

Teaching AI to Configure Proof Automation in Lean

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

December 6, 2025



[Lean](#) is an open-source **theorem prover** and **programming language** that has become central to **AI for mathematics**, powering systems like DeepMind's AlphaProof, Harmonic's Aristotle.

Theorem proving in Lean is an interactive game

The screenshot shows the Lean IDE interface. On the left, the source code for a file named `Odd.lean` is displayed. The code defines a function `odd` and a theorem `square_of_odd_is_odd`. The theorem is currently in a `by` block, indicating it is being proved interactively. On the right, the `Lean Infoview` panel shows the current tactic state. The state is `1 goal` with the goal `odd n → odd (n * n)`. A blue box with the text "The 'game board'" has an arrow pointing to the goal in the tactic state.

```
Odd.lean > square_of_odd_is_odd
1 import Mathlib
2
3 def odd (n : ℕ) := ∃ k, n = 2 * k + 1
4
5 -- Prove that the square of an odd number is always odd
6 theorem square_of_odd_is_odd : odd n → odd (n * n) := by
7   done
8
9
10
11
12
```

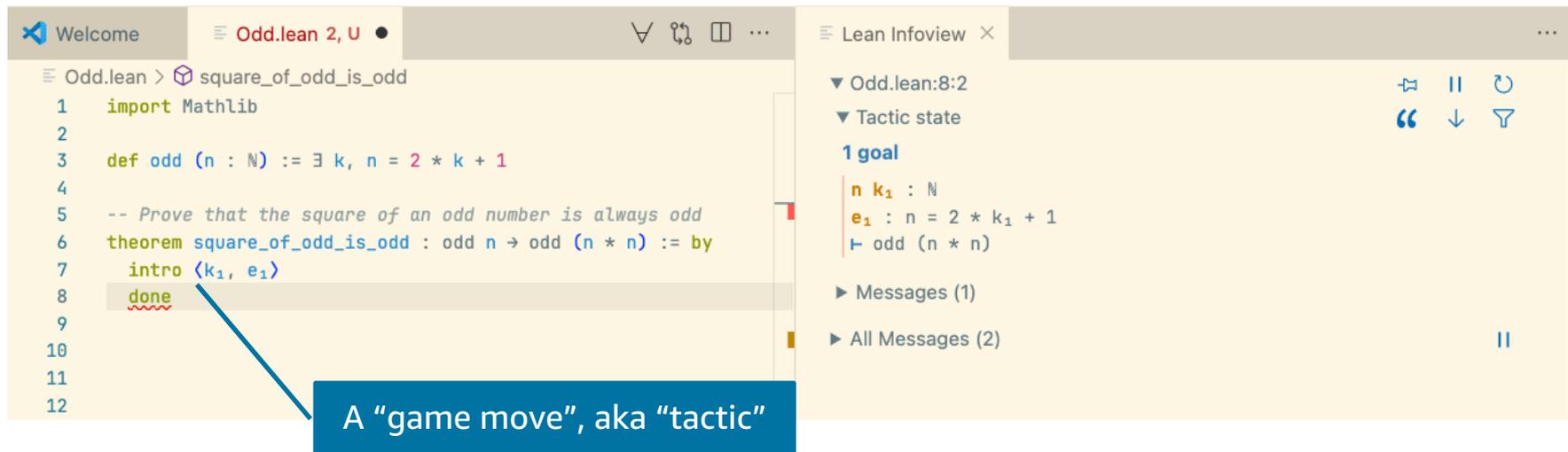
Lean Infoview

- Odd.lean:7:2
- Tactic state
- 1 goal
- `n : ℕ`
- `⊢ odd n → odd (n * n)`
- Messages (1)
- All Messages (2)

The "game board"

"You have written my favorite computer game", Kevin Buzzard

Theorem proving in Lean is an interactive game



```
Odd.lean > square_of_odd_is_odd
1  import Mathlib
2
3  def odd (n : ℕ) := ∃ k, n = 2 * k + 1
4
5  -- Prove that the square of an odd number is always odd
6  theorem square_of_odd_is_odd : odd n → odd (n * n) := by
7    intro ⟨k1, e1⟩
8    done
9
10
11
12
```

▼ Odd.lean:8:2

▼ Tactic state

1 goal

n k_1 : \mathbb{N}

e_1 : $n = 2 * k_1 + 1$

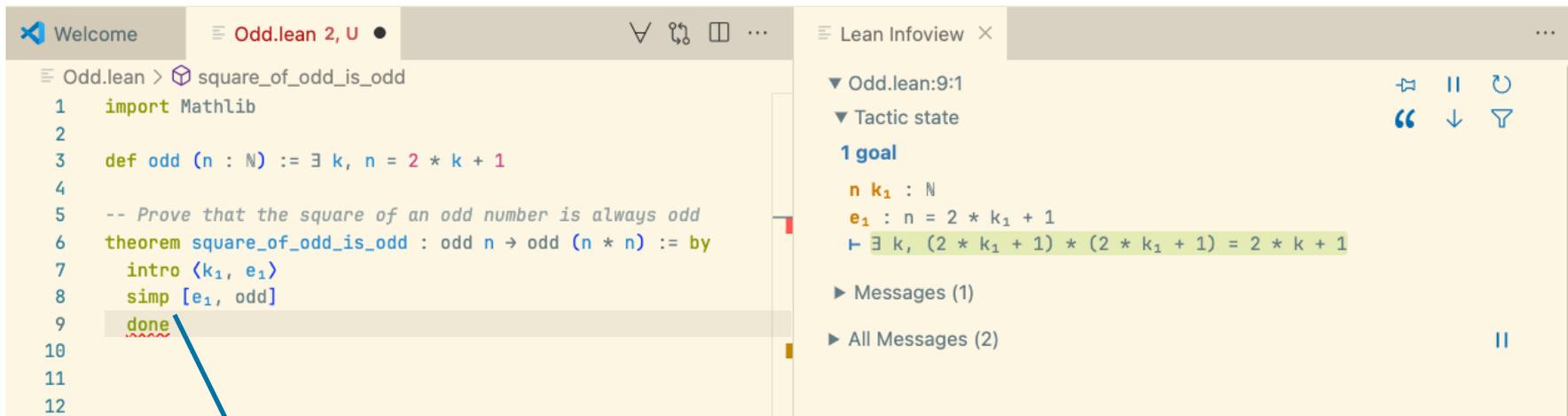
┆ odd (n * n)

► Messages (1)

► All Messages (2)

A "game move", aka "tactic"

Theorem proving in Lean is an interactive game



The screenshot shows the Lean IDE interface. On the left, a code editor displays a Lean script for proving that the square of an odd number is odd. The script includes an import, a definition of an odd number, a theorem statement, and a proof using the `simp` tactic. A blue arrow points from the `simp` line to a callout box. On the right, the 'Lean Infoview' panel shows the current tactic state, including the goal and the hypotheses used by the `simp` tactic.

