The State of Lean

Leo de Moura
Chief Architect, Lean FRO
Senior Principal Applied Scientist, AWS

Lean Together, January 2026

Lean is an open-source programming language and proof assistant.

Lean and its tooling are implemented in Lean. Lean is very **extensible**.

LSP, Parser, Macro System, Elaborator, Type Checker, Tactic Framework, Proof automation, Compiler, Build System, Documentation Authoring Tool.

Lean has a **small trusted kernel**, proofs can be exported and independently checked.

> "Lean is not just interesting as a theorem prover. It is an example of a ***well-designed, modern programming language*** that can serve as a model for current and future languages."
>
> — Harry Goldstein, DC Systems

lean-lang.org

2025

A Transformative Year for Lean

2025 By The Numbers

12

Releases

v4.15 – v4.26

90K+

VS Code

Unique Installs

7,100+

GitHub Stars

50+

University Courses

~4,000

PRs to Lean 4 Repo

2

Major Awards

5+

AI Startups on Lean

2025 Key Milestones

MAY
DeepMind Formal Conjectures in Lean

JUNE
ACM SIGPLAN Software Award

JULY
IMO Gold: Harmonic and BydeDance
Skolem Award, $10M Gerko

AUGUST
grind, Carleson

SEPTEMBER
AI Math Fund: $18M, 14 feature Lean

OCT-DEC
Velvet, Aleph, Axiom, Erdős problems

New Website
lean-lang.org rebuilt with Verso

ICARM Institute
NSF institute at CMU

CSLib
leanprover/cslib launched

Recognition in 2025

JUNE 2025

ACM SIGPLAN Programming Languages Software Award

Recognizing Lean's impact on programming languages research and practice

JULY 2025

Skolem Award

For the 2015 CADE paper

"The Lean Theorem Prover"

2025 Funding

July 2025 — Alex Gerko

$10M

$5M — Mathlib Initiative

$5M — Lean FRO: Next-Gen UX

Next-Gen UX Initiative (3 years)

Team hired and ramping up

Literate programming (Verso), Lean Online Workbench, scientific paper authoring, educational resources, VS Code enhancements

Team: Principal + 4 Senior Engineers

Existing support: Simons Foundation, Alfred P. Sloan Foundation, Richard Merkin, Alex Gerko

The Lean FRO team

2025 HIGHLIGHTS

Compiler & Performance

New Module System & Parallel Elaboration

Module System Benefits

Robust module interfaces, better build times, reduced memory footprint. New shake import minimizer bundled with Lean.

Parallel Elaboration

Drastically improves latency in proof-heavy files and makes builds scale to many more cores.

Memory opening all of Mathlib

6GB → 3GB

50% reduction

Sebastian Ullrich will present on Tuesday.

Continued Performance Improvements

Measured on Mathlib4 build instructions:

v4.17.0	v4.20.0	v4.21.0	v4.24.0	v4.26.0	v4.27.0
-2.3%	-2.7%	-9.7%*	-2.9%	-2.8%	-2.2%

Scalability: Large inductive data types and match statements handled more gracefully.

***Of the 9.7% reduction, 8.1%** was due to an external PR by Jovan Gerbscheid. Many thanks to him!

The New Compiler

Lean Compiles Itself

In 2025, we replaced the old C++ bootstrap compiler with a new one fully implemented in Lean.

Development started in late 2022, frozen until 2025

Now: Only the kernel and runtime are still in C/C++

What This Enabled

Zero Cost BaseIO — only possible with new compiler infrastructure

Many optimizations — easier to implement and maintain in Lean

GitHub issues closed — long-standing bugs fixed by the transition

A major milestone in Lean's self-hosting journey.

2025 HIGHLIGHTS

Proof Automation

bv_decide: Verified Bit-Blasting

The fastest verified bit-blaster, now solving **96%** of SMT-LIB bitvector problems.

Best solver Bitwuzla sits at 99%.

Key components — all implemented and verified in Lean:

Bit-blaster, AIG (And-Inverter Graph), LRAT SAT proof checker, CaDiCaL SAT Solver integration

96%

SMT-LIB bitvector problems solved

New in 2025: Support for enum and structure types

The **grind** Tactic

New proof automation (v4.22, August 2025).

A proof-automation tactic inspired by modern SMT solvers. Think of it as a **virtual whiteboard**:

Discovers new equalities, inequalities

Merges equivalent terms

Multiple engines cooperate

Cooperating Engines: Congruence closure, E-matching, Constraint propagation, Guided case analysis

Satellite Solvers: Linear integer arithmetic, commutative rings (Gröbner basis), fields

Kim Morrison will dive deeper into grind later this week.

grind: Design Principles

Native to Dependent Type Theory

No translation to first-order or higher-order logic

Fast Startup Time

No server startup, no external dependencies

Rich Diagnostics

When it fails, it tells you why

Extensible

Users can plugin their own theory solvers

No Mathlib Dependency

Works with just the standard library

Configurable via Type Classes

Stdlib and Mathlib pre-annotated

Great for software verification applications.

2,800+ occurrences in Mathlib

2025 HIGHLIGHTS

Tooling & Developer Experience

Lake: Build System Improvements

Remote Caching

New lake cache for sharing build artifacts — built-in replacement for Mathlib's cache

Multi-Version Workspaces

Experimental support for multiple versions of a package in the dependency tree

Local Artifact Cache

Share artifacts between package instances on a system; fast rebuilds when switching branches

