

AI for (Formal) Mathematics

Sean Welleck

April 15, 2025

Carnegie Mellon University

AI agents in expert domains

- Finance
- Medicine
- Mathematics

AI agents in expert domains

AI agents in expert domains

- Finance
- Medicine
- Mathematics
 - Open-ended dialogue
 - Come up with counterexamples
 - Help write proofs
 - ...

“Informal” mathematics

Math as raw data (text, images, ...)

- Flexible
- Widely used
- Difficult to check

Input:

Let r be a root of $x^2 + 5x + 7 = 0$. Compute $(r - 1)(r + 2)(r + 6)(r + 3)$.

LLEMMA 34B solution:

We have

$$\begin{aligned}(r - 1)(r + 2)(r + 6)(r + 3) &= (r^2 + r - 2)(r^2 + 9r + 18) \\&= r^4 + 10r^3 + 25r^2 + 19r - 36 \\&= r^4 + 5r^3 + 7r^2 + 5r + 7 + 5r^3 + 20r^2 + 14r - 36 \\&= r^4 + 5r^3 + 25r^2 + 19r - 29 \\&= (r^2 + 5r + 7)(r^2 + 4r - 5) \\&= (0)(r^2 + 4r - 5) \\&= \boxed{0}.\end{aligned}$$

Final Answer: The final answer is 0.

Language model solution.

Math as **source code**

- Write a specification (e.g., $1+1=2$)
- Write a proof
- **Automatically checked**
 - Code compiles \equiv correct proof

$$1 + 1 = 2$$

proof ✓

```
lemma one_plus_one_equals_two:
  shows "1 + 1 = 2"
proof -
  have "1 + 1 = Suc (0 + 1)" by simp
  also have "... = Suc 1" by simp
  also have "... = 2" by simp
  finally show ?thesis by simp
qed
```

Math as source code.

Math as **source code**

- Write a specification (e.g., $1+1=2$)
- Write a proof
- **Automatically checked**
 - Code compiles \equiv correct proof



Lean



Isabelle



Coq

Theorem proving languages

If $R \subseteq S$ and $S \subseteq T$ then $R \subseteq T$



How is formal math used in practice?

Growing use in mathematics:



Terence Tao

@tao@mathstodon.xyz

Finished formalizing in #Lean4 the proof of an actual new theorem (Theorem 1.3) in my recent paper arxiv.org/abs/2310.05328 :

Terence Tao's Lean formalization project (October 2023)

How is formal math used in practice?

Growing use in mathematics:



Terence Tao

@tao@mathstodon.xyz

Finished formalizing in #Lean4 the proof of an actual new theorem (Theorem 1.3) in my recent paper arxiv.org/abs/2310.05328 :

Terence Tao's Lean formalization project (October 2023)

- **Lean Mathlib** project: 1+ million lines of code, 300+ contributors

How is formal math used in practice?

Why?¹

- **Collaboration**

- Break down a big problem into multiple pieces
- Anyone can submit code to solve a piece
- We know we can trust the code since it is automatically checked!

¹See e.g., *Mathematics and the formal turn*, AFM Aims and Scope

How is formal math used in practice?

Why?¹

- **Collaboration**

- Break down a big problem into multiple pieces
- Anyone can submit code to solve a piece
- We know we can trust the code since it is automatically checked!

- **Instant feedback**

¹See e.g., *Mathematics and the formal turn*, AFM Aims and Scope

How is formal math used in practice?

Why?¹

- **Collaboration**
 - Break down a big problem into multiple pieces
 - Anyone can submit code to solve a piece
 - We know we can trust the code since it is automatically checked!
- **Instant feedback**
- **Guaranteed correctness**

¹See e.g., *Mathematics and the formal turn*, AFM Aims and Scope

How is formal math used in practice?

Why?¹

- **Collaboration**
 - Break down a big problem into multiple pieces
 - Anyone can submit code to solve a piece
 - We know we can trust the code since it is automatically checked!
- **Instant feedback**
- **Guaranteed correctness**
- ...

¹See e.g., *Mathematics and the formal turn*, AFM Aims and Scope

Why is AI \cap formal math important?

Formal math for AI

- **Verifiable**
 - Prevent incorrect math and code generation
 - Feedback signal for learning

Why is AI \cap formal math important?

Formal math for AI

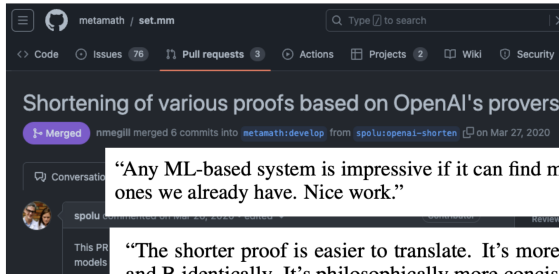
- **Verifiable**
 - Prevent incorrect math and code generation
 - Feedback signal for learning
- Tests **reasoning**
 - From easy: $1+1 = 2$
 - To hard: Fermat's Last Theorem