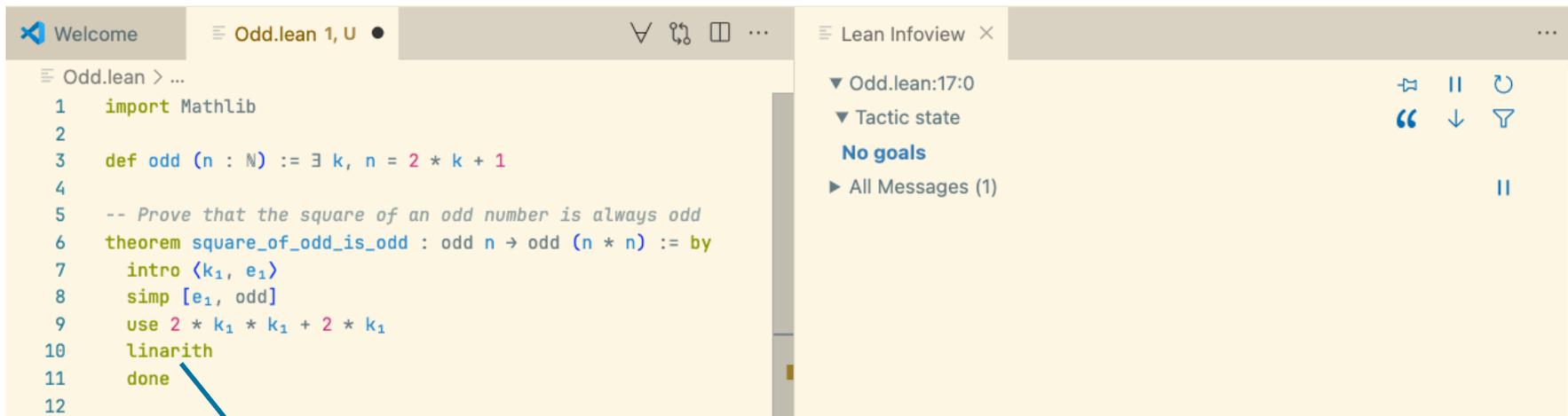
```
Odd.lean > square_of_odd_is_odd
1  import Mathlib
2
3  def odd (n : ℕ) := ∃ k, n = 2 * k + 1
4
5  -- Prove that the square of an odd number is always odd
6  theorem square_of_odd_is_odd : odd n → odd (n * n) := by
7    intro (k1, e1)
8    simp [e1, odd]
9    done
10
11
12
```

Lean Infoview

- Odd.lean:9:1
- Tactic state
- 1 goal
- n k₁ : ℕ
- e₁ : n = 2 * k₁ + 1
- ┆ ∃ k, (2 * k₁ + 1) * (2 * k₁ + 1) = 2 * k + 1
- Messages (1)
- All Messages (2)

The “game move” `simp`, the simplifier, is one of the most popular moves in our game

Theorem proving in Lean is an interactive game



```
1 import Mathlib
2
3 def odd (n : ℕ) := ∃ k, n = 2 * k + 1
4
5 -- Prove that the square of an odd number is always odd
6 theorem square_of_odd_is_odd : odd n → odd (n * n) := by
7   intro (k₁, e₁)
8   simp [e₁, odd]
9   use 2 * k₁ * k₁ + 2 * k₁
10  linarith
11  done
12
```

Lean Infview ×

- Odd.lean:17:0
- Tactic state
- No goals
- All Messages (1)

We complete this level using `linarith`, the linear arithmetic, move



Challenge

AI systems are writing impressive Lean proofs, but **they often bypass powerful automation** tactics, **generating long manual proofs that a single tactic call could replace.**

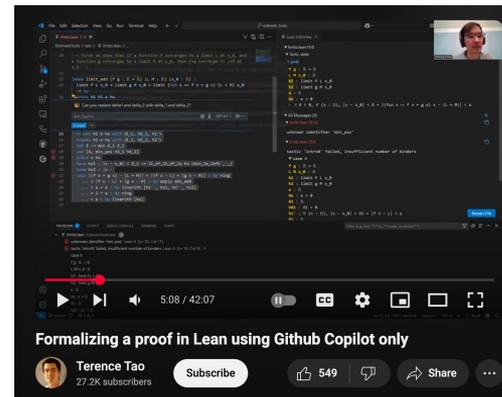
- **The bottleneck is annotations:** metadata that tells **symbolic automation** how to use theorems.
- Most mathematicians can write proofs but can't write good annotations.
- **We propose training AI to predict annotations.**

Lean is Taking Mathematics by Storm

*"Lean enables large-scale collaboration by allowing mathematicians to break down complex proofs into smaller, verifiable components. This formalization process ensures the correctness of proofs and facilitates contributions from a broader community. **With Lean, we are beginning to see how AI can accelerate the formalization of mathematics, opening up new possibilities for research.**" — Terence Tao*

Fermat's Last Theorem – Kevin Buzzard

Carleson's Theorem – Floris van Doorn



Lean is Taking Mathematics by Storm

17 out of 29 AI for Math Fund winners reference Lean

Renaissance Philanthropy and XTX Markets
Launch New \$9 million AI for Math Fund



2025 Winners

An AI-Focused Tactic for Language Learning Explore project	A Dataset of Modern Formalized Theorem Statements Explore project	A Principled Approach to Proof Search with Applications to Siderenko's Conjecture Explore project	A Structured Representation of Tactics for Machine-Assisted Theorem Proving Explore project
Bridging AI Proof Assistants, and Mathematical Data (BRIDGE) Explore project	Bridging Complexity and Automation to Advance Automatic Theorem Proving Explore project	Bridging Proof and Computation Explore project	Constraining LLMs for Theorem Proving Explore project
Copilots for Isabelle Explore project	Crowdsourcing and Reinventing the Next Generation of Dynamic and Scalable Math Benchmarks Explore project	Databases of Structured Motivated Proofs Explore project	DEEPER Explore project
Document Level Autoformalization	Domain Specific Documentation for	Game Over or QED?	GNN-SMT

renaissancephilanthropy.org/ai-for-math-fund-projects

Move Over, Mathematicians, Here Comes AlphaProof

A.I. is getting good at math — and might soon make a worthy collaborator for humans.

