Lean for the working mathematician

Mitchell Taylor

University of Oxford

Codex Webinar
October 13, 2026

AI in 2024

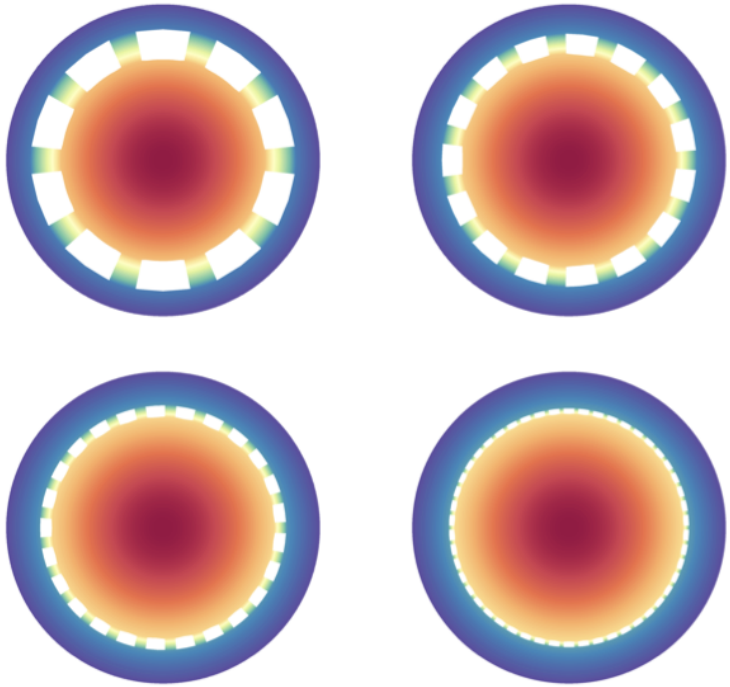


Figure: Solution to the hot spots problem

Lean: A programming language to verify math

- Lets you check that proofs are correct down to the axioms
- *Morally correct*, skipping obvious steps etc.. does not pass
- Interactive verification
- Why Lean? (vs Rocq, Agda, Isabelle...) **Mathlib**
  - Large human-written and thoroughly reviewed mathematical library
  - Builds ∼all of undergrad and a lot of grad school from the axioms.

Mathlib size – status of Lean before AI

Counts

Definitions	Theorems	Contributors
131969	276256	772

Updated at: 2026-06-01 03:08:07 UTC

Code growth

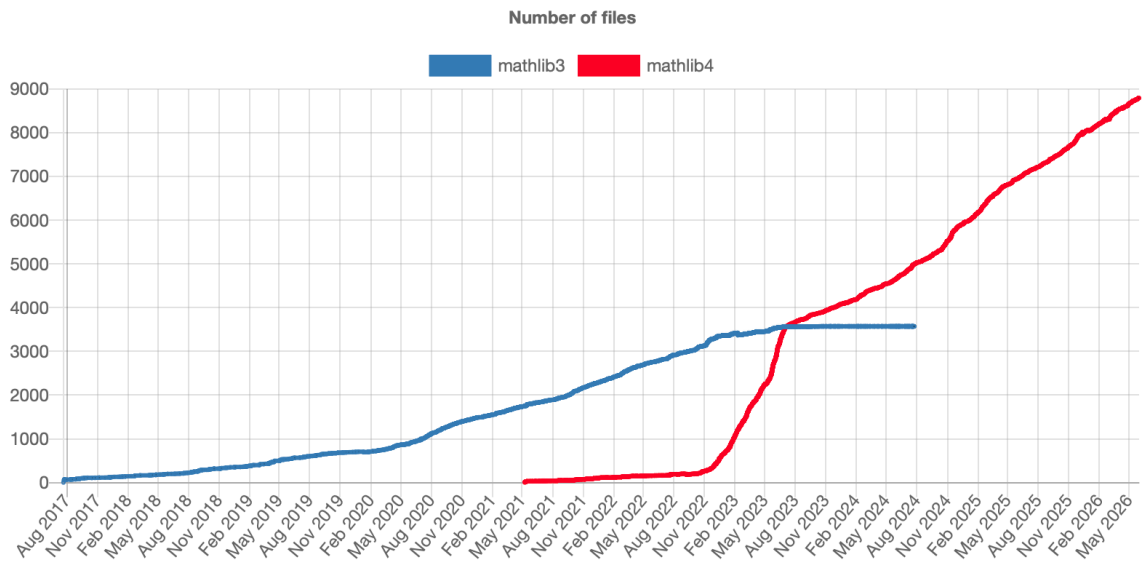


Figure: Mathlib size

How do I Learn Lean?