Generative Language Modeling for Automated Theorem Proving

Stanislas Polu
OpenAI
spolu@openai.com

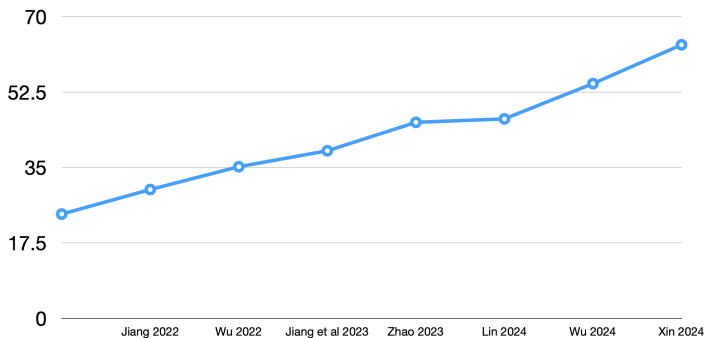
Ilya Sutskever
OpenAI
ilyasu@openai.com

gpt-f (2020)



gpt-f (2020)

Rapid progress in methods based on language models:



miniF2F benchmark performance, 2022-2024

```

theorem imo_1960_p2 (x : ℝ) (h₀ : 0 ≤ 1 + 2 * x) (h₁ : (1 - Real.sqrt (1 + 2 *
  x)) ^ 2 ≠ 0)
  (h₂ : 4 * x ^ 2 / (1 - Real.sqrt (1 + 2 * x)) ^ 2 < 2 * x + 9) : -(1 / 2)
  ≤ x ∧ x < 45 / 8 := by
  norm_num at h₀ h₁ h₂
  have h₃ : 0 ≤ 1 + 2 * x := by linarith
  have h₄ : 0 < 1 + Real.sqrt (1 + 2 * x) := by
    nlinarith [Real.sqrt_nonneg (1 + 2 * x)]
  have h₅ : 4 * x ^ 2 / (1 - Real.sqrt (1 + 2 * x)) ^ 2 < 2 * x + 9 := by
    linarith
  have h₆ : 1 - Real.sqrt (1 + 2 * x) ≠ 0 := by
    intro h
    apply h₁
    nlinarith
  have h₇ : 4 * x ^ 2 / (1 - Real.sqrt (1 + 2 * x)) ^ 2 = (1 + Real.sqrt (1 +
    2 * x)) ^ 2 := by
    field_simp [h₆]
    nlinarith [sq_sqrt (show 0 ≤ 1 + 2 * x by linarith)]
  rw [h₇] at h₅
  constructor <=> nlinarith [sq_sqrt (show 0 ≤ 1 + 2 * x by linarith)]

```

Generated International Math Olympiad solution in Lean
(DeepSeek Prover-1.5B, Xin et al 2024)



Terence Tao

@tao@mathstodon.xyz

Finished formalizing in #Lean4 the proof of an actual new theorem (Theorem 1.3) in my recent paper arxiv.org/abs/2310.05328 :

The ability of Github copilot to correctly anticipate multiple lines of code for various routine verifications, and inferring the direction I want to go in from clues such as the names I am giving the theorems, continues to be uncanny.

Terence Tao's Lean formalization project (October 2023)



Terence Tao

@tao@mathstodon.xyz

Finished formalizing in #Lean4 the proof of an actual new theorem (Theorem 1.3) in my recent paper arxiv.org/abs/2310.05328 :

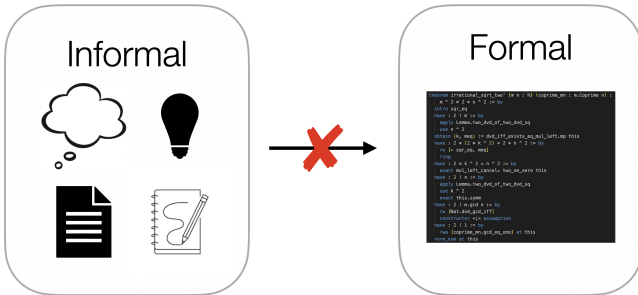
The ability of Github copilot to correctly anticipate multiple lines of code for various routine verifications, and inferring the direction I want to go in from clues such as the names I am giving the theorems, continues to be uncanny.

Terence Tao's Lean formalization project (October 2023)

So...why don't people and AI always use formal math?

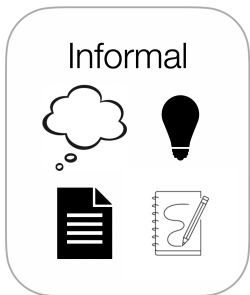
Key challenge: the informal-formal gap

Informal ideas, intuitions, and even proofs are difficult to express formally:

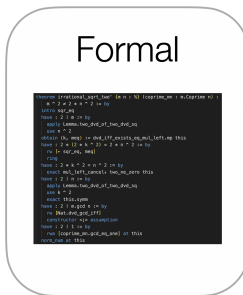


- Each step of reasoning needs to be specified in detail
- Requires a deep knowledge of the formal system

Bridging Informal and Formal Mathematical Reasoning



Flexible



Easy to check

This talk: Bridging Informal and Formal Mathematical Reasoning with AI

This talk: Bridging Informal and Formal

1. Informal thoughts
2. Informal provers
3. Research-level mathematics

This talk: Bridging Informal and Formal

1. Informal thoughts
 - Training models to think informally
 - Lean-STaR
2. Informal provers
3. Research-level mathematics