New Targets

input_file, input_dir, improved inter-target dependencies

Semantic Version Ranges

require foo @ "≥1.2.3, ≤1.2.8"

lakefile.toml Support

Basic language server support for TOML configuration

Language Server & VS Code

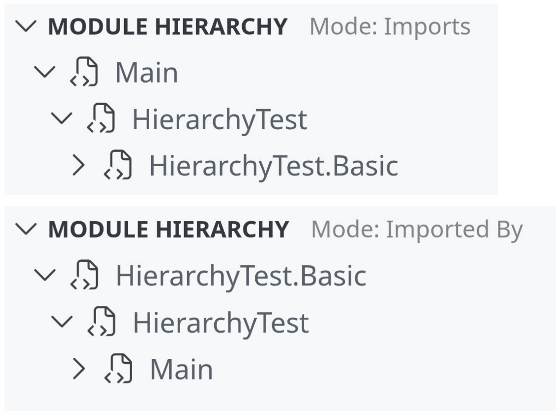
Language Server

3.5× autocomplete speedup

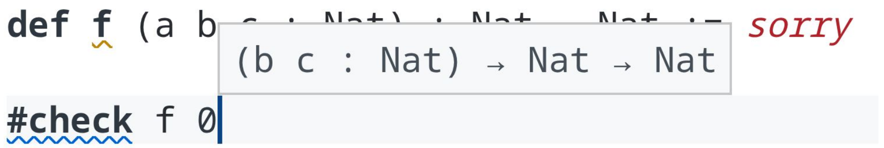
Signature help, auto-implicit inlay hints, unknown identifier code actions, go-to-definition for type class instances, module hierarchy navigation, trace search

VS Code Extension

Gutter decorations (errors, warnings, ✓), "Goals accomplished!" message, simplified one-step installation, Elan integration, InfoView UI improvements



```
MODULE HIERARCHY   Mode: Imports
  Main
    HierarchyTest
      HierarchyTest.Basic

MODULE HIERARCHY   Mode: Imported By
  HierarchyTest.Basic
    HierarchyTest
      Main
```



```
def f (a b c : Nat) : Nat → Nat := sorry
           (b c : Nat) → Nat → Nat
#check f 0
```

Better Error Messages

Error Explanations

Integrated examples and extra context directly into error messages

Smart Identifier Hints

Manual knows to redirect *Rustaceans* looking for `Result` to `Except`

"Most Wanted" Bad Messages

Systematically identified and fixed the worst offenders

Before

```
typeclass instance problem is stuck, it is often due to metavariables
 HAdd ?m.9 ?m.9 (?m.3 x)
```

After

```
typeclass instance problem is stuck
 HAdd ?m.9 ?m.9 (?m.3 x)
Note: Lean will not try to resolve this typeclass instance problem because the first
and second type arguments to `HAdd` are metavariables. These arguments must
be fully determined before Lean will try to resolve the typeclass.
```

"it is often due to metavariables" is gone forever

2025 HIGHLIGHTS

Language & Library Features

Standard Library Expansion

Iterators

Composable abstraction with combinators like mapM, takeWhile. Verification library included.

Async Primitives

libuv bindings, networking, signal handlers, HTTP server.

Ranges & Slices

Polymorphic ranges (1...5) and slices (xs[1...5])

Verified Containers

HashMap and TreeMap verification. Set operations with lemmas.

Grove

New tool for tracking library defects at scale.

Markus Himmel established Std as a reliable foundation with clear scope and quality guidelines.

New Language Features

Coinductive Predicates

Syntax identical to inductive definitions, powered by lattice theory.

```
coinductive infSeq (r : α → α → Prop) : α → Prop where
  | step : r a b → infSeq r b → infSeq r a
```

Auto-generated coinduction principles. Supports mutual coinduction.

mvcgen: Monadic Verification

Break down monadic programs into pure verification conditions via weakest precondition.

Std.Do logic + proof mode for discharging verification conditions.

partial_fixpoint

Declare possibly non-terminating functions and verify them

functional_induction

Unfolds functions: fun_induction foo <> grind

Wojciech Rozowski: coinductive predicates on Thursday; **Sebastian Graf:** mvcgen on Tuesday.

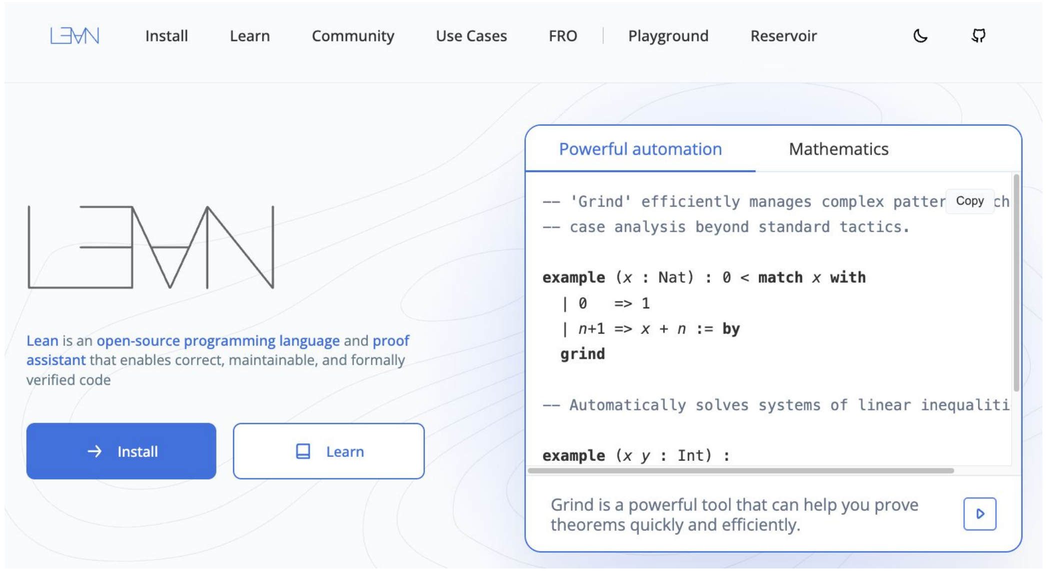
2025 HIGHLIGHTS

Website & Documentation

New Website: lean-lang.org

Completely redesigned website built with Verso, featuring interactive code examples and clear navigation.

Key sections: Install, Learn, Community, Use Cases, FRO, Playground, Reservoir



Lean is an open-source programming language and proof assistant that enables correct, maintainable, and formally verified code

```
-- 'Grind' efficiently manages complex patter ch
-- case analysis beyond standard tactics.

example (x : Nat) : 0 < match x with
  | 0   => 1
  | n+1 => x + n := by
  grind

-- Automatically solves systems of linear inequaliti

example (x y : Int) :
```

Grind is a powerful tool that can help you prove theorems quickly and efficiently.