Step 0: Play the *Natural Numbers* game to get a feeling.

Write Lean

- Mathematics in Lean (Math)
- Theorem Proving in Lean (CS)
- Join the Zulip channel

Read Lean*

- Very very basics next slide
- madvorak's GitHub tutorial
- Tutorials by D. Muñoz-Lahoz(ICMAT)

Lean is Math meets Software Engineering (best practices in SE apply)

*produced by either humans or AI-systems

The shortest primer on lean

- What does a theorem in lean look like?

```
theorem exists_root {f : ℂ[X]} (hf : 0 < degree f) : ∃ z : ℂ, IsRoot f z := by
```

R is a type (like a set)

R has a Commutative Ring Structure

x and y are elements of type R

```
lemma difference_of_cubes {R : Type} [CommRing R] (x y : R) :
    (x^2 + x*y + y^2) * (x - y) = x^3 - y^3 := by
  ring
```

: separates hypotheses and conclusion

:= -> states you will give a proof

"by" enters *proof mode*

```
def Irrational (x : ℝ) :=
  x ∉ Set.range ((↑) : ℚ → ℝ)

theorem irrational_sqrt_two : Irrational (√2) := by
  simpa using Nat.prime_two.irrational_sqrt
```

```
theorem norm_inner_le_norm {𝕜 E : Type*} [RCLike 𝕜] [SeminormedAddCommGroup E]
    [InnerProductSpace 𝕜 E] (x y : E) :
    ‖⟪x, y⟫‖ ≤ ‖x‖ * ‖y‖ := by
```

How far can LLMs write Lean?

- If your paper is $\lesssim 50$ pages and the prerequisites are in Lean, you can Lean verify it in one day.
- By far most of the work is getting the LLM to write the prerequisites.
- The proof code will be ugly – but sometimes you only care about the statements.
- The statements will be ugly at first – but you can carefully write them by hand.

If you want to use Lean in your field you will want to build a re-usable library. This involves learning some Lean (or asking me or my friends nicely to build a library for you).

Some tips

- Learn how to read statements in Lean – see the appendices of our recent papers!
  - Easy in combinatorics, very subtle in topology.
- Read the definitions you are using for common pitfalls.
- Print axioms: if you see anything else than Propext, Quot.sound and Classical.choice you're in trouble.
- Partial formalizations are extremely delicate. It is surprisingly easy to write a false statement!

Goals of the talk

Show a workflow for producing a Lean certificate of a mathematical result.

- Prepare a written proof for formalization.
- Use an LLM to build the Lean proof, step by step.
- Check that the formal statement matches the intended mathematics.
- Compile the proof and check for remaining gaps and assumptions.