This talk: Bridging Informal and Formal

1. Informal thoughts

- Training models to think informally
 - Lean-STaR

2. Informal provers

- Sketching proofs and filling in the gaps
 - Draft, Sketch, Prove
 - LeanHammer

3. Research-level mathematics

This talk: Bridging Informal and Formal

1. Informal thoughts

- Training models to think informally
 - Lean-STaR

2. Informal provers

- Sketching proofs and filling in the gaps
 - Draft, Sketch, Prove
 - LeanHammer

3. Research-level mathematics

- Assisting in research-level projects
 - Practical tools
 - MiniCTX

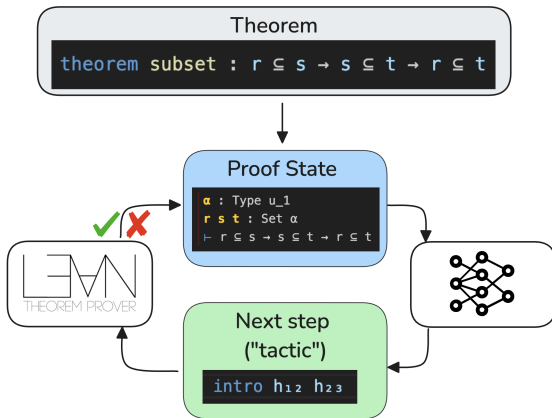
I: Informal thoughts

1. Training models to “think” — Lean-STaR

Lean-STaR: Learning to Interleave Thinking and Proving
Haohan Lin, Zhiqing Sun, Yiming Yang, Sean Welleck
ICLR 2025 (Spotlight)

1. Training models to “think” — Neural theorem proving

Neural theorem proving



- Math as checkable code
- Proof: sequence of (state, step)

1. Training models to “think” — Neural theorem proving

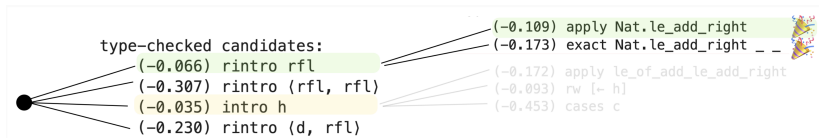
Language model-based proving:

- **Train** a model $p_{\theta}(y|x)$ on a dataset $\mathcal{D} = \{(x, y)\}$, e.g.,
 - x : proof state
 - y : next tactic (next “step”)
 - \mathcal{D} : extracted from theorems and proofs

1. Training models to “think” — Neural theorem proving

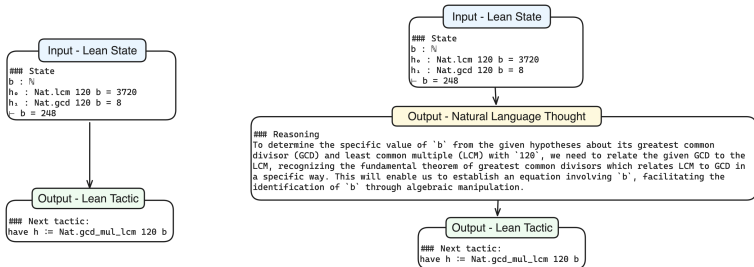
Language model-based proving:

- **Train** a model $p_{\theta}(y|x)$ on a dataset $\mathcal{D} = \{(x, y)\}$, e.g.,
 - x : proof state
 - y : next tactic (next “step”)
 - \mathcal{D} : extracted from theorems and proofs
- **Generate** proofs:



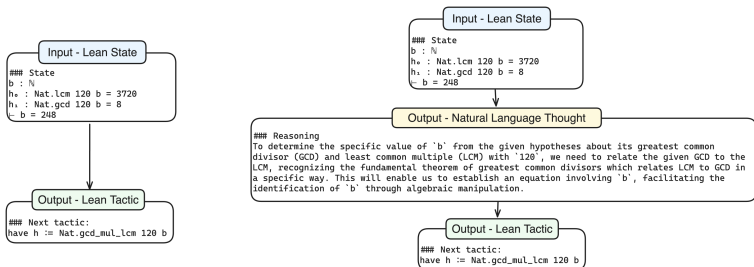
Best-first search

1. Training models to “think” — Lean-STaR



Can we train a model to “think” before each step of formal reasoning?

1. Training models to “think” — Lean-STaR



Why?

- Plan proof steps
- Diversify search space
- More tokens can give more computational capacity

1. Training models to “think” — Lean-STaR

Lean-STaR (Self-taught reasoner²)

Learn to generate thoughts via reinforcement learning

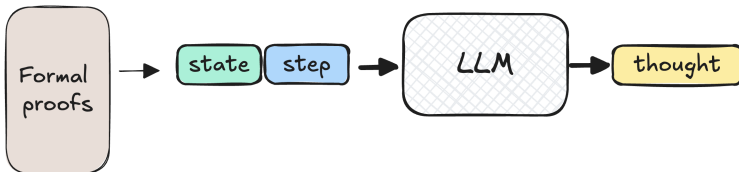
1. Initialization
2. Reinforcement learning