Lean Language Reference & Verso

Lean Language Reference

Comprehensive coverage: type system, elaboration, tactics, metaprogramming, interactivity, and more

New: "Validating Lean Proofs" section for high-trust applications

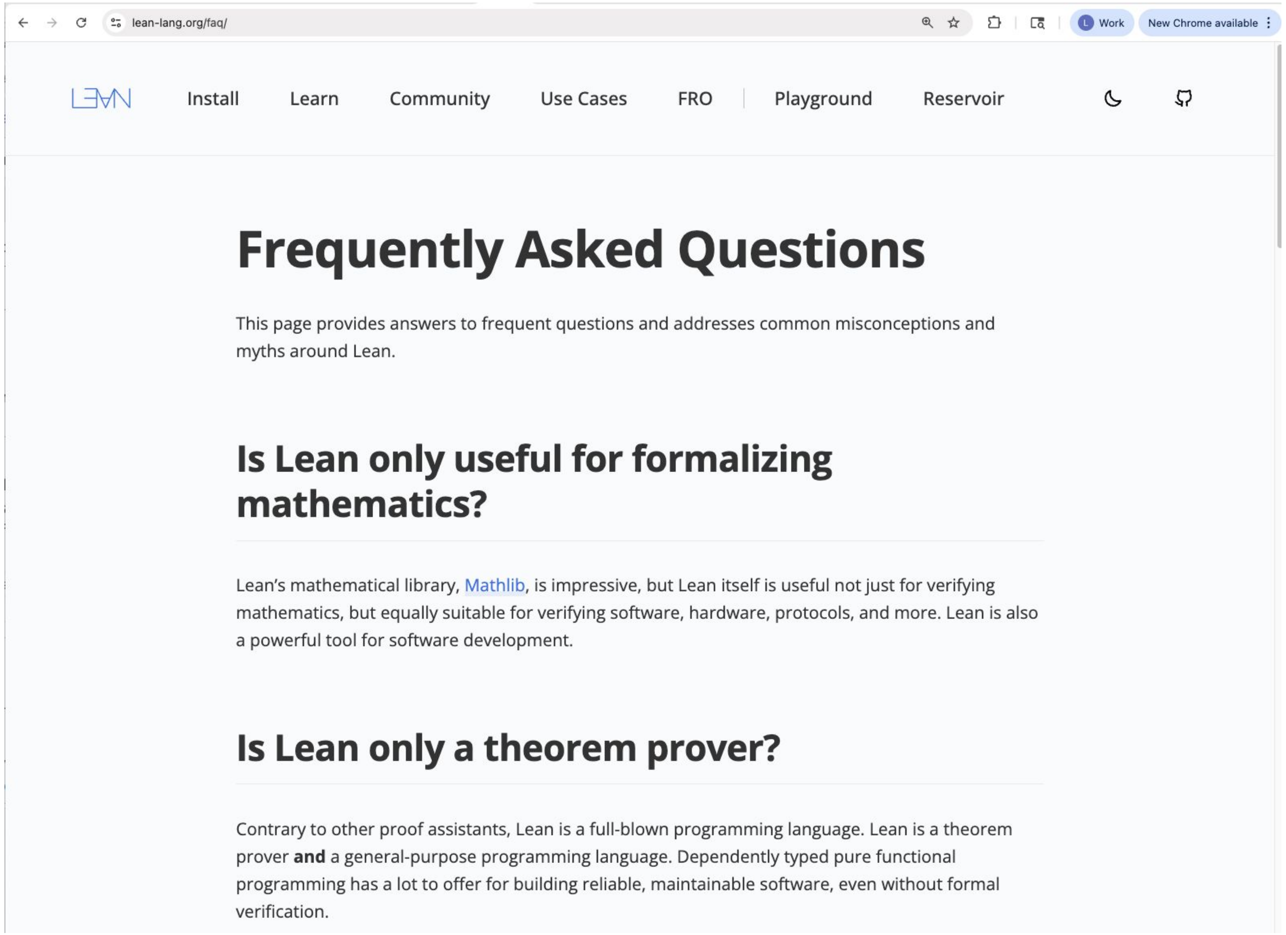
Continuously updated for new features and libraries.

Verso: Literate Programming

Theorem Proving in Lean → Verso

Functional Programming in Lean → Verso

Automatic cross-linking between documents, hovers and proof states in books, red squigglies on errors in docstrings!


lean-lang.org/faq/

Work | New Chrome available

Install | Learn | Community | Use Cases | FRO | Playground | Reservoir

Frequently Asked Questions

This page provides answers to frequent questions and addresses common misconceptions and myths around Lean.

Is Lean only useful for formalizing mathematics?

Lean's mathematical library, Mathlib, is impressive, but Lean itself is useful not just for verifying mathematics, but equally suitable for verifying software, hardware, protocols, and more. Lean is also a powerful tool for software development.

Is Lean only a theorem prover?