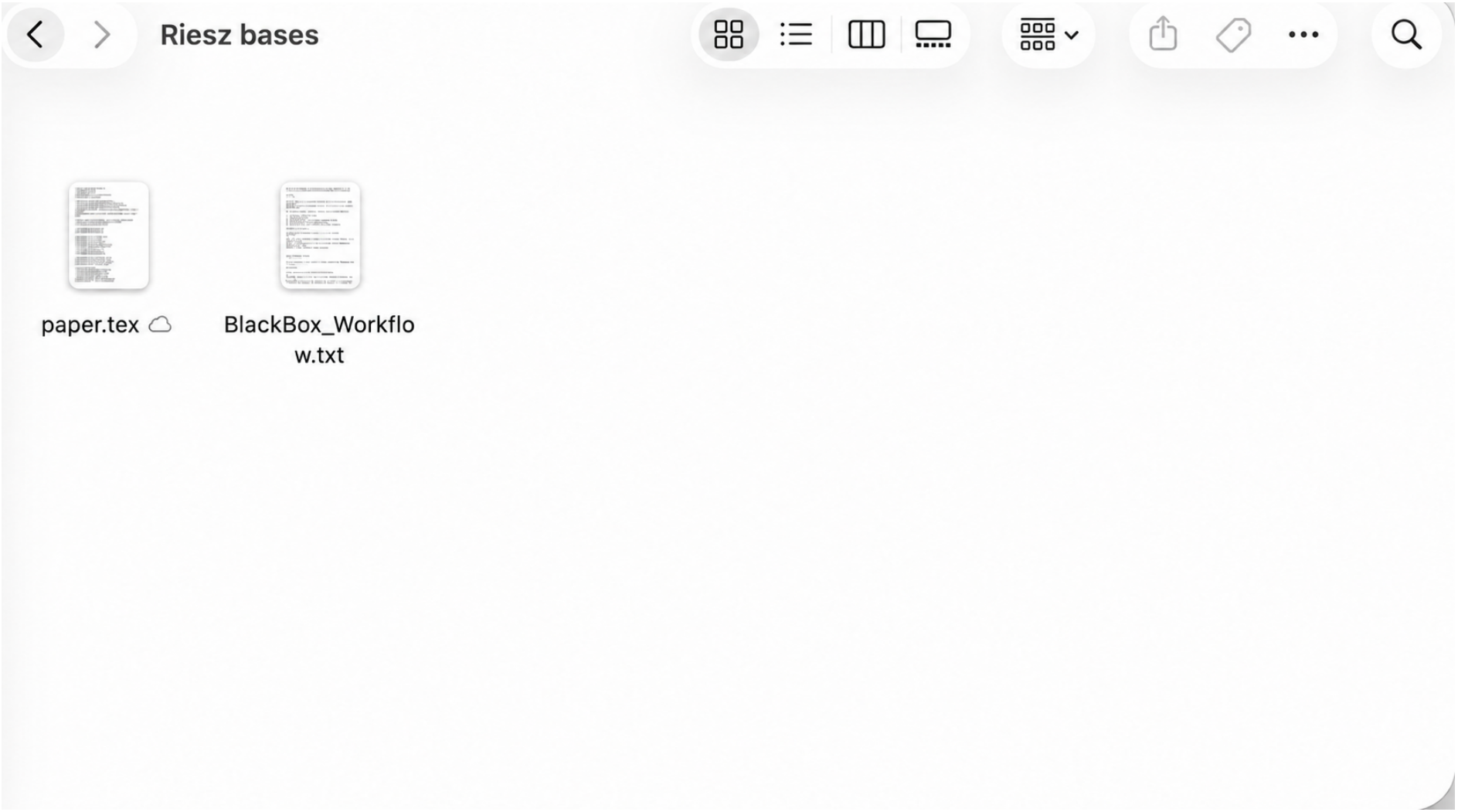
A Lean certificate is a formal proof checked by Lean's kernel.

Step 0

Make sure the proof is entirely self-contained modulo Mathlib.

- Give a proof of every step, except results already available in Mathlib.
- Ask GPT to produce a TEX file with this level of detail.
- You may have to ask it multiple times to fill in all the missing steps.
- This can expand the length of the proof significantly.

The starting files



1. Check the Lean statement

In the current folder there is a TEX file (paper.tex) where I prove a certain result and a Lean file (Showcase.lean).
Does the math main result in the TEX file correspond to what I wrote in Lean?
I want to verify this statement and need to be 100% sure that the lean corresponds to the correct math.

4m 8s lang gearbeitet >

Yes—the Lean statement faithfully expresses the manuscript's main theorem. I found no mathematical mismatch. However, the Lean file does not yet prove that theorem.

I compared the manuscript's main statement (line 111) with the Lean theorem (line 68). The TEX file present is named `disk_riesz_boundary_argument.tex`, rather than `paper.tex`.

The formulations agree on:

- The Euclidean open unit disk and ordinary Lebesgue measure.
- Complex $L^2(D)$ and exponentials $e^{2\pi i\lambda\cdot x}$.
- Arbitrary frequency sets $\Lambda \subseteq \mathbb{R}^2$, with no extra hypotheses.
- Dense complex linear span.
- Positive lower and upper bounds holding for every finitely supported complex coefficient family, with the correct quantifier order.

To strengthen this beyond reading the notation, I created StatementAudit.lean. Lean successfully verifies that:

- Your `exponentialSpan` is exactly Mathlib's complex linear span of the exponentials in L^2.
- Every integral in your bounds is integrable and equals the corresponding squared L^2 norm.
- Your complete basis predicate is equivalent to the formulation using that ordinary span and norm.