²Inspired by *STaR: Bootstrapping Reasoning with Reasoning*, Zelikman et al 2022

1. Training models to “think” — Lean-STaR

1. Initialization

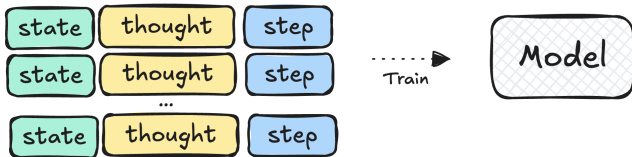
Annotate thoughts “retrospectively”



1. Training models to “think” — Lean-STaR

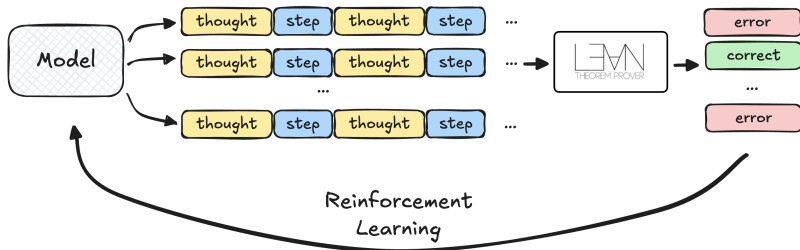
1. Initialization

Train initial model on
(state, thought) \rightarrow step examples



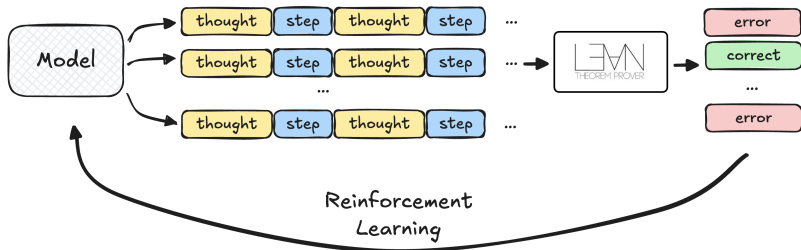
1. Training models to “think” — Lean-STaR

2: Reinforcement learning



1. Training models to “think” — Lean-STaR

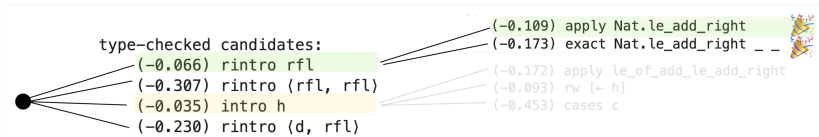
2: Reinforcement learning



Need:

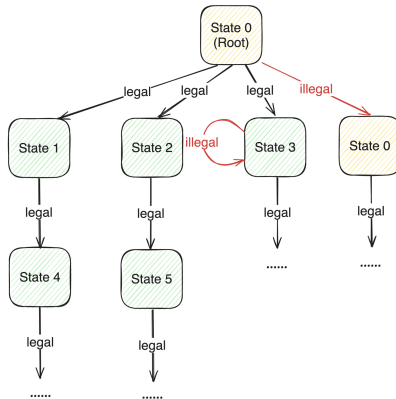
- Method to generate proofs
- Learning algorithm

1. Training models to “think” — Lean-STaR



Best-first search: difficult to score (thought, tactic) candidates

1. Training models to “think” — Lean-STaR



New sampling method

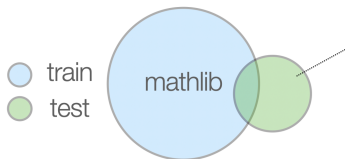
Algorithm: train on the successful proofs, and repeat:³

- Collect (state, thought, tactic) from successful proofs
- Train a new model $p_{\theta}^1(\textit{thought}, \textit{tactic} | \textit{state})$
- Generate proofs
- ...

³I.e. Expert Iteration [Polu et al 2022], Rest-EM [Singh et al 2024]

Training models to “think” — Lean-STaR

- **miniF2F**: competition problems (AMC, AIME, IMO)



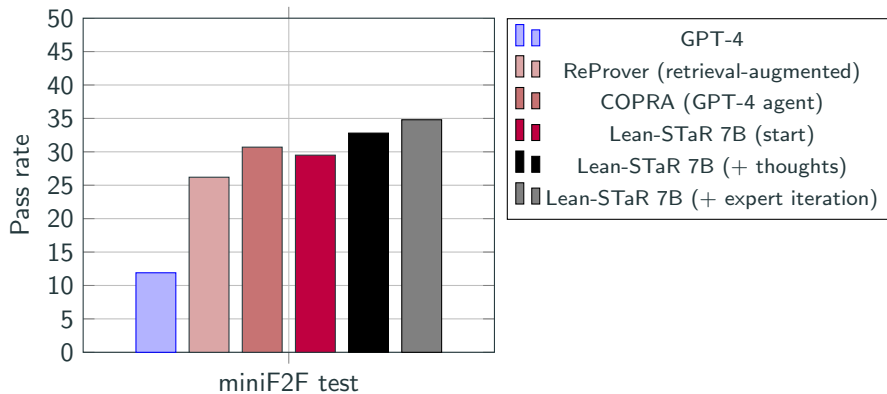
Problem 1959 IMO Problems/Problem 1

Prove that the fraction $\frac{21n+4}{14n+3}$ is irreducible for every natural number n .