Share full article 47



Ring the gong at Google Deepmind's London headquarters, a ritual to celebrate each A.I. milestone, including its recent triumph of reasoning at the International Mathematical Olympiad. Google Deepmind

IMO 2025: 3 Gold, 1 Silver*

OpenAI (informal)



Alexander Wei ✓
@alexwei_



1/N I'm excited to share that our latest @OpenAI experimental reasoning LLM has achieved a longstanding grand challenge in AI: gold medal-level performance on the world's most prestigious math competition—the International Math Olympiad (IMO).

Harmonic (Lean)

🔗 [Aristotle Achieves Gold Medal-Level Performance at the International Mathematical Olympiad, iOS App Beta Launch](#)

DeepMind (informal)

Advanced version of Gemini with Deep Think officially achieves gold-medal standard at the International Mathematical Olympiad

ByteDance (Lean)

ByteDance Seed Prover Achieves Silver Medal Score in IMO 2025

IMO-Level Proof Search as a Service

Aristotle Lean4 API

Mathematical Superintelligence, Formally Verified

Get access now

```

Ask follow-up...
+  📄 Math Input
• Output: 7/6s
have h_alg_int : IsIntegral (2 * Real.cos θ) = by
  - By Niven's theorem,  $2 \cos(\theta)$  is an algebraic integer.
have h_alg_int : IsIntegral (Complex.exp (θ * Complex.I) + Complex.exp (-θ * Complex.I)) =
by
  - Since  $\theta$  is a rational multiple of  $2\pi$ ,  $e^{i\theta}$  is a root of unity and hence an
algebraic integer.
obtain ⟨hr, r, θ⟩ : Real.pi >= θ
have h_root_of_unity : n > 0 (Complex.exp (θ * Complex.I))^n = 1 => hr
  - Since  $\theta$  is rational, we can write  $\theta = 2\pi \cdot \frac{p}{q}$  with  $\gcd(p, q) = 1$ 

```

AlphaProof Interest Form

Thanks for your interest in AlphaProof. To deliver a reliable user experience, we are carefully managing our compute resources to maintain system stability. We will be reviewing applications and prioritizing access over time as we scale the system. We appreciate your patience and understanding.

To help you get the most from this process, we encourage you to first explore the [AlphaProof Blog Post](#), the [AlphaProof Nature Paper](#), and the [AlphaProof Tool Demos \[1\]\[2\]](#). Understanding AlphaProof beforehand will empower you to complete this form more effectively and confidently.

deepmind.google/alphaproof

Highly Active, Healthy Competition on Putnam Bench

#	Model	num-solved	compute
1	Aleph Prover	500	pass@1, avg 1834 tool calls
2	Hilbert	462	avg pass@1840
3	Seed-Prover	329	MEDIUM
4	Ax-Prover 	91	pass@1, avg. 100 tool calls
5	Goedel-Prover-V2 	86	pass@184
6	DeepSeek-Prover-V2 	47	pass@1024
7	GPT-5 (ReAct, 10 turns)	28	pass@1, 10 tool calls
8	DSP+ 	23	pass@128
9	Bourbaki 	14	pass@512
10	Kimina-Prover-7B-Distill 	10	pass@192
11	Self-play Theorem Prover 	8	pass@3200
12	Goedel-Prover-SFT 	7	pass@512

UC San Diego/Apple, Sep 2s025

ByteDance, Jul 2025

Axiomatic_AI et al, Oct 2025

Princeton/NVIDIA et al, Aug 2025

DeepSeek-AI, Jul 2025

Microsoft Research et al, Jun 2025

Huawei et al, Jul 2025

Numina/Kimi, Apr 2025

Stanford, Mar 2025

Princeton et al, Apr 2025

trishullab.github.io/PutnamBench/leaderboard.html

 fully open-sourced

 partially open-sourced

Startups using Lean & AI



Math, Inc.



Logical Intelligence



...and more to come



Extensibility and Introspection

"At Google DeepMind, we used Lean to build AlphaProof, a new reinforcement-learning based system for formal math reasoning. **Lean's extensibility and verification capabilities were key in enabling the development of AlphaProof.**" — Pushmeet Kohli, Vice President, Research Google DeepMind

Lean 4 is implemented in Lean, allowing for unprecedented extension and introspection by users.

leanprover-community/repl is a simple Lean program for communicating with the Lean compiler via JSON.

AI labs have customized it in many ways, especially around parallelized proof tree search.

stanford-centaur/PyPantograph is the state of the art for open-source Lean REPLs.