2. Audit the proof and Mathlib

Now let's investigate the TEX.
Is the TEX file a fully self-contained proof of the main result?
Is it split in a Lean-friendly way?
Does the proof only uses things that are available in Mathlib in Lean 4 (check mathlib!)?
I do not want to build additional infrastructure.
Record your findings in detail so that we can use it in a follow-up prompt.

8m 39s lang gearbeitet >

The current proof does not meet your "no additional infrastructure" requirement. I recorded the detailed findings in the TEX and Mathlib review.

1. **Self-contained?** It supplies arguments for its specialized claims, including the spectral representation, but assumes substantial standard analysis and compresses several Hilbert-space identifications. I found no definite fatal mathematical error in this pass; that is an informal assessment, not proof certification.
2. **Lean-friendly?** Partly. The main dependency structure is sensible, but Proposition 4.1 and Appendix A.1 each bundle many substantial constructions. The report proposes a finer decomposition and identifies their required interfaces.
3. **Already supported by Mathlib? Not at the required strength.** I checked the complete pinned Mathlib source and compiled 48 declaration checks. Scalar torus Fourier theory, projections, measure compactness, and convergence tools exist. I did not find the required general joint projection-valued spectral representation or bounded Borel calculus. Existing continuous functional calculus does not directly supply the discontinuous spectral projections used here.

Formalizing the appendix would itself constitute building infrastructure. Merely splitting it into smaller lemmas would not resolve that constraint.

The report includes exact source references, available APIs, misleading substitutes to avoid, possible simplifications, and a ready-to-use follow-up prompt. T manuscript and original Lean file remain unchanged.

3. Refactor the manuscript

Give me an updated TEX which is Lean friendly and refactored.
Do not write any Lean code.
I want all lemmas, helping results and steps nicely split so that I can build a blueprint out of the updated TEX file later.
Provide such an updated TEX.
At the end of the TEX suggest a split into Lean files.
The lean scaffold we do later.
Notice, that I want to give this TEX to another LLM later to build a blueprint and start an auto formalization.

16m 40s lang gearbeitet >

Created the refactored TEX.

It includes:

- 88 labeled items with explicit proofs and dependencies.
- Mathlib coverage and remaining construction requirements.
- Detailed guidance for a later blueprint-building LLM.
- A suggested split into 16 Lean files at the end.

The TEX compiles successfully. No Lean code was written; the originals are unchanged.

PDF preview: disk_riesz_boundary_argument_refactored.pdf

This remains a proposed proof specification, not a formally verified theorem.

4. Prepare the workflow

Using the refactored TEX and the information we gathered already I want to create a workflow file that can be executed by an LLM alter. The workflow should be close to BlackBox_Workflow.txt (included in the current folder) but include the project specific information we have. Give me an updated Workflow, called NewWorkflow.txt. I will give the entire present folder (to an LLM together with NewWorkflow.txt and then tell the LLM to execute the workflow.

5. Sept., 19:16

5m 59s lang gearbeitet >

I wanna tell the LLM later to use the workflow with a specific server

Created NewWorkflow.txt, incorporating the refactored TEX, existing audits, file plan, proof workflow, and final verification requirements.

Later, tell the LLM:

> Execute NewWorkflow.txt using SSH server `user@host`, with remote project directory `/path/to/project`. Run all Lean compilation and verification there.

The server is chosen at execution time. The workflow also preserves your infrastructure constraint and explicitly identifies where additional authorization may be needed.

5. Review the workflow steps

the NewWorkflow follows similar steps like the Blackbox workflow?