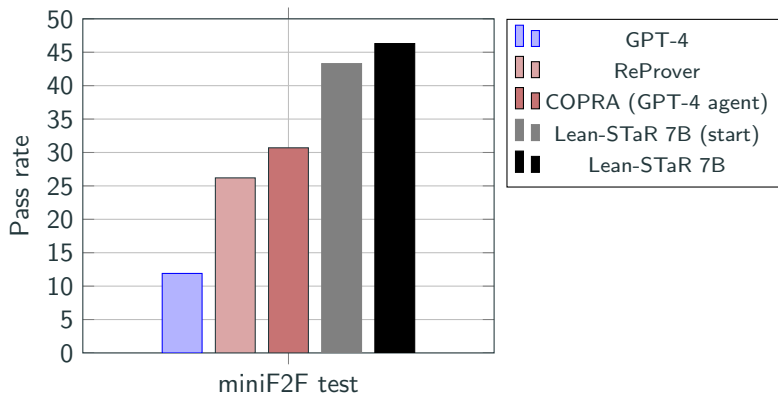


```
theorem imo_1959_p1
  (n : ℕ)
  (h₀ : 0 < n) :
  Nat.gcd (21*n + 4) (14*n + 3) = 1 := by sorry
```

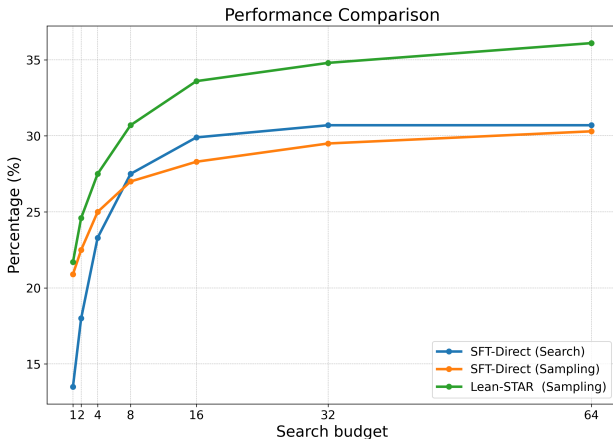
Training models to “think” — Lean-STaR



Training models to “think” — Lean-STaR



Training models to “think” — Lean-STaR



Increasing the search budget is more effective with thoughts

Training models to “think” — Lean-STaR

```
theorem amc12a_2002_p6 (n : ℕ) (h₀ : 0 < n)
  : ∃ m, m > n ∧ ∃ p, m * p ≤ m + p :=
  use n + 1, by linarith only [h₀] -- Reasoning: To demonstrate the existence of
  such an `m` and `p` that satisfy the given inequalities, we leverage the
  property of natural numbers that ensures an increase in `n` from `0` leads to
  the possibility of finding such `m` and `p`. This approach is supported by the
  fundamental property of natural numbers and their arithmetic operations,
  suggesting that for a given `n` greater than `0`, there exists a successor `m`
  and a number `p` that satisfy the conditions outlined.
exact ⟨0, by simp⟩ -- Reasoning: To demonstrate that there exists an  $(n \in \mathbb{N})$ 
such that  $((n + 1) * p \leq n + 1 + p)$  holds for some  $(p \in \mathbb{N})$ , we
recognize that setting  $(p = 0)$  satisfies the inequality by the properties of
zero in arithmetic. This is because  $(n + 1 > 0)$  by assumption, ensuring the
left-hand side is non-negative and the right-hand side simplifies to  $(n + 1)$ ,
making the inequality trivially true.
```