Auto-formalization

Auguste Poiroux

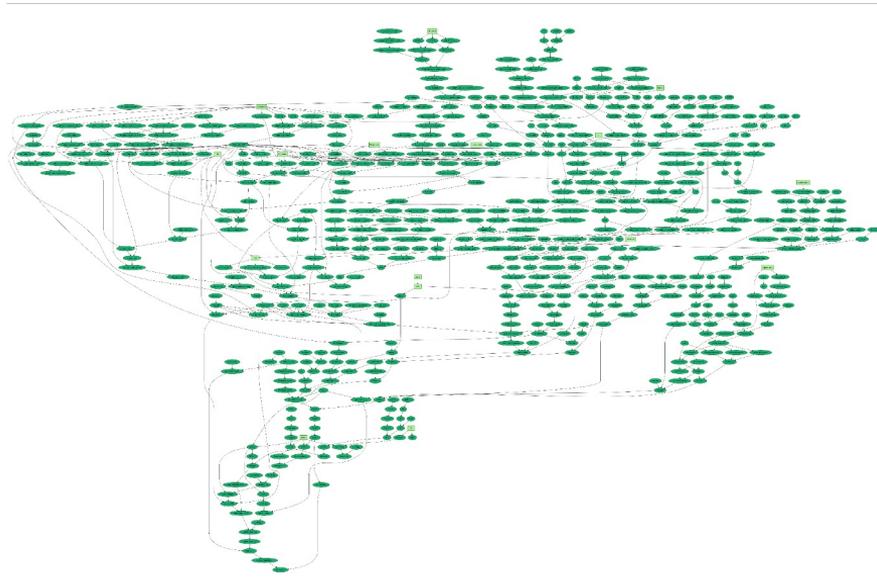
Dear all,

We, at Math, Inc., are excited to introduce [Gauss](#), a new autoformalization agent designed to assist mathematicians in their Lean formalization.

Using Gauss, and building over the recent [medium PNT progress](#) by Kontorovich et al., we formalized the **strong Prime Number Theorem** in Lean. Over the course of three weeks, Gauss produced ~25k lines of Lean code and over 1,000 theorems and definitions. To put these numbers in perspective, our previous iteration, three months ago, produced ~3.5k lines of Lean code and 100 theorems and definitions to formalize the [abc conjecture almost always](#).

Our goal is to make Gauss accessible and easy to use. If you are interested, you can register for early access to Gauss [here](#). We are happy to discuss about Gauss and Math, Inc. in the discussion thread.

github.com/math-inc/strongpnt





Auto-informalization

```

have h := zetacnot0 t
exact norm_pos_iff.mpr h

lemma Zeta1_Zeta_Expand :
  ... ∃ A > 1, ∃ b > 1,
  ... ∀ (t : ℝ) (ht : |t| > 2)
  ... (r1 · r · R1 · R : ℝ)
  ... (hr1_pos : 0 < r1) (hr1_lt_r : r1 < r)
  ... (hr_pos : 0 < r) (hr_lt_R1 : r < R1) (hr1_pos : 0 < R1) (hr1_lt_
  ... let c := (3/2 : ℝ) + I * t;
  ... ∀ (hfin : (zerosetKfRc R1 · c · riemannZeta).Finite),
  ... ∀ z ∈ closedBall c r1 \ zerosetKfRc R1 · c · riemannZeta,
  ... ||logDerivZeta z - ∑ p ∈ hfin.toFinset,
  ... ((analyticOrderAt riemannZeta p).toNat : ℝ) / (z - p)|| ≤
  ... (16 * r^2 / ((r - r1)^3) +
  ... 1 / ((R^2 / R1 - R1) * Real.log (R / R1))) * (Real.log |t| + Rea
  -- Apply the three lemmas mentioned in the informal proof
obtain ⟨b, hbgt1, hb⟩ := zeta32upper
obtain ⟨A, hAgt1, hA⟩ := zeta32lower_log

-- Provide the constants A, b as required
refine ⟨A, hAgt1, b, hbgt1, ?_⟩
intro t ht r1 r R1 R hr1_pos hr1_lt_r hr_pos hr_lt_R1 hr1_pos hr1_
intro c hfin z hz

-- Apply log_Deriv_Expansion_Zeta
have hexp_lemma := log_Deriv_Expansion_Zeta t ht r1 r R1 R hr1_pos

-- Set B = b * |t| as mentioned in informal proof
have htpos : (0 : ℝ) < |t| := by linarith [ht]
have hbgt1 : b * |t| > 1 := by
have hb_pos : (0 : ℝ) < b := by linarith [hbgt1]

```

Mathematical significance:

This is likely part of a proof related to the Prime Number Theorem, where you need precise control over how well the logarithmic derivative of the zeta function can be approximated by summing over its zeros. The bound includes both geometric terms (involving the radii) and growth terms (logarithmic in $|t|$).

The proof strategy mentioned in the comments indicates it combines results from `zeta32upper` and `zeta32lower_log` lemmas to establish the required constants and bounds.

What is the definition of the logarithmic derivative here?

PNT3_RiemannZeta.lean:4428-4440

I'll search for the definition of `logDerivZeta` in this file to show you how the logarithmic derivative is defined here.

Searched text for `logDerivZeta` (`/**/StrongPNT/PNT3_RiemannZeta.lean`), 20 results

Perfect! I found the definition. Let me read the context around line 3495 to give you the complete picture:

Read `PNT3_RiemannZeta.lean`, lines 3490 to 3500



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.

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 📌 ⬆

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.

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

```

-- find_of_lemma h_481a_proved at 1894d
-- lemma : k / a^n = 1 / 12 * M / c + 11
-- From h / a^n = 1 / 12 * M / c + 11, get a^n / k = 240c / (12 * M - 11 * c)
have h_481a_proved : a^n = k / (k + 1) > 2 * M / c := by
  have h2 : 2 * M / c > k / (k + 1) := by
    rw [h_481a_proved]
    have h3 : 1 / (k + 1) > 1 / (k + 1) := by
      apply (lt_of_lt_of_le (h2) (le_of_lt (h3)))
      simp only [h2, h3] at h2
    done
  
```

Harmonic @HarmonicMath · Nov 30 🔗 ...

Bartosz Naskręcki @nasqret · Nov 29

Aristotle by @Harmonic 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](#)



Autonomous Exploration

MATHEMATICAL EXPLORATION AND DISCOVERY AT SCALE

BOGDAN GEORGIEV, JAVIER GÓMEZ-SERRANO, TERENCE TAO, AND ADAM ZSOLT WAGNER

ABSTRACT. AlphaEvo1ve [223] is a generic evolutionary coding agent that combines the generative capabilities of LLMs with automated evaluation in an iterative evolutionary framework that proposes, tests, and refines algorithmic solutions to challenging scientific and practical problems. In this paper we showcase AlphaEvo1ve as a tool for autonomously discovering novel mathematical constructions and advancing our understanding of long-standing open problems.