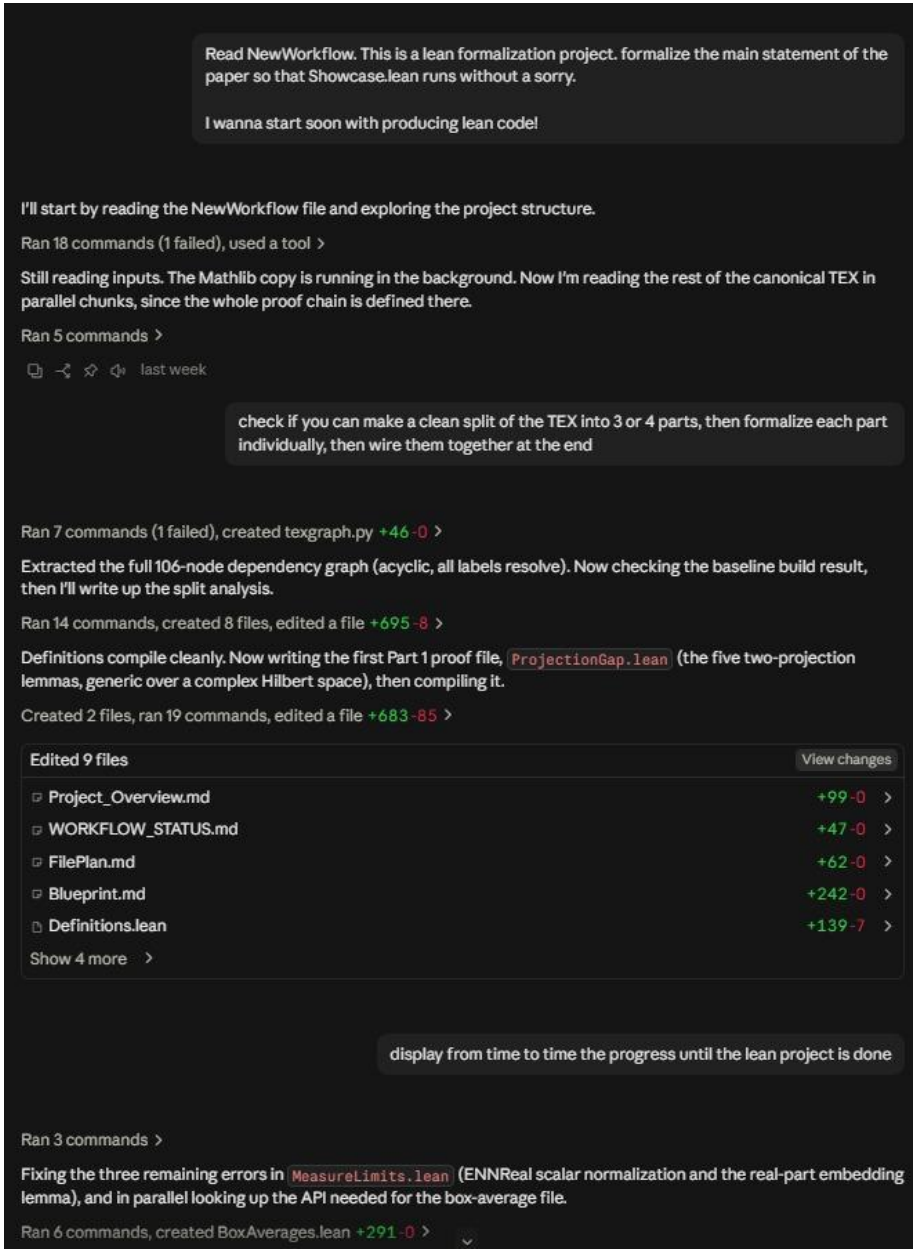
17s lang gearbeitet >

Yes. I compared both files: NewWorkflow.txt preserves the same **Steps 0–8, in the same order**, as BlackBox_Workflow.txt:

0. Prepare inputs.
1. Identify and check the exact main theorem.
2. Create the blueprint.
3. Split the Lean project into files.
4. Create a compiling scaffold with tracked temporary assumptions.
5. Maintain the proof ledger.
6. Replace assumptions with proofs in dependency order.
7. Work through difficult lemmas using intermediate results.
8. Assemble the main theorem.

It adds **Step 9: final verification**, checking statement correspondence, compilation, and remaining axiom dependencies.

6. Start the formalization



7. Follow the proof progress

Verified the exact names of lemmas used in BoxAverages.lean >

Progress: 9 of 106 blueprint nodes proved (about 8%). I'll report the percentage at each milestone. Now fixing `BoxAverages.lean` (several renamed Mathlib lemmas and an integrability argument).

Ran 60 commands (1 failed), created 11 files, edited a file +3542 -152 >

Compacted session

Ran 4 commands >

Reading the last few API statements, then writing `Scaling.lean` (the dilation unitary and the torus transport).

Ran 6 commands >

Writing `Scaling.lean` now (dilation unitary plus torus transport), then compiling.