Example generated thoughts and proof from Lean-STaR

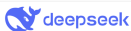
Training models to “think” Lean-STaR

```
theorem aime_1983_p2 (x p : ℝ) (f : ℝ → ℝ) (h₀ : 0 < p ∧ p < 15) (h₁ : p ≤ x ∧ x ≤ 15)
(h₂ : f x = abs (x - p) + abs (x - 15) + abs (x - p - 15))
: 15 ≤ f x :=
cases' h₁ with h₁ h₂ -- Reasoning: Given a function  $\lfloor f \rfloor$  defined over the real
numbers, where  $\lfloor f(a) \rfloor$  equals the sum of the absolute values of the
differences  $\lfloor a - p \rfloor$ ,  $\lfloor a - 15 \rfloor$ , and  $\lfloor a - p - 15 \rfloor$ , along with conditions
on  $\lfloor p \rfloor$  and  $\lfloor a \rfloor$  that ensure  $\lfloor a \rfloor$  lies between  $\lfloor p \rfloor$  and  $\lfloor 15 \rfloor$  and  $\lfloor p \rfloor$ 
is positive, we aim to show that  $\lfloor f(a) \rfloor \geq 15$ . To demonstrate this, we
consider all possible cases for the position of  $\lfloor a \rfloor$  relative to  $\lfloor p \rfloor$  and
 $\lfloor 15 \rfloor$ , specifically at  $\lfloor p \rfloor$  itself, between  $\lfloor p \rfloor$  and  $\lfloor 15 \rfloor$ , and at  $\lfloor 15 \rfloor$ .
By evaluating  $\lfloor f(a) \rfloor$  for each case, we can establish a lower bound of
 $\lfloor f(a) \rfloor$  that is greater than or equal to  $\lfloor 15 \rfloor$ .
simp (h₁, abs_of_nonneg (sub_nonneg.2 h₁), abs_of_nonneg (sub_nonneg.2 h₂), *)
-- Reasoning: The objective is to show that  $\lfloor f(a) \rfloor$  is at least 15 for  $\lfloor a \rfloor$ 
within the bounds  $\lfloor p \rfloor \leq a \leq 15$ , where  $\lfloor p \rfloor$  is positive but less than
15. The provided expression for  $\lfloor f(a) \rfloor$  suggests using properties of absolute
values and inequalities to simplify and analyze  $\lfloor f(a) \rfloor$ . By applying
simplification rules for absolute values and inequalities, along with the
given bounds, we aim to demonstrate the desired inequality  $\lfloor f(a) \rfloor \geq 15$ .
simp only [abs_of_nonneg] at * -- Reasoning: The objective is to establish an
inequality involving an expression with absolute values and a sum, where all
terms are functions of  $\lfloor a \rfloor$ . By recognizing that the absolute value functions
are always non-negative, we can simplify the given expression by removing the
absolute value symbols, thus focusing on the arithmetic manipulation of the
terms to demonstrate the inequality.
simp only [abs_of_nonneg, sub_add_eq_add_sub] at * -- Reasoning: The objective
is to demonstrate that the expression  $15 \leq a + p + \lfloor a - 15 \rfloor + \lfloor a - p - 15 \rfloor$ 
holds given the conditions on  $\lfloor p \rfloor$ ,  $\lfloor a \rfloor$ , and the function  $\lfloor f \rfloor$ . Since the
expression in question involves absolute values which inherently involve
non-negativity, manipulating these expressions to highlight their
non-negativity or equality is a logical initial step. This involves
recognizing that the absolute value of a difference can be rewritten as the
difference of absolute values, which simplifies the manipulation of the
expression to prove the inequality.
congr 1 -- Reasoning: To establish the inequality  $15 \leq a + \lfloor a - 15 \rfloor + \lfloor a - p - 15 \rfloor - p$ ,
we recognize that the left-hand side of the inequality involves a
mix of additions and subtractions of real numbers. By introducing a congruence
transformation that simplifies operations involving absolute values, we aim to
align both sides of the inequality in a form that directly reflects the
properties of real numbers under these operations.
simp only [abs_of_nonpos (sub_nonpos.of_le h₂), abs_of_nonneg (sub_nonneg.of_le
h₁)] -- Reasoning: The objective is to prove an inequality involving absolute
values and subtractions that aligns the given conditions and constraints.
Simplification using the properties of absolute values and the provided bounds
can directly lead to the desired conclusion by expressing the absolute values
in terms of their underlying expressions and applying the given constraints.
simp [abs_of_pos, *] -- Reasoning: To prove the given inequality involving
absolute values, simplification using algebraic properties and given
hypotheses will directly lead us to the desired inequality by recognizing the
algebraic manipulation and application of the known bounds  $\lfloor h_1 \rfloor$  and  $\lfloor h_2 \rfloor$ , as
well as the non-negativity of  $\lfloor p \rfloor$ .
rw [abs_of_nonpos] -- Reasoning: The objective is to demonstrate that  $15 \leq 15 + \lfloor a - p - 15 \rfloor - p$ 
under the given constraints on  $\lfloor p \rfloor$  and  $\lfloor a \rfloor$ . Given  $\lfloor p \rfloor < 15$ 
and  $\lfloor p \rfloor \leq a \leq 15$ , it's apparent that  $\lfloor a - p - 15 \rfloor$  is non-positive. This
observation allows us to apply the property that the absolute value of a
non-positive number is equal to its negative value, thereby simplifying the
expression on the right-hand side of the inequality.
all_goals linarith -- Reasoning: The goal involves demonstrating a relationship
between expressions involving inequalities and subtraction. The simplification
of these expressions into a form that directly compares their numerical values
can lead to a straightforward application of known inequalities and algebraic
properties, demonstrating the inequality's validity under the given conditions.
```

Example generated thoughts and proof from Lean-STaR

Training models to “think” — After Lean-STaR

After Lean-STaR, incorporating thoughts became a widely-used component of LLM-based theorem proving:



DeepSeek-Prover-V1.5: Harnessing Proof Assistant Feedback for Reinforcement Learning and Monte-Carlo Tree Search

Huajian Xin*, Z.Z. Ren*, Junxiao Song*, Zhihong Shao*, Wanjia Zhao, Haocheng Wang, Bo Liu, Liyue Zhang
Xuan Lu, Qiushi Du, Wenjun Gao, Qihao Zhu, Dejian Yang, Zhibin Gou, Z.F. Wu, Fuli Luo, Chong Ruan

DeepSeek-AI

Output:

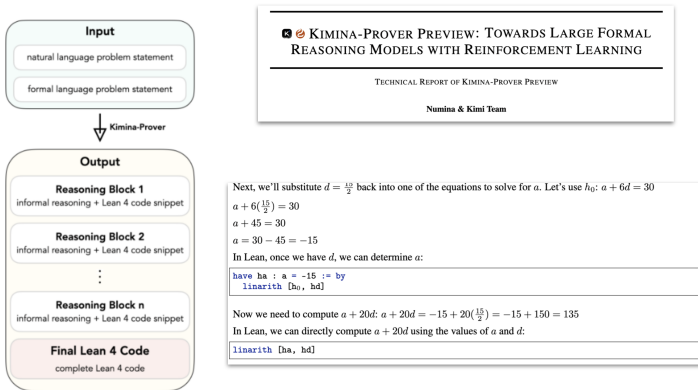
```
-- This is equivalent to showing that  $9x^2 \sin^2 x - 12x \sin x + 4 \geq 0$ , and the  
  left-hand side can be rewritten as a perfect square  $(3x \sin x - 2)^2$ .  
-- We use the fact that  $(3x \sin x - 2)^2$  is non-negative to establish this.  
nlinarith [sq_nonneg (3 * x * Real.sin x - 2)]  
, , ,
```

generating theorem proof code. Similar to Lean-STaR (Lin et al., 2024), which performs isolated chain-of-thought reasoning (Wei et al., 2022; Feng et al., 2023) before each proof step, our method integrates this reasoning directly as comments within the proof code. We use the DeepSeek-

Informal thoughts in DeepSeek Prover 1.5

Training models to “think” — After Lean-STaR

After Lean-STaR, incorporating thoughts became a widely-used component of LLM-based theorem proving:



Informal thoughts in Kimina Prover (April 14 2025)

Training models to “think” — After Lean-STaR

More broadly, *reasoning models* that generate long chains-of-thought have subsequently begun to revolutionize LLM reasoning:



OpenAI o1 reasoning model



DeepSeek-R1: Incentivizing Reasoning Capability in LLMs via Reinforcement Learning

DeepSeek-AI
research@deepseek.com

DeepSeek R1 reasoning model

Recap: **Lean-STaR**

- Training on formal code may be insufficient to learn the underlying thought process needed to produce the code
- Learn to generate thoughts via reinforcement learning

This talk: Bridging Informal and Formal

1. Informal thoughts
2. **Informal provers**
 - Sketching proofs and filling in the gaps
 - Draft, Sketch, Prove
 - LeanHammer
3. Research-level mathematics

II: Informal and formal provers

Combining informal and formal provers

Overall goal: combine high-level reasoning and low-level reasoning

Motivation: informal proofs and formal proofs

Statement

Prove that n is 70
if $\gcd(n, 40) = 10$ and
 $\text{lcm}(n, 40) = 280$.

Informal proof

We know that $\gcd(a, b) \cdot \text{lcm}(a, b) = ab$,
hence $10 \cdot 280 = n \cdot 40$.

Then $n = 10 \cdot 280/40 = 70$,

completing the proof. ■

How would we write this as a formal proof?

Motivation: informal proofs and formal proofs

Informal proof

We know that $\gcd(a, b) \cdot \text{lcm}(a, b) = ab$,
hence $10 \cdot 280 = n \cdot 40$.

Then $n = 10 \cdot 280 / 40 = 70$,

completing the proof. ■

Formal proof

```
have c1: "10*280 = n*40"  
using assms
```

```
by (smt (z3) prod_gcd_lcm_nat)
```

```
then have c2: "n = 10*280/40"
```

```
by auto
```

```
then show ?thesis
```

```
by auto
```



A proof with a **high-level** sketch and **low-level** proof steps.

Low-level provers: Sledgehammer



Low-level provers: Sledgehammer

```
theorem algebra_simps2 [simp]:  
  fixes x :: 'a  
  shows "0 \le x \implies (x + 1) = x + 2"
```

Theorem



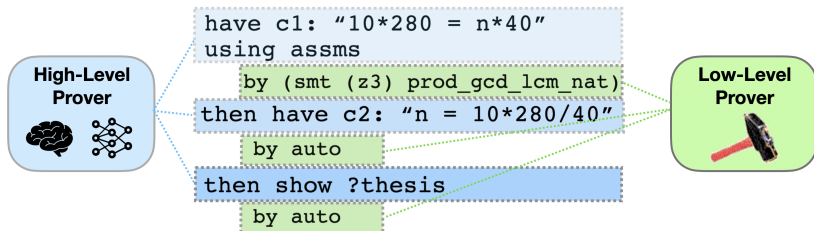
Sledgehammer [Paulson 2010]

```
theorem algebra_simps2 [simp]:  
  fixes x :: 'a  
  shows "0 \le x \implies (x + 1) = x + 2"  
proof -  
  have c0: "x \le 0 \implies False"  
  proof -  
    assume c1: "x \le 0" and c2: "0 \le x" and c3: "x + 1 = x + 2"  
    then show False using c1 c2 c3 by auto </ATP>  
  qed  
  have c4: "1 < x \implies False"  
  proof -  
    assume c5: "1 < x" and c6: "0 \le x" and c7: "x + 1 = x + 2"  
    then show False using c5 c6 c7 by auto </ATP>  
  qed  
  have c8: "abs(x-1) + abs x + abs(x+1) = -(x-1) - x + (x+1)" using c1 c2  
  by auto </ATP>  
  have c9: "abs(x-1) + abs x + abs(x+1) = 2*x" <ATP> by auto </ATP>  
  have c10: "2*x = x+2" using c9 c8 <ATP> by auto </ATP>  
  have c11: "x = 0" <ATP> by auto </ATP>  
  have c12: "x = 0" <ATP> by auto </ATP>  
  show thesis using c11 c12 <ATP> by auto </ATP>  
qed  
have c13: "x > 1 \implies False"  
proof -  
  assume c14: "x > 1"  
  have c15: "abs(x-1) + abs x + abs(x+1) = x-1 + x + (x+1)" using c14  
  by auto </ATP>  
  then have c16: "abs(x-1) + abs x + abs(x+1) = 3*x" <ATP> by auto </ATP>  
  then have c17: "3*x = x+2" using c16 <ATP> by auto </ATP>  
  then have c18: "x = 1" <ATP> by auto </ATP>  
  then show False using c18 <ATP> by auto </ATP>  
qed  
then show thesis using c13 c18 <ATP> by fastforce </ATP>  
qed
```