1.5. Building a pipeline of several AI tools. Even more strikingly, for the finite field Kakeya problem (cf. Problem 6.1), AlphaEvo1ve discovered an interesting general construction. When we fed this programmatic solution to the agent called Deep Think [148], it successfully derived a proof of its correctness and a closed-form formula for its size. This proof was then fully formalized in the Lean proof assistant using another AI tool, AlphaProof [147]. This workflow, combining pattern discovery (AlphaEvo1ve), symbolic proof generation (Deep Think), and formal verification (AlphaProof), serves as a concrete example of how specialized AI systems can be integrated. It suggests a future potential methodology where a combination of AI tools can assist in the process of moving from an empirically observed pattern (suggested by the model) to a formally verified mathematical result, fully automated or semi-automated.

New result distribution

Visualization of results across 67 problems.

● New result ● Former new result, got improved upon ● Worse than literature bound
● Matched known optimal bound ● Matched literature bound / N/A



Autonomous Exploration

```

83 -- Inside this block, when we write `(K1 p)`, Lean uses the `p` from the context.
84 lemma card_K1 (hp_odd : p > 2) : (4 * (K1 p).card : ℤ) = p^3 + 2*p^2 + p := by
85   have := (Fact.out : (p.Prime)).odd_of_ne_two fun and' =>by simp_all
86   rcases this with ⟨x, hx⟩
87   subst hx
88   delta K1
89   rw_mod_cast[Finset.card_filter]
90   trans(4)*∑S:ℤMod (2*x+1) ×_ ×_,ite (IsSquare (S.1^2+4* S.2.1) ∧IsSquare (S.1^2+4* S.2.2)) (1) 0
91   · exact (congr_arg _) (Fintype.sum_equiv ⟨ fun and=> (and 0, and (1), and 2), fun and=>![ _,_, _]
92   · trans(4)*∑S:ℤMod (2*x+1),(( Finset.univ.image fun and=> (and* and-S^2 :) /4) ×^s.image (@ fun and
93   · rw[ Fintype.sum_prod_type, Fintype.sum_congr _ _ fun and=>(Finset.card_filter _ _).symm.trans
94   field_simp[IsSquare, sub_eq_iff_eq_add'.trans (comm),show (4 :ℤMod (2*x + 1))#0 from(( ℤMod.
95   · push_cast[sq, add_assoc,.,., Finset.card_product, true, Finset.mul_sum]at *
96   trans∑S:ℤMod (2*x+1), 4*(( Finset.univ.filter fun and=>and.val≤and.val).image fun p=>(p*p-S
97   · push_cast only [le_refl, true, Finset.filter_true_of_mem, implies_true,sq]
98   · trans∑S:ℤMod (2*x+1), 4*(( Finset.range (x + 1)).image fun p : ℕ=>(p*p-S* S:ℤMod (2*x + 1)
99   · refine Fintype.sum_congr _ _ fun and=>congr_arg _ ((congr_arg (.^2) (congr_arg _ (Finset.
100   use Finset.mem_image.2 (if h: a.val≤x then⟨ _, Finset.mem_range_succ_iff.2 h,by simp_aritl
101   · rewrite[ Finset.sum_congr rfl fun and x =>(congr_arg _) ((congr_arg) ( .^2) ( Finset.card.
102   · simp_all[show (2*x+1)*(4*(x+1)^2) = (2*x+1)^3+ (2 *((2*x+1)*(2*x + 1))+ (2*x + 1))by rii
103   · field_simp[show (4 :ℤMod (2 *by bound + 1))#0 from((ℤMod.isUnit_iff_coprime _ _).2 (Odd
104   use (by valid:).elim (by valid ◦ (ℤMod.val_cast_of_lt (by valid)).symm.trans.comp (. ▶ ℤ

```

A vibrant community of users at leanprover.zulipchat.com

Machine Learning for Theorem Proving Machine Learning and AI or theorem proving. HOList, AI for...

Standard view

- MCP Tools for LLMs and Agentic Mathematics
- Building an Autoformalizer on Analysis
- ✓ Executing Conv and Calc in Pantograph
- Dataset to rule them all?
- REPL: automated incremental state reuse across commands
- Better way for tracing tactic states
- Blind Speculation about IMO 2025
- Claude 4 agent in VS Code
- DeepMind and Navier Stokes
- LeanTool feature: Sorry Hammer
- AI tutorial for LEAN beginner
- Proof or Bluff
- Autoformalization of the probabilistic abc-conjecture
- Illusion of Thinking
- Tool for Lean code verify
- Gemini 2.5 Pro 06/05
- Current state of AI for Mathematics and what could come n...
- System card: Claude Opus 4 & Claude Sonnet 4
- DeepSeek-Prover V2
- I was pleasantly surprised by DeepSeek

Prospective: AI+CS as the Next Frontier is Imminent

[leanprover/cslib](#) founded, supported by AWS, Google DeepMind, SDU, Centaur (Stanford)

Companies are starting to pivot their math-proven AIs towards program verification

► Beyond Math

While the IMO benchmark highlights Aristotle's mathematical ability, the implications reach much further. The same architecture that can solve Olympiad problems also supports:

- Code generation that doesn't require human verification
- Engineering tasks where precision is mission-critical
- Scientific research where progress depends on rigorous mathematical understanding

[harmonic.fun/news#blog-post-aristotle-tech-report](#)

Spec auto-formalization + program synthesis + verification as a service



Ilya Sergey
@ilyasergey



The proof of the last remaining Lean theorem in our upcoming conference submission has now been completed with the help of AI.