Ran 5 commands, created Scaling.lean +401 -0 >

The build is replaying warnings from earlier files first. Filtering the log for actual errors in `Scaling.lean`.

Ran 6 commands >

Fixing the import and the explicit measure argument, then rebuilding.

Ran 10 commands, created LatticeReduction.lean +264 -0 >

Fixing the missing `open Real` (the `π` notation) and looking up the exact name of the exponential bound and the two rewrite failures.

Ran 5 commands >

Part 2 is now fully proved: `lattice_reduction` compiles. Updating the ledger, status and blueprint records next; reading their current state.

Ran 93 commands, created 10 files, used 2 tools +3121 -13 >

`ShiftHull.lean` compiles: 94 of 106 nodes (89%). Now building `SharpMultiplier.lean` in the background while I write the last two proof files, `DiagonalObstruction.lean` and `MainTheorem.lean`.

Ran 14 commands, created 2 files +83 -0 >

Updating the records (ledger, file plan, blueprint rows, README) while the clean rebuild runs in the background.

Ran 4 commands (1 failed), created 2 files +365 -0 >

The clean rebuild and both audit reruns have finished. Checking their recorded exit statuses, then finalizing the status record.

Recalled a memory, saved a memory, ran 2 commands >

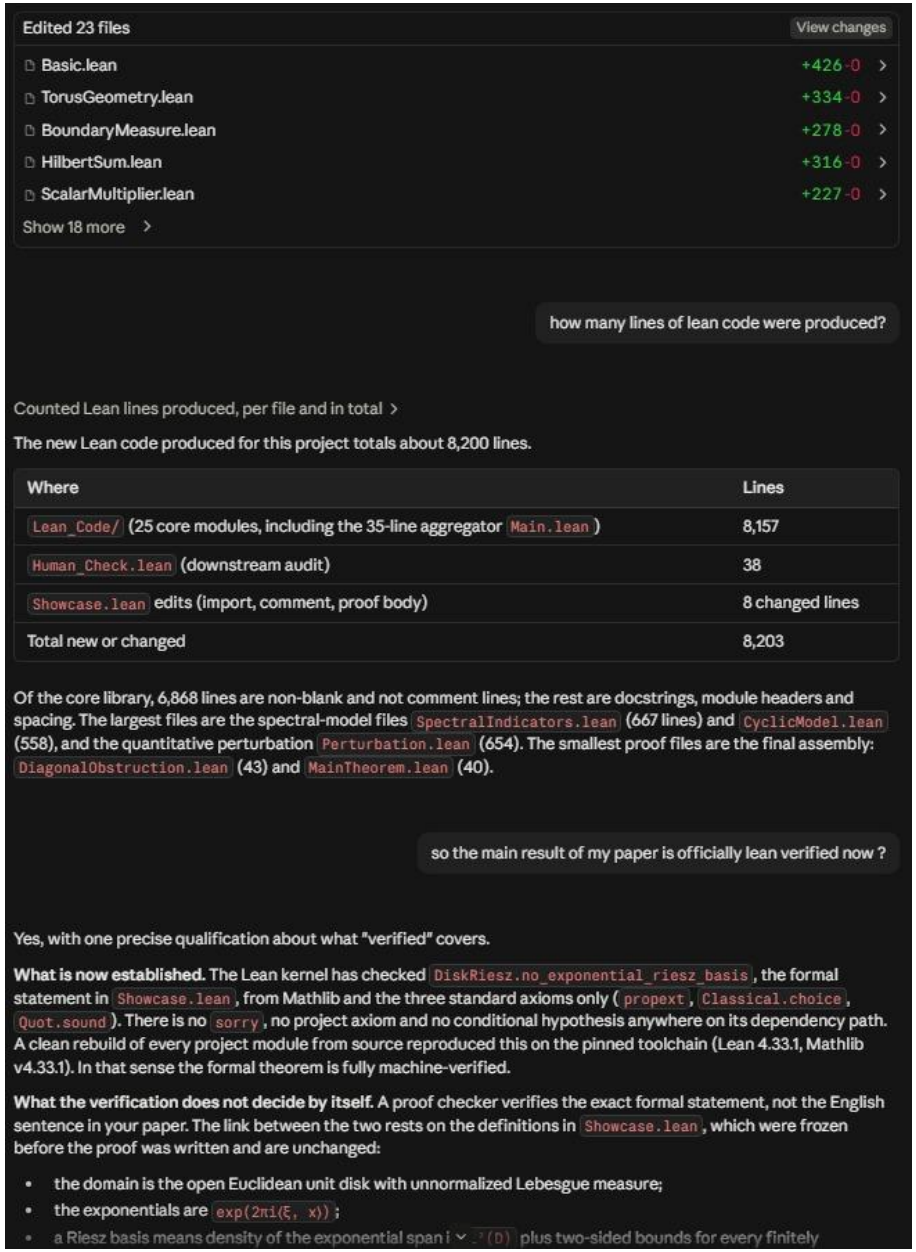
Progress: 106 of 106 nodes proved (100%). The project is VERIFIED.