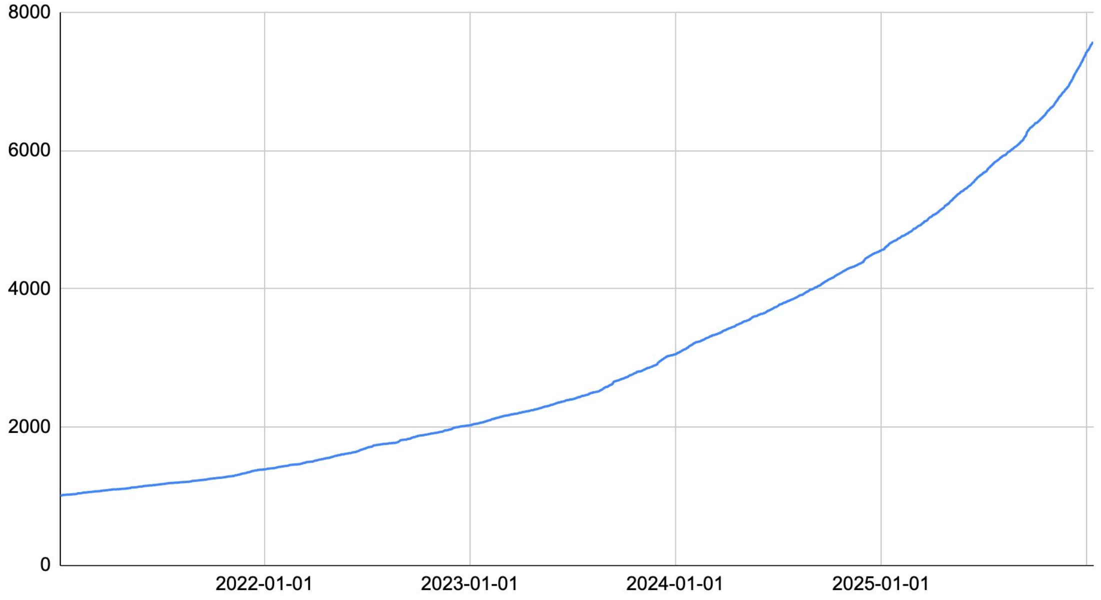
Contrary to other proof assistants, Lean is a full-blown programming language. Lean is a theorem prover **and** a general-purpose programming language. Dependently typed pure functional programming has a lot to offer for building reliable, maintainable software, even without formal verification.

2025 HIGHLIGHTS

Growth & Adoption

Repository Growth

Total Lean repositories (GitHub) 2021 - Present



900+

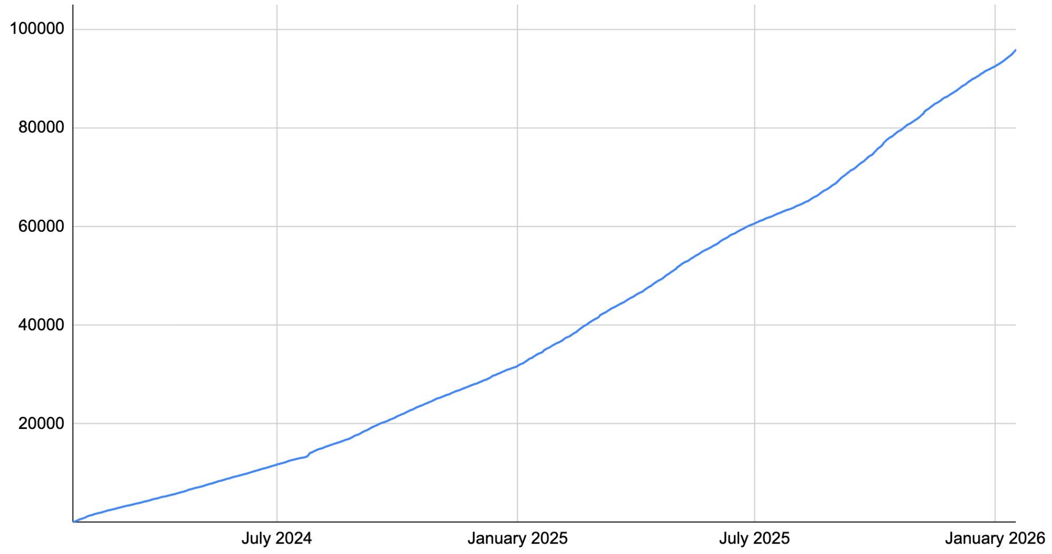
new repos in Q4 2025

+32%

increase over Q3

VS Code Extension Unique Installations

New Lean 4 VS Code extension installs 2024 - Present



90K+

Since beginning of 2024

17K+

new installs Q4 2025

+21%

increase over Q3

Who Uses Lean?

Industry

AWS · Google DeepMind · Meta FAIR · Microsoft · OpenAI

AI Startups

Harmonic · Axiom · Math, Inc. · Mistral · Logical Intelligence · Axiomatic AI

Academia (50+ Courses)

CMU · MIT · Stanford · Imperial · ETH Zürich · Cambridge · LMU · + many more

Lean is the **only proof assistant** appearing in respectable position on RedMonk rankings — competing with general-purpose languages.

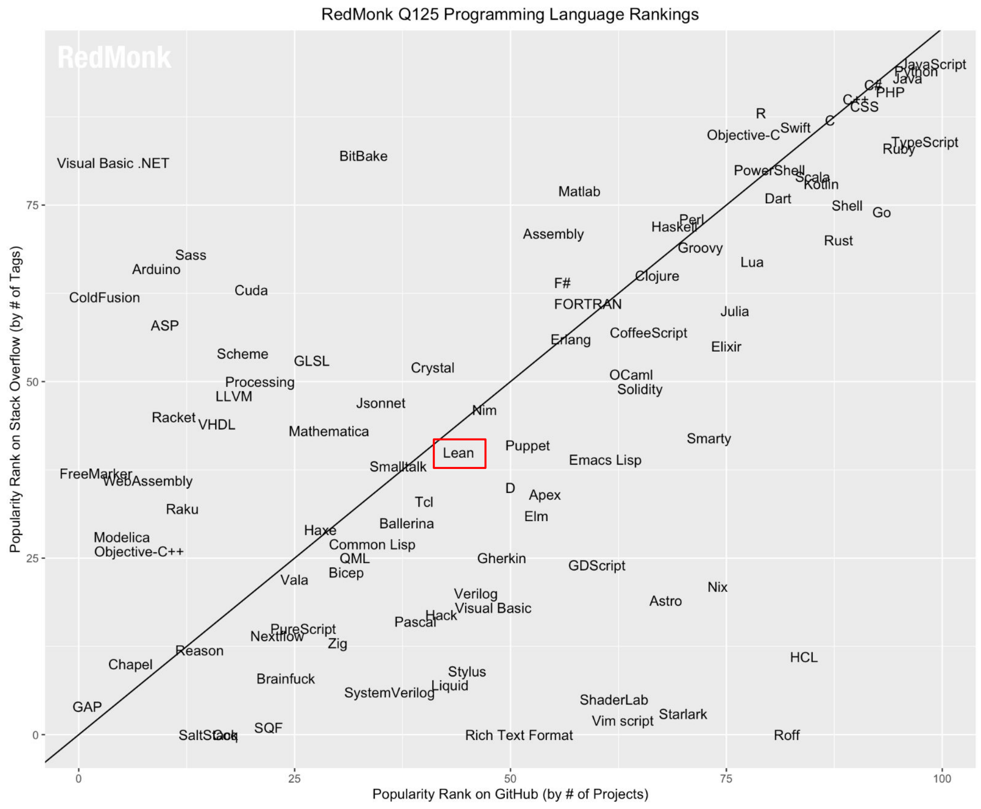
RedMonk 2025 Language Rankings

Lean now appears in the RedMonk Programming Language Rankings — measured by GitHub projects and Stack Overflow activity.

