph
namespace SimpleGraph
variable {V : Type*} (G : SimpleGraph V)

def IsAcyclicEdgeSet (S : Set (Sym2 V)) : Prop :=
  ∀ {v : V} (p : G.Walk v v),
    (∀ e ∈ p.edges, (e : Sym2 V) ∈ S)
      → p.Nil

theorem isAcyclicEdgeSet_empty :
    G.IsAcyclicEdgeSet ∅ := by
  intro v p hp
  rw [← Walk.edges_eq_nil]
  exact List.eq_nil_iff_forall_not_mem.mpr
    fun e he ⇒ by simpa using hp e he

theorem IsAcyclicEdgeSet.subset {S T}
    (hT : G.IsAcyclicEdgeSet T)
    (h : S ⊆ T) :
    G.IsAcyclicEdgeSet S := by
  intro v p hp
  exact hT p (fun e he ⇒ h (hp e he))

section Finite
variable [Fintype V] [DecidableEq V]
         [DecidableRel G.Adj]

theorem IsAcyclicEdgeSet.augment {I J}
    (hI : G.IsAcyclicEdgeSet I)
    (hI_fin : I.Finite)
    (hJ : G.IsAcyclicEdgeSet J)
    (hJ_fin : J.Finite)
    (hIJ : I.ncard < J.ncard) :
    ∃ e ∈ J, e ∉ I ∧
      G.IsAcyclicEdgeSet (insert e I) := by
  sorry -- connected-components argument

theorem isAcyclicEdgeSet_compact (I)
    (h : ∀ J ⊆ I, J.Finite →
      G.IsAcyclicEdgeSet J) :
    G.IsAcyclicEdgeSet I := by
  sorry -- cycles are finite

noncomputable def cycleMatroid :
    Matroid (Sym2 V) :=
  (IndepMatroid.ofFinitaryCardAugment
    (E := G.edgeSet)
    (Indep := fun S ⇒
      G.IsAcyclicEdgeSet S
        ∧ S ⊆ G.edgeSet)
    (indep_empty :=
      ⟨G.isAcyclicEdgeSet_empty,
       Set.empty_subset _⟩)
    (indep_subset :=
      fun _ _ hJ hIJ ⇒
        ⟨hJ.1.subset hIJ,
         Set.Subset.trans hIJ hJ.2⟩)
    (indep_aug :=
      fun I J hI hIfin hJ hJfin hc ⇒ by
        obtain ⟨hIa, hIs⟩ := hI
        obtain ⟨hJa, hJs⟩ := hJ
        obtain ⟨e, heJ, heI, hea⟩ :=
          hIa.augment hIfin hJa hJfin hc
        exact ⟨e, heJ, heI, hea,
          Set.insert_subset
            (hJs heJ) hIs⟩)
    (indep_compact :=
      fun I hI ⇒ by
        constructor
        · exact G.isAcyclicEdgeSet_compact I
            (fun J hJI hJf ⇒
              (hI J hJI hJf).1)
        · intro e he
          exact (hI {e}
            (Set.singleton_subset_iff.mpr he)
            (Set.finite_singleton _)).2
            (Set.mem_singleton _))
    (subset_ground :=
      fun _ hI ⇒ hI.2)).matroid

theorem cycleMatroid_indep_iff {S} :
    G.cycleMatroid.Indep S ↔
      G.IsAcyclicEdgeSet S
        ∧ S ⊆ G.edgeSet := by
  simp [cycleMatroid,
    IndepMatroid.ofFinitaryCardAugment]

end Finite
end SimpleGraph
```