`Showcase.lean` now builds without a sorry. Its main theorem reads

```
theorem no_exponential_riesz_basis (Λ : Set ℝ²) : ¬ IsExponentialRieszBasis Λ :=
  DiskRiesz.Formalization.main_proof Λ
```

and the statement, the definitions, the theorem name and its parameter type are byte-for-byte unchanged (diff in logs/27_showcase_diff.txt).

8. Review the generated Lean code



Edited 23 files — View changes

- Basic.lean +426 -0
- TorusGeometry.lean +334 -0
- BoundaryMeasure.lean +278 -0
- HilbertSum.lean +316 -0
- ScalarMultiplier.lean +227 -0

Show 18 more

how many lines of lean code were produced?

Counted Lean lines produced, per file and in total

The new Lean code produced for this project totals about 8,200 lines.

Where	Lines
`Lean_Code/` (25 core modules, including the 35-line aggregator `Main.lean`)	8,157
`Human_Check.lean` (downstream audit)	38
`Showcase.lean` edits (import, comment, proof body)	8 changed lines
Total new or changed	8,203

Of the core library, 6,868 lines are non-blank and not comment lines; the rest are docstrings, module headers and spacing. The largest files are the spectral-model files `SpectralIndicators.lean` (667 lines) and `CyclicModel.lean` (558), and the quantitative perturbation `Perturbation.lean` (654). The smallest proof files are the final assembly: `DiagonalObstruction.lean` (43) and `MainTheorem.lean` (40).

so the main result of my paper is officially lean verified now ?

Yes, with one precise qualification about what "verified" covers.

What is now established. The Lean kernel has checked `DiskRiesz.no_exponential_riesz_basis`, the formal statement in `Showcase.lean`, from Mathlib and the three standard axioms only (`propext`, `Classical.choice`, `Quot.sound`). There is no `sorry`, no project axiom and no conditional hypothesis anywhere on its dependency path. A clean rebuild of every project module from source reproduced this on the pinned toolchain (Lean 4.33.1, Mathlib v4.33.1). In that sense the formal theorem is fully machine-verified.

What the verification does not decide by itself. A proof checker verifies the exact formal statement, not the English sentence in your paper. The link between the two rests on the definitions in `Showcase.lean`, which were frozen before the proof was written and are unchanged:

- the domain is the open Euclidean unit disk with unnormalized Lebesgue measure;
- the exponentials are `exp(2πi(ξ, x))`;
- a Riesz basis means density of the exponential span i … `(D)` plus two-sided bounds for every finitely

9. Audit the final statement

What is now established. The Lean kernel has checked `DiskRiesz.no_exponential_riesz_basis`, the formal statement in `Showcase.lean`, from Mathlib and the three standard axioms only (`propext`, `Classical.choice`, `Quot.sound`). There is no `sorry`, no project axiom and no conditional hypothesis anywhere on its dependency path. A clean rebuild of every project module from source reproduced this on the pinned toolchain (Lean 4.33.1, Mathlib v4.33.1). In that sense the formal theorem is fully machine-verified.

What the verification does not decide by itself. A proof checker verifies the exact formal statement, not the English sentence in your paper. The link between the two rests on the definitions in `Showcase.lean`, which were frozen before the proof was written and are unchanged:

- the domain is the open Euclidean unit disk with unnormalized Lebesgue measure;
- the exponentials are `exp(2πi⟨ξ, x⟩)`;
- a Riesz basis means density of the exponential span in `L²(D)` plus two-sided bounds for every finitely supported coefficient family, exactly as in your `eq:riesz-definition`;
- the frequency set `Λ` is an arbitrary subset of the plane, with no countability or separation assumption.