Lean's neighbors:

Mathematica, Nim, Puppet, Smalltalk

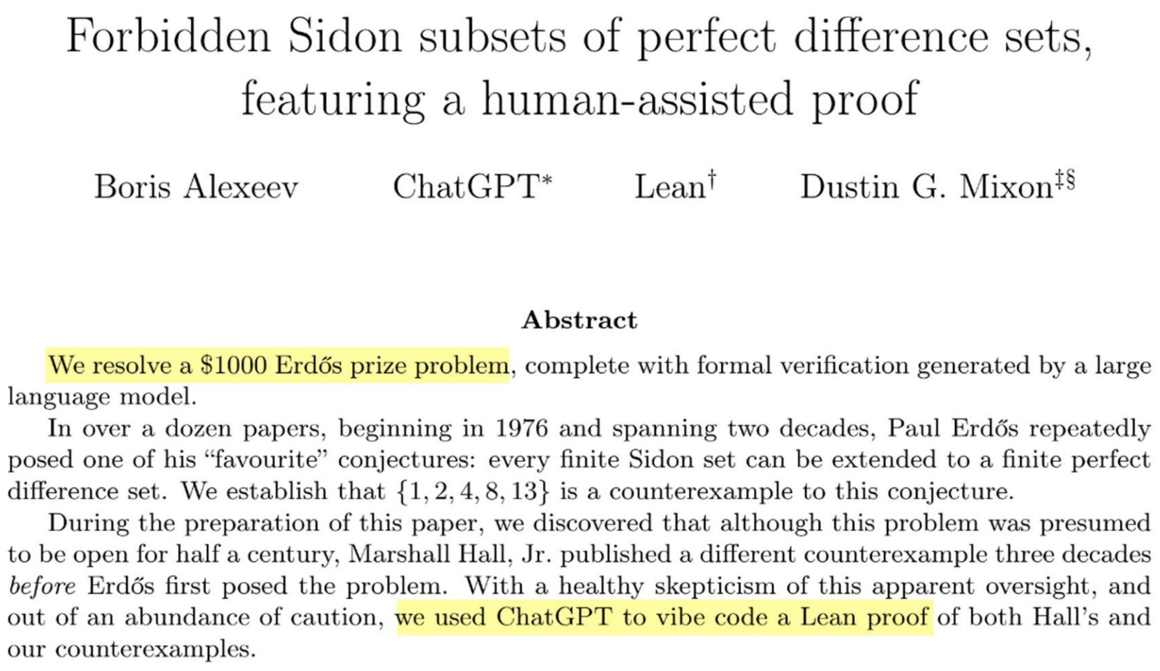
Key insight: Lean is the only proof assistant in a respectable position. Coq is at the very bottom of the chart.



2025 HIGHLIGHTS

AI & Software Verification

Vibe Proving



Forbidden Sidon subsets of perfect difference sets, featuring a human-assisted proof

Boris Alexeev   ChatGPT[*]   Lean[†]   Dustin G. Mixon[‡§]

Abstract

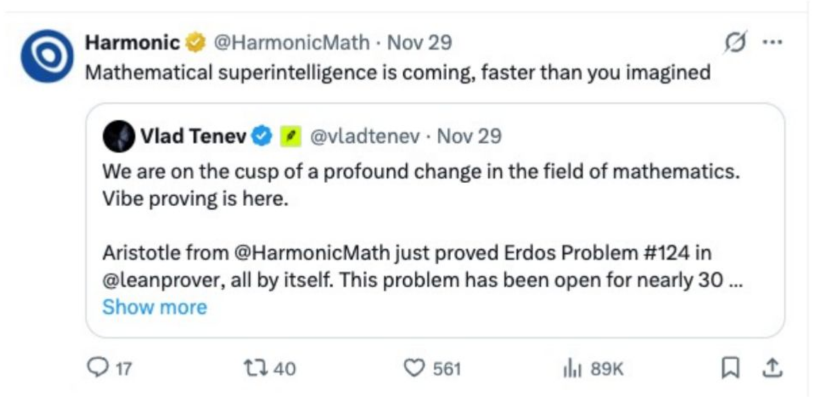
We resolve a $1000 Erdős prize problem, complete with formal verification generated by a large language model.

In over a dozen papers, beginning in 1976 and spanning two decades, Paul Erdős repeatedly posed one of his "favourite" conjectures: every finite Sidon set can be extended to a finite perfect difference set. We establish that $\{1, 2, 4, 8, 13\}$ is a counterexample to this conjecture.