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

8:32 PM · Nov 13, 2025 · **15.8K** Views

Prospective: AI+CS as the Next Frontier is Imminent



Beyond math: Aristotle achieves SOTA 96.8% proof generation on VERINA: Benchmarking Verifiable Code Generation. You can read more about this performance on our engineering blog linked in bio

12:24 PM · Dec 3, 2025 · 1,860 Views



Maintained by software engineers



- src
- common.rs
- ffi.rs
- hash.rs
- key.rs
- lib.rs
- mlkem.rs
- ntt.rs
- test.rs

Aeneas
Automatically generated

Code regenerated and proofs replayed in CI



- Code
 - Funs.lean
 - FunsExternal_Template.lean
 - FunsExternal.lean
 - Types.lean
- Properties
 - BarrettReduction.lean
 - Basic.lean
 - CompressEncode.lean
 - CompressEncodeSpecAux.lean
 - MontReduction.lean
 - NormMod.lean
 - Ntt.lean
 - NttSpecAux.lean
 - Polynomials.lean

Pure, functional model

Prove properties about

Maintained by proof engineers



Teaching AI to Configure Proof Automation in Lean



NeuroSymbolic AI

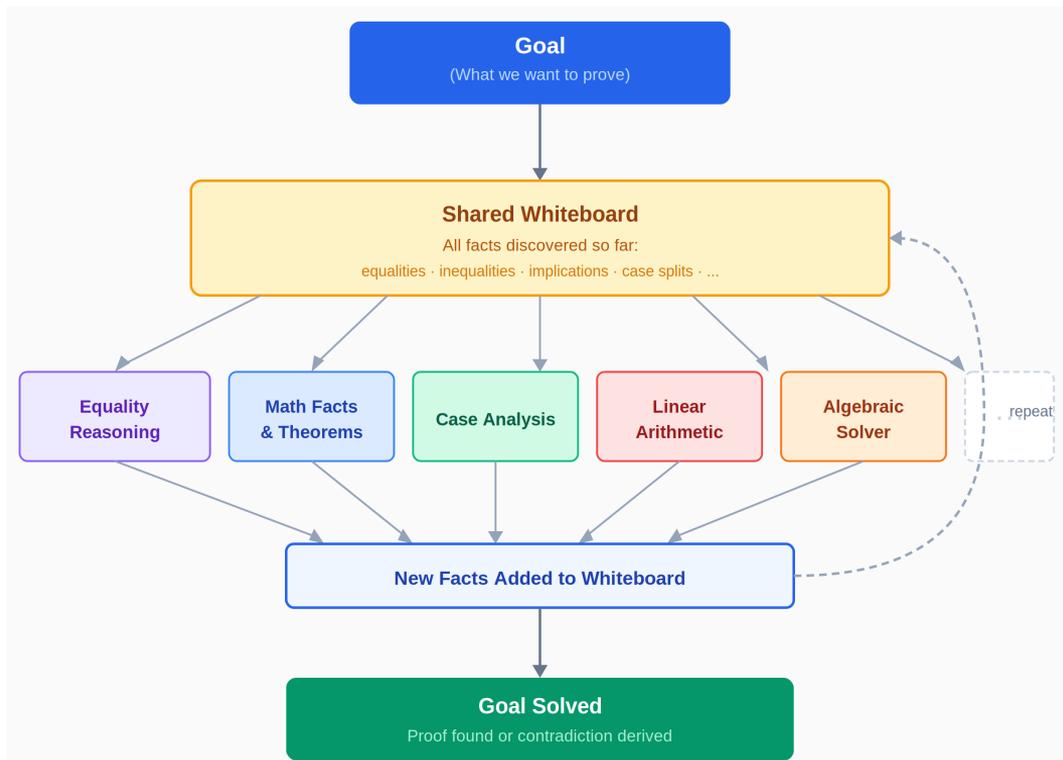
The future of AI for mathematics is not pure neural or pure symbolic. It's the combination.

Symbolic engines provide exhaustive search in structured domains, guaranteed correctness, and decision procedures refined over decades.

AI provides pattern recognition, heuristic guidance, and retrieval.

grind is the symbolic in Neurosymbolic AI

New proof automation (Lean v4.22 – released August 2025) developed by Kim Morrison and myself.



Automating Quantum Algebra

```
example {α} [CommRing α] [IsCharP α 0] (d t c : α) (d_inv PS03_inv : α)
  (Δ40 : d^2 * (d + t - d * t - 2) * (d + t + d * t) = 0)
  (Δ41 : -d^4 * (d + t - d * t - 2) *
    (2 * d + 2 * d * t - 4 * d * t^2 + 2 * d * t^4 + 2 * d^2 * t^4 - c * (d + t + d * t)) = 0)
  : d * d_inv = 1 →
    (d + t - d * t - 2) * PS03_inv = 1 →
      t^2 = t + 1 := by
grind
```

This is not a toy: it encodes a real algebraic constraint derived from relations among diagrams in a pivotal tensor category.

grind can handle this kind of reasoning automatically, in milliseconds.

Generate proof term is massive, more than 20,000 terms.



grind: Initial reactions



Markus de Medeiros Jul 22nd at 1:43 PM

I keep being surprised by how many nuisance goals

`grind` is able to solve. Props to everyone who worked on it!



You and Kim Morrison, Oliver Nash

AUG 8



Oliver Nash EDITED

8:27 AM

I was just singing `grind`'s praises in the Mathlib community meeting and highlighted my favourite example was [#27372](#) (which massively golfs some of my work).

After the call somebody suggested I highlight it to you both for your enjoyment :)



SEP 7



Fabrizio Montesi

11:28 AM

Testing a bundled definition of `Bisimulation`, and holy cow does `grind` shine. With the right annotations, it managed to prove that bisimilarity is a bisimulation.

```
def Bisimilarity (lts : Lts State Label) : Bisimulation lts :=
  rel s1 s2 := ∃ r : Bisimulation lts, r s1 s2
  is_bisimulation := by grind
```