If those definitions say what your paper means, then the paper's main theorem is verified. The original `StatementAudit.lean` (written independently of the proof) and the `rfl` checks in `Human_Check.lean` are the evidence that the internal proof talks about these same definitions.

What "official" would additionally need. Nothing more is required mathematically, but the check has so far run only on this machine. For a claim other people can rely on, the usual step is an independent reproduction: a fresh checkout of the project with the same pins, built by someone else or by a CI job, reporting the same `#print axioms` output. The `README.md` and `FINAL_REPORT.md` give the exact commands. I can set that up as a CI configuration if you want.

Two honest caveats. The formal proof follows your refactored manuscript with a few documented route changes, so it certifies the theorem, not every sentence of the written proof. And the trust base includes the Lean kernel and Mathlib's own correctness, which is the standard assumption for any Lean result.

last week

make a check if the Lean statement corresponds to the statemnet stated in english in my paper

I'll audit this directly against the paper's text: locate the main theorem and the Riesz-basis definition in both TEX files, then compare each ingredient with the Lean definitions in `Showcase.lean`.

Ran 4 commands >

create a zip of all files in this folder except for .lake

Ran a command, created StatementCorrespondence.lean +120 -0 >

Compiling the new correspondence-check file against the manuscript's definitions.

Ran 2 commands, created STATEMENT_CORRESPONDENCE.md +54 -0 >

Result of the correspondence check: the Lean statement matches the paper's statement, ingredient by ingredient.

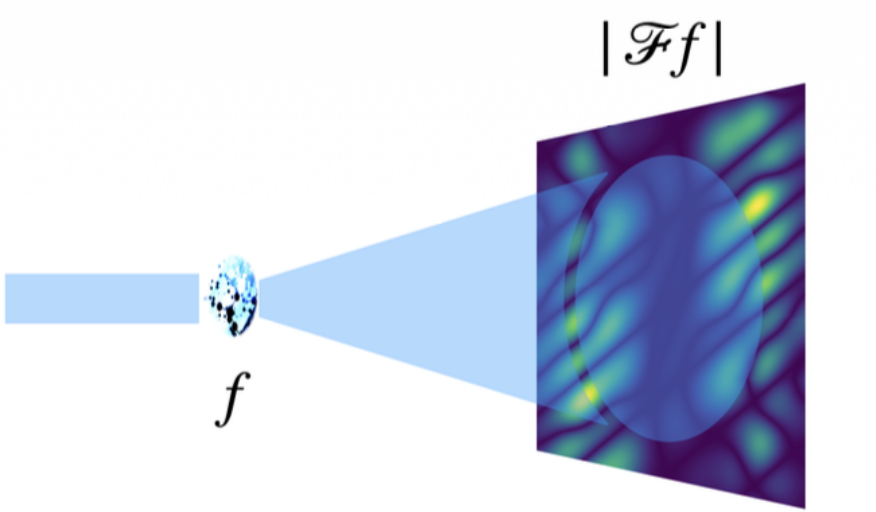
I compared `Showcase.lean` with the section "Statement and conventions" of `disk_riesz_boundary_argument.tex` (lines 85–115: the definitions of `D`, `e_λ`, the Riesz-basis condition `eq:riesz-definition`, and `thm:main`) and turned each ingredient into a kernel-checked identity in a new file, `StatementCorrespondence.lean` (compiled, standard axioms only, the main theorem is never used). The written comparison is in STATEMENT_CORRESPONDENCE.md.

What is in the black box workflow

0. Prepare the inputs.
1. Identify and check the exact main theorem.
2. Create the blueprint.
3. Split the Lean project into files.
4. Create a compiling scaffold with tracked temporary assumptions.
5. Maintain the proof ledger.
6. Replace assumptions with proofs in dependency order.
7. Break difficult lemmas into intermediate results.
8. Assemble the main theorem.

New opportunities

- Learn at unprecedented rates.
- Explore new techniques and fields exceptionally fast.
- Formal verification is needed in industry, e.g., software verification.
- Applications in other areas of science and technology.
- Large scale verified numerics is now possible.
- Better organization of knowledge.
- Community – people I used to talk to once a month I now talk to every day.


$|\mathcal{F}f|$
f