During the preparation of this paper, we discovered that although this problem was presumed to be open for half a century, Marshall Hall, Jr. published a different counterexample three decades *before* Erdős first posed the problem. With a healthy skepticism of this apparent oversight, and out of an abundance of caution, we used ChatGPT to vibe code a Lean proof of both Hall's and our counterexamples.

borisalexeev.com/pdf/erdos707.pdf

Vibe Proving


Harmonic @HarmonicMath · Nov 29

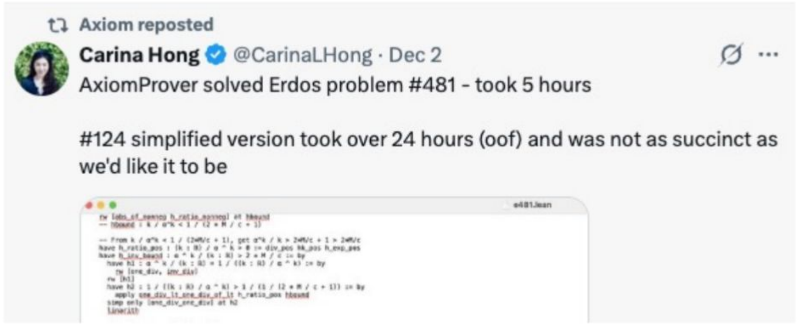
Mathematical superintelligence is coming, faster than you imagined

Vlad Tenev @vladtenev · Nov 29

We are on the cusp of a profound change in the field of mathematics. Vibe proving is here.

Aristotle from @HarmonicMath just proved Erdos Problem #124 in @leanprover, all by itself. This problem has been open for nearly 30 ...
Show more

17 · 40 · 561 · 89K


Axiom reposted

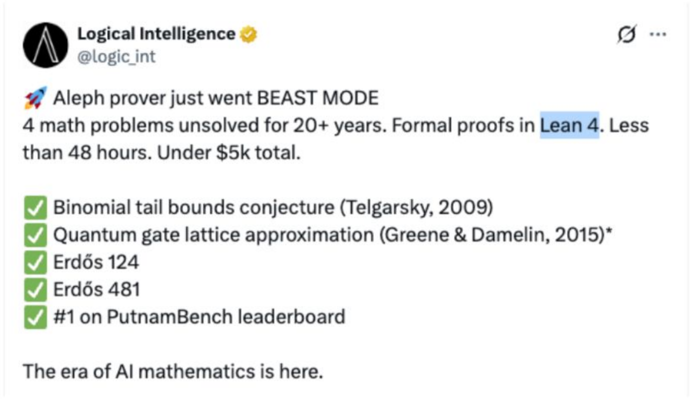
Carina Hong @CarinaLHong · Dec 2

AxiomProver solved Erdos problem #481 - took 5 hours

#124 simplified version took over 24 hours (oof) and was not as succinct as we'd like it to be

e481.lean

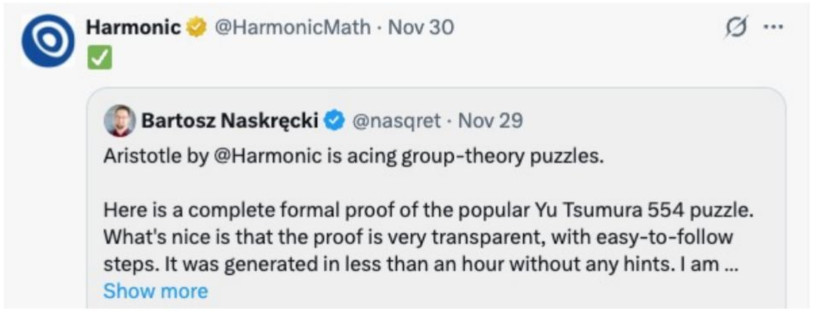
```
rw [abs_of_nonneg h_ratio_nonneg] at hbound
-- hbound : k / a^k < 1 / (2 * M / c + 1)

-- From k / a^k < 1 / (2*M/c + 1), get a^k / k > 2*M/c + 1 > 2*M/c
have h_ratio_pos : (k : ℝ) / a ^ k > 0 := div_pos hk_pos h_exp_pos
have h_inv_bound : a ^ k / (k : ℝ) > 2 * M / c := by
  have h1 : a ^ k / (k : ℝ) = 1 / ((k : ℝ) / a ^ k) := by
    rw [one_div, inv_div]
  rw [h1]
  have h2 : 1 / ((k : ℝ) / a ^ k) > 1 / (1 / (2 * M / c + 1)) := by
    apply one_div_lt_one_div_of_lt h_ratio_pos hbound
  simp only [one_div_one_div] at h2
  linarith
```


Logical Intelligence @logic_int

🚀 Aleph prover just went BEAST MODE
4 math problems unsolved for 20+ years. Formal proofs in Lean 4. Less than 48 hours. Under $5k total.

✅ Binomial tail bounds conjecture (Telgarsky, 2009)
✅ Quantum gate lattice approximation (Greene & Damelin, 2015)*
✅ Erdős 124
✅ Erdős 481
✅ #1 on PutnamBench leaderboard

The era of AI mathematics is here.


Harmonic @HarmonicMath · Nov 30
✅

Bartosz Naskręcki @nasqret · Nov 29

Aristotle by @Harmonic is acing group-theory puzzles.

Here is a complete formal proof of the popular Yu Tsumura 554 puzzle. What's nice is that the proof is very transparent, with easy-to-follow steps. It was generated in less than an hour without any hints. I am ...
Show more

Startups using Lean & AI

Harmonic

Axiom

Math, Inc.

Logical Intelligence

Axiomatic_AI

...and more to come

A thriving startup ecosystem validates Lean as the platform of choice for verified AI reasoning.

CSLib: AI + CS is the Next Frontier

A Focused Effort on Formalizing Computer Science in Lean

Supported by: AWS, Google DeepMind, SDU, Centaur (Stanford)

Formalizing CS Foundations — computational models, complexity

Reasoning about Code — deductive verification techniques

Verified Code Repository — algorithms & data structures

AI Integration — training datasets, AI-assisted tools

cslib.io

Harmonic

"Aristotle achieves **SOTA 96.8%** on VERINA benchmark" — verifiable code generation

Ilya Sergey

"The research community's perception of program verification is about to change **irreversibly**."

Fabrizio Montesi will present CSLib at Lean Together.

Software Verification Ecosystem

Verification Tools Building on Lean

Velvet

Imperative program verifier (Oct 2025)

mvcgen

Lean FRO's monadic verification framework

Aeneas

Rust verification via translation to Lean

Strata

A unified platform for formalizing language syntax and semantics

Why CS + Lean?

AI systems excel at generating code. But can you **trust** that code? Lean provides the missing piece: **proof that the code is correct**.