Chris Henson, Shreyas Srinivas, Kim Morrison

```
- refine (λ _ _ _ _ ha, haj, hb, hbj, hc, hcj, hd, hdj, ?_, ?_, ?_, ?_, ?_)
- <=> rw [mem_insert] at * <=> try rintro rfl
- · obtain (rfl | ha) := ha
- · obtain (rfl | hb) := hb
- · exact hw.isPathGraph3Compl.fst_ne_snd rfl
- · exact hw.fst_notMem_right hb
- · obtain (rfl | hb) := hb
- · exact hw.snd_notMem_left ha
- · exact haj <| hw <| mem_inter_of_mem ha hb
- · obtain (rfl | ha) := ha
- · obtain (rfl | hd) := hd
- · exact hw.isPathGraph3Compl.ne_fst rfl
- · exact hw.fst_notMem_right hd
- · obtain (rfl | hd) := hd
- · exact hw.notMem_left ha
- · exact haj <| hw <| mem_inter_of_mem ha hd
- · obtain (rfl | hb) := hb
- · obtain (rfl | hc) := hc
- · exact hw.isPathGraph3Compl.ne_snd rfl
- · exact hw.snd_notMem_left hc
- · obtain (rfl | hc) := hc
- · exact hw.notMem_right hb
- · exact hbj <| hw <| mem_inter_of_mem hc hb
- · intro hat
- obtain (rfl | ha) := ha
- · exact hw.fst_notMem_right hat
- · exact haj <| hw <| mem_inter_of_mem ha hat
- · intro hbs
- obtain (rfl | hb) := hb
- · exact hw.snd_notMem_left hbs
- · exact hbj <| hw <| mem_inter_of_mem hbs hb
+ exact (λ _ _ _ _ ha, haj, hb, hbj, hc, hcj, hd, hdj, by grind)
```

Comment on line R312



Yael Dillies 19 minutes ago

Collaborator ...

Wow! 🤩



grind: Initial reactions



Terence Tao

@tao@mathstodon.xyz

In contrast, AI chatbots are usually tuned to avoid a "failure mode" as much as possible, at the expense of increasing the occurrence of "intermediate modes" where the chatbot response looks potentially useful, and invites further interaction from the user, but is not exactly providing what the user wants, and could contain hallucinations or some fundamental misunderstanding of the task that would take significant effort to uncover. Paradoxically, such tools may become significantly more useful if they simply reported that they were unable to provide a high quality answer to a query in such cases.

A comparison may be drawn with the increasingly advanced, but stringently verified, "tactics" used in a modern proof assistant such as Lean. I have been experimenting recently with the new tactic ``grind`` in Lean, which is a powerful tool (inspired more by "good old-fashioned AI" such as satisfiability modulo theories (SMT) solvers, than modern data-driven AI) to try to close complex proof goals if all the tools needed to do so are already provided in the proof environment; roughly speaking, this corresponds to proofs that can be obtained by "expanding everything out and trying all obvious combinations of the hypotheses". When I apply ``grind`` to a given subgoal, it can report a success within seconds, closing that subgoal in a Lean-verified fashion and allowing me to move on to the next subgoal. But, importantly, when this does not work, I quickly get a "``grind` failed" message, in which case I simply delete `grind` from the code and proceed by a more pedestrian sequence of lower level tactics. (2/3)`

grind annotations

```
@[grind =]
theorem length_tail {α} {as : List α} : as.tail.length = as.length - 1 := by
  cases as <;> rfl

@[grind →]
theorem mem_or_eq_of_mem_set {α} {as : List α} {i : Nat} {a b : α} : a ∈ as.set i b → a ∈ as ∨ a = b
```

Annotations are **hints** to proof automation.

Lean standard library comes pre-annotated.

We are annotating Mathlib, it is a monumental task. What about user libraries?

Can AI make theorem-proving AI better?

If we had perfect annotations:

Proof synthesis becomes dramatically easier (predict grind instead of 20 tactic steps)

AI-generated proofs become shorter, more robust, more maintainable



Extension Point: grind +suggestions

What about unannotated libraries?

grind +suggestions will “guess” theorems for you.

Lean comes light implementation based on symbolic reasoning.

grind +suggestions is extensible.

You can attach your own AI.

grind +suggestions in action

```

/-- A product set is included in a product set if and only factors are included, or a factor
first set is empty. -/
theorem prod_subset_prod_iff : s ×s t ⊆ s1 ×s t1 ↔ s ⊆ s1 ∧ t ⊆ t1 ∨ s = ∅ ∨ t = ∅ := by
  rcases (s ×s t).eq_empty_or_nonempty with h | h
  · simp [h, prod_eq_empty_iff.1 h]
  have st : s.Nonempty ∧ t.Nonempty := by rwa [prod_nonempty_iff] at h
  refine ⟨fun H => Or.inl ⟨?_, ?_⟩, ?_⟩
  · have := image_mono (f := Prod.fst) H
    rwa [fst_image_prod _ st.2, fst_image_prod _ (h.mono H).snd] at this
  · have := image_mono (f := Prod.snd) H
    rwa [snd_image_prod st.1, snd_image_prod (h.mono H).fst] at this
  · intro H
    simp only [st.1.ne_empty, st.2.ne_empty, or_false] at H
    exact prod_mono H.1 H.2

```



```

/-- A product set is included in a product set if and only factors are included, or a factor
first set is empty. -/
theorem prod_subset_prod_iff : s ×s t ⊆ s1 ×s t1 ↔ s ⊆ s1 ∧ t ⊆ t1 ∨ s = ∅ ∨ t = ∅ := by
  grind +suggestions

```

grind +suggestions: do you have to keep guessing?

```

/-- A product set is included in a product set if and only factors are included, or a factor
first set is empty. -/
theorem prod_subset_prod_iff : s xs t ⊆ s1 xs t1 ↔ s ⊆ s1 ∧ t ⊆ t1 ∨ s = ∅ ∨ t = ∅ := by
  grind +suggestions

```



```

/-- A product set is included in a product set if and only factors are included, or a factor
first set is empty. -/
theorem prod_subset_prod_iff : s xs t ⊆ s1 xs t1 ↔ s ⊆ s1 ∧ t ⊆ t1 ∨ s = ∅ ∨ t = ∅ := by
  grind? +suggestions

```



grind finds out for you exactly with theorems are relevant

```

/-- A product set is included in a product set if and only factors are included, or a factor
first set is empty. -/
theorem prod_subset_prod_iff : s xs t ⊆ s1 xs t1 ↔ s ⊆ s1 ∧ t ⊆ t1 ∨ s = ∅ ∨ t = ∅ := by
  grind only [prod_eq_empty_iff, prod_subset_iff, prod_mono, = subset_def, mem_empty_iff_fa
  mem_prod, empty_subset]

```

First reward signal: reproving existing libraries

Hammer-bench: collecting **data about proof automation effectiveness**.