2025 HIGHLIGHTS

Soundness & Trust

Why Soundness Matters

Lean's value proposition rests on **trust**. If the kernel has bugs, proofs mean nothing.

Critical Use Cases

AI math competitions, verified software, hardware verification, published theorems — all depend on kernel correctness.

Community Expectations

As Lean adoption grows, the community rightfully expects transparency about soundness and clear guidance on trust.

"How do I know I can trust a Lean proof?"

This is a fair question. We take it seriously.

Our commitment: Provide the tools, infrastructure, and documentation for users to verify proofs to their own standards.

Our Approach to Soundness

Small Trusted Kernel

~8K lines of C++ that must be correct. Everything else generates terms checked by this kernel.

Export Format

Lean can export proofs for independent checkers to verify.

Multiple Independent Kernels

If n kernels all accept a proof, the chance of a bug affecting all n is very low.

Independent Kernel Implementations

lean4checker — Reference checker in Lean itself

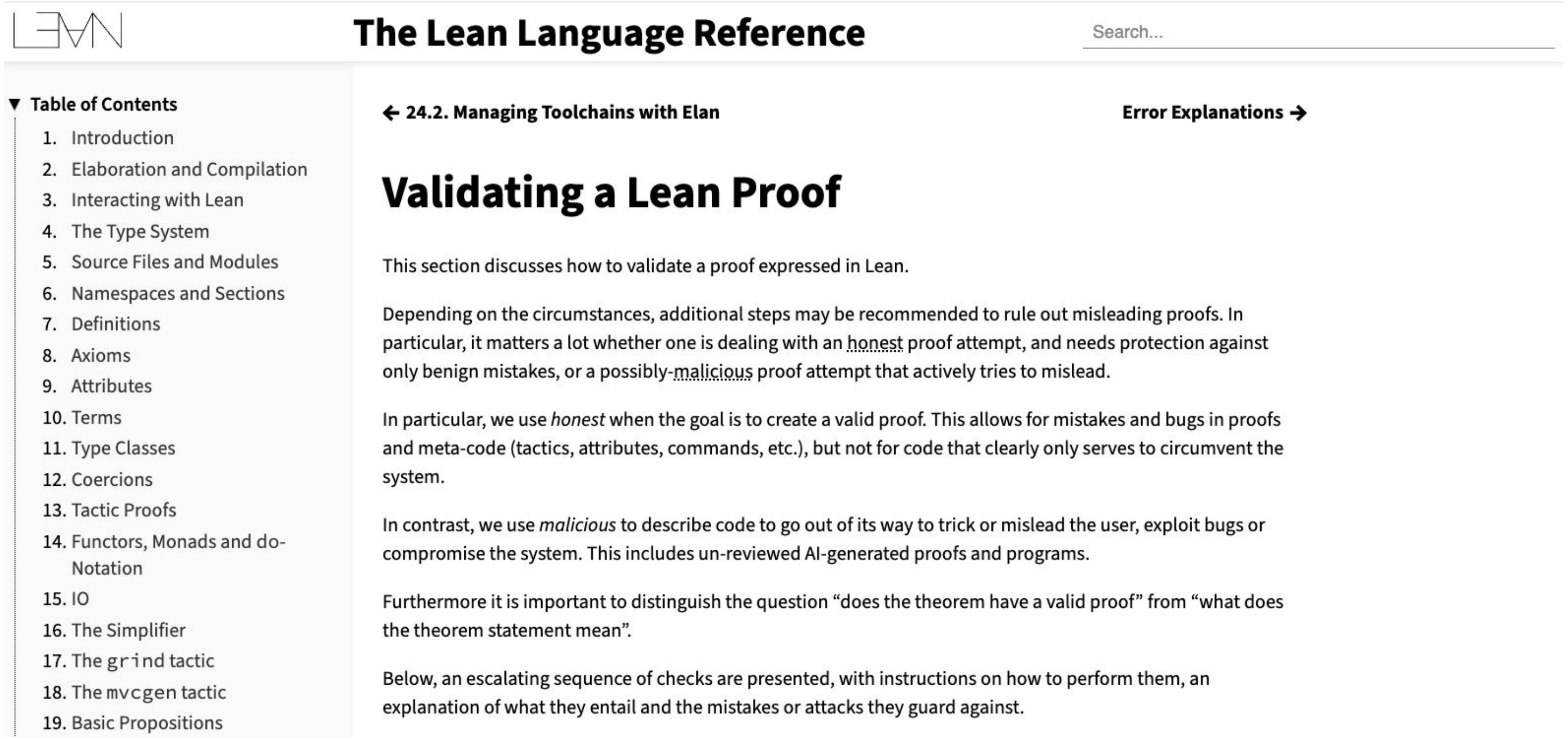
nanoda — Rust implementation

lean4lean — Full kernel in Lean

+ more — Community implementations

Different languages, different authors, different approaches

New documentation



The Lean Language Reference

Search...

← 24.2. Managing Toolchains with Elan   Error Explanations →

Validating a Lean Proof

This section discusses how to validate a proof expressed in Lean.

Depending on the circumstances, additional steps may be recommended to rule out misleading proofs. In particular, it matters a lot whether one is dealing with an honest proof attempt, and needs protection against only benign mistakes, or a possibly-malicious proof attempt that actively tries to mislead.

In particular, we use *honest* when the goal is to create a valid proof. This allows for mistakes and bugs in proofs and meta-code (tactics, attributes, commands, etc.), but not for code that clearly only serves to circumvent the system.

In contrast, we use *malicious* to describe code to go out of its way to trick or mislead the user, exploit bugs or compromise the system. This includes un-reviewed AI-generated proofs and programs.

Furthermore it is important to distinguish the question “does the theorem have a valid proof” from “what does the theorem statement mean”.

Below, an escalating sequence of checks are presented, with instructions on how to perform them, an explanation of what they entail and the mistakes or attacks they guard against.

Comparator: Trustworthy Proof Judge

What is it?

A trustworthy judge for Lean proofs. Verify that a solution proves exactly what was claimed, using only permitted axioms.

Sandboxed Execution

Solutions run in isolated sandbox — no access to modify trusted files

Multi-Kernel Verification

Supports Lean kernel + nanoda for increased trust

How It Works

1. Build Challenge and Solution in sandboxes

2. Export both to kernel-checkable format

3. Verify declarations match exactly

4. Check only permitted axioms used

Use case: AI math competitions, untrusted proof verification

Lean Kernel Arena: arena.lean-lang.org

Lean Kernel Arena

Gamified competition for kernel implementations — **arena.lean-lang.org**

Leaderboard Metrics

Completeness %, Soundness %, Performance, Days since last unsoundness, Lines of Code

Test Suite

Positive tests: All of Mathlib must pass

Negative tests: Collection of tricky exploits and edge cases

Goal: Make writing a Lean kernel a standard exercise in programming language courses

Why a "Competition"?

Incentivizes diverse implementations. More implementations = more eyes on the spec = higher confidence in soundness.

Future: "official-beta" Kernels

New kernel features can be tested on arena participants before merging into the official kernel.

Watch out for an announcement on Zulip.

The Vision: Decentralized Trust

"Don't trust us. Verify."

We don't want you to trust the Lean FRO. We want to give you the tools to trust *yourself*.

Export your proofs. Run them through independent kernels you trust. If they all agree, you have a proof.

No Single Point of Failure

If one kernel has a bug → other kernels catch it

Different authors, languages, approaches

Common interface, common test suite

Result: Trust in the collective, not in any single implementation

2026

Moving Forward

2026: Year 3 Goals

Std 1.0 — Finalize standard library; support Mathlib in reducing technical debt

Proof Automation — grind improvements, scalable simp, try?, +suggestions, counter-example generation

Compiler — Smaller binaries, unboxed types, new backend optimizations

New do-notation — Verification condition generation, async/await

Benchmarking — Reliable profiling infrastructure for Lean and Mathlib — radar.lean-lang.org

Usability — Auto-formatter, clearer errors, IDE polish

Lake — Deploy remote cache to Lean core and Mathlib

Case Studies — Demonstrate Lean's prowess in multiple domains

2026: Next-Gen UX & Literate Programming

Lean Online Workbench

Zero-install browser environment with multi-file projects, full-featured editing, real-time collaboration

Next-gen Lean Games editor — Natural Number Game successor

GitHub integration — Seamless workflow

Verso Blueprints — Relate formal and informal objects in the same file

Verso + doc-gen4

Shared code, auto cross-linking, consistent rendering

Zero-config Websites

"Turn my code into a website" with one command

Auto-formatter

In VS Code extension, scales with Mathlib

GUI Installer

Cross-platform, installs VS Code too

2026: Software Verification

New do-notation

Extensible elaborator framework for intrinsic Dafny-style verification using do notation.

Finer-grained Incrementality & Parallelism

Robust against whitespace changes. Run expensive terminal tactics in parallel.

Call-by-value Tactic

"Proofs by computation" supporting well-founded recursion. Partial evaluation for hardware verification.

SymM

Framework for developing verification condition generators, decision procedures, etc.

Counterexample Generator

When proofs fail, show why with concrete counterexamples.

Goal: Make Lean the best platform for verified software.

2026: Compiler & Backend

Smaller Binaries

Reducing executable size for better distribution and deployment

Unboxed Types

Better memory layout, improved cache performance

New Code Generator

Stack allocation, ownership annotations, more efficient codegen

Finer-grained Incrementality

Robust against whitespace changes after a tactic. Parallelized expensive terminal tactics.

Improved Derived Instances

Scalable noConfusion, better deriving handlers

Goal: Make Lean even faster for production use at scale

Stability & Backward Compatibility

Our commitment to the community

The Journey So Far

July 2023

Lean 4 officially released after Mathlib was fully ported in the same month.

Since then: 25 releases, rapid feature development, growing ecosystem, +7,500 PRs merged.

We hear you: Stability and backward compatibility are essential for growth and adoption.

Why Changes Were Necessary

Some design decisions only reveal problems at scale (e.g., Mathlib)

Better to fix fundamental issues early than accumulate technical debt

Module system, type class resolution, stdlib design needed refinement, etc

What We're Still Improving

Implicit Argument DefEq issues

Current strategy negatively impacts Mathlib performance and usability. This must be fixed — it's a fundamental issue.

Standard Library Completion

Stdlib will be "completed" in 2026 — stable APIs, comprehensive coverage, minimal future breaking changes.

Module System

New module system (Sebastian's presentation Tuesday) is a big deal for compilation and code organization.

These fixes have breaking changes, but they're **necessary** to build a solid foundation for the next decade.

Our Commitment

2026 Goal

2026 will be the **last year** with significant changes to the language and standard library.

After 2026, we shift to a stability-first approach. Breaking changes will be rare and carefully considered.

What We're Doing Now

Release notes: All breaking changes clearly highlighted

Deprecation annotations: Old APIs marked deprecated before removal

Migration guides: Documentation for updating code

We understand stability is essential for adoption. We're working to earn your trust.

Looking Ahead

The Opportunity

AI is transforming how we do mathematics and write software. Lean is uniquely positioned at this intersection — the only tool that's both a serious programming language and a serious proof assistant.

What's Next

AI systems that can discover and verify mathematics

Verified software at scale — not just proofs of concept

A stable, mature platform for the next decade

We're building the infrastructure for trustworthy AI and verified software. Thank you for being part of it.

Get Involved

Contribute New Projects

Math and Software

Write Documentation using Verso

Tutorials, blog posts, teaching materials

Report Issues

Bug reports and feature requests help improve Lean

Join the Community

Zulip: leanprover.zulipchat.com

GitHub: github.com/leanprover

Build Independent Kernels

Participate in the Kernel Arena — help strengthen Lean's trust model

Thank You

https://leanprover.zulipchat.com/
x: @leanprover
LinkedIn: Lean FRO
Mastodon: @leanprover@functional.cafe
#leanlang, #leanprover

https://www.lean-lang.org/

Questions?

lean-lang.org • leanprover.zulipchat.com