<https://github.com/leanprover-community/hammer-bench>

grind provides diagnostic information.

```
set_option diagnostics true
theorem mem_insert (m : IndexMap  $\alpha$   $\beta$ ) (a a' :  $\alpha$ ) (b :  $\beta$ ) :
  a'  $\in$  m.insert a b  $\leftrightarrow$  a' = a  $\vee$  a'  $\in$  m := by
  grind
```

▼ grind_indexmap.lean:149:2

```
[grind] Diagnostics ▼
[thm] E-Matching instances ▼
[] Array.size_push  $\mapsto$  6
[] WF  $\mapsto$  3
[] getElem_indices_lt  $\mapsto$  3
[] size.eq_1  $\mapsto$  3
[] HashMap.contains_iff_mem  $\mapsto$  3
[] HashMap.contains_insert  $\mapsto$  3
[] HashMap.mem_insert  $\mapsto$  3
[] insert.match_1.congr_eq_1  $\mapsto$  3
[] insert.match_1.congr_eq_2  $\mapsto$  3
```



Second reward signal: #grind_lint

```
/--
info: instantiating `List.replicate_sublist_iff` triggers 30 additional `grind` theorem instantiations
---
info: List.replicate_sublist_iff
[thm] instances
  [thm] List.count_eq_length_filter ↪ 3
  [thm] List.count_le_length ↪ 3
  [thm] List.length_filter_le ↪ 3
  [thm] List.Sublist.count_le ↪ 3
  [thm] List.Sublist.eq_of_length ↪ 3
  [thm] List.Sublist.length_le ↪ 3
  [thm] List.replicate_sublist_iff ↪ 2
  [thm] List.subset_def ↪ 2
  [thm] List.Sublist.subset ↪ 2
  [thm] List.count_replicate ↪ 1
  [thm] List.filter_replicate ↪ 1
  [thm] List.filter_subset ↪ 1
  [thm] List.length_replicate ↪ 1
  [thm] List.sublist_filter_iff ↪ 1
  [thm] List.Sublist.filter ↪ 1
---
info: Try this to display the actual theorem instances:
  [apply] set_option trace.grind.ematch.instance true in
  #grind_lint inspect (detailed := 20)List.replicate_sublist_iff
-/
#guard_msgs in
#grind_lint inspect (detailed := 20) List.replicate_sublist_iff
#grind_lint check (min := 30) in List
```

A tool for analyzing annotations.

We want to minimize the number of theorem instantiations generated by grind without reducing the number of problems we can solve.

Can AI/users control grind? Yes, grind interactive mode.

```
example {x : R} (f : R → Nat)
  : max 3 (4 * f ((cos x + sin x)^2)) ≠ 2 + f (2 * cos x * sin x + 1) := by
  grind =>
  use [Nat.max_def, trig_identity]
  ring
  cases_next
```

Already available in v4.25.0.

Higher level of abstraction.

Prediction: AI will learn how to use grind interactive mode in a few months.



What do we need from you?

Baselines: How well does Claude / Gemini / GPT-5 do zero-shot on annotation prediction?

Suggestion oracle: Can you build a neural model that, plugged into grind +suggestions, improves success rate on held-out goals?

RL experiments: Can you train an AI with hammer-bench and #grind_lint that learns to write good annotations?

The hard version: Given a new library the model hasn't seen, can it produce annotations matching expert quality?

Before we wrap up...



Lean FRO: Shaping the Future of Lean Development

The Lean Focused Research Organization (FRO) is a non-profit dedicated to Lean's development.

Founded in **July 2023**, the organization has 19 members.

Its mission is to enhance critical areas: **scalability, proof automation, usability, and documentation.**

We are very grateful for all philanthropic support we have received.



Lean FRO: by numbers

25 releases and **+7,000 pull requests** merged in the main repository only since its launch in July 2023.

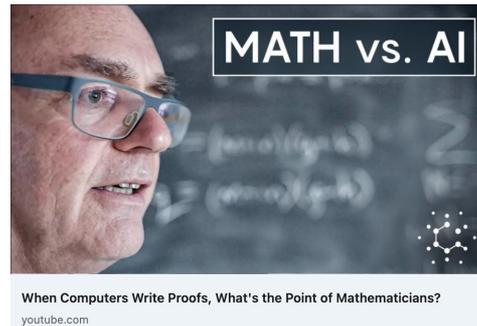
Public roadmaps: <https://lean-lang.org/fro/roadmap/y3/>

Lean project was featured in multiple venues NY Times, Quanta, Scientific American, etc.



A.I. Is Coming for Mathematics, Too

For thousands of years, mathematicians have adapted to the latest advances in logic and reasoning. Are they ready for artificial intelligence?





How can I contribute?

Help building [Mathlib](#) and [CSLib](#).

Want to engage with the vibrant Lean community? Join our [Zulip channel](#).

Start your own open-source Lean project! Your package will be available on our registry [Reservoir](#).

Start using Lean online: live.lean-lang.org

Support the Lean FRO: Funding, partnerships, or simply advocating the project.

Conclusion

Lean is an **efficient programming language** and **proof assistant**.

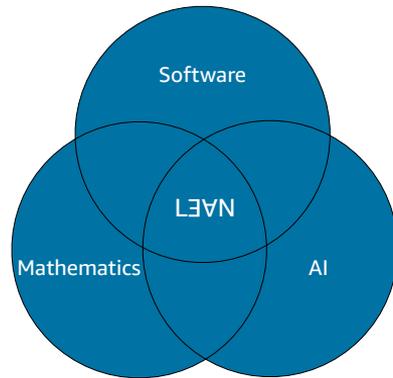
Lean enables Verified AI for Mathematics and Code.

AI should learn to use new symbolic tools.

Proof automation based on NeuroSymbolic AI will accelerate progress.

We have the infrastructure. We need you to build the AI.

[Teaching AI to Make Proof Automation Work](#)



Thank You

<https://leanprover.zulipchat.com/>

x: @leanprover

LinkedIn: Lean FRO

Mastodon: @leanprover@functional.cafe

#leanlang, #leanprover

<https://lean-lang.org/>