complex proof

Struggles due to the large search space of possible proofs

Idea: combine high-level and low-level proving



Idea: combine high-level (human, LLM) and low-level proving

Draft, Sketch, Prove: Guiding Formal Theorem Provers with Informal Proofs

Albert Q. Jiang, Sean Welleck, Jin Peng Zhou

Jiacheng Liu, Wenda Li, Mateja Jamnik

Guillaume Lample, Yuhuai Wu

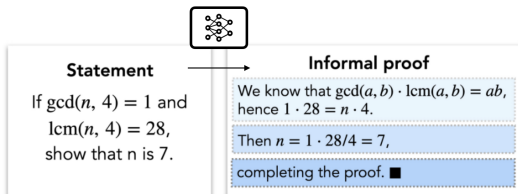
ICLR 2023 (Oral)

Idea:

Draft an informal proof, translate it into a formal **sketch**, then use a low-level **prover** to fill in the gaps.

Given informal theorem x_I ,
formal theorem x_F

1. **Draft** $y_I \sim p(\cdot | x_I)$

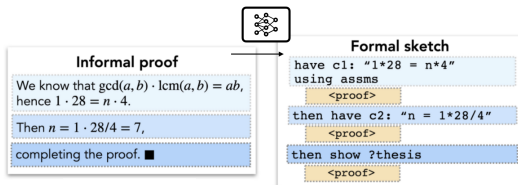


Human-written or LLM-generated draft

Draft, Sketch, Prove

Given informal theorem x_I ,
formal theorem x_F

1. Draft $y_I \sim p(\cdot | x_I)$
2. **Sketch** $z_F \sim p(\cdot | x_F, x_I, y_I)$

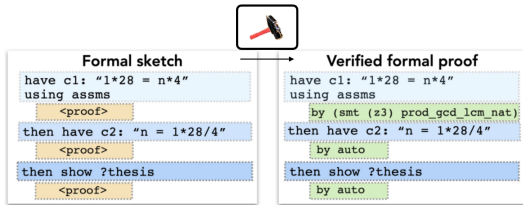


LLM-generated sketch

Draft, Sketch, Prove

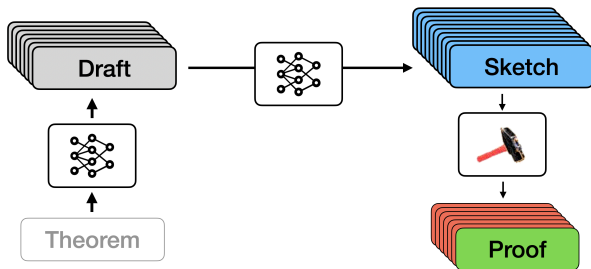
Given informal theorem x_I ,
formal theorem x_F

1. Draft $y_I \sim p(\cdot | x_I)$
2. Sketch $z_F \sim p(\cdot | x_F, x_I, y_I)$
3. **Prove** $y_F = f(x_F, z_F)$

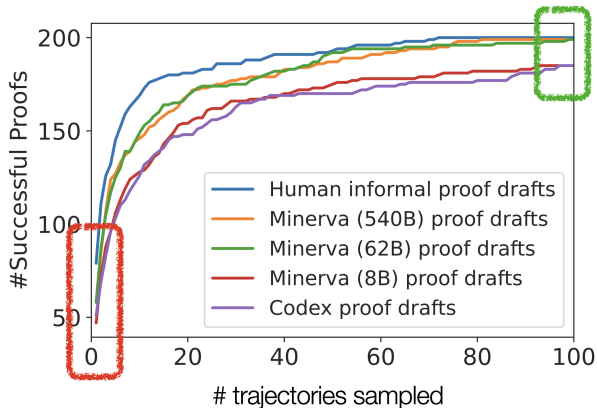


Low-level prover (*Sledgehammer*) fills in the gaps

Proof search



Draft-sketch-prove



Inference-time proof search scaling



Statement: Prove that the fraction $\frac{21n+4}{14n+3}$ is irreducible for every natural number n .

Informal Proof (Minerva 62B):

We must show that $\gcd(21n + 4, 14n + 3) = 1$.

The Euclidean algorithm gives

$$21n + 4 = 1 \cdot (14n + 3) + 7n + 1$$

$$14n + 3 = 2 \cdot (7n + 1) + 1.$$

Since $\gcd(7n + 1, 1) = 1$, we have $\gcd(21n + 4, 14n + 3) = 1$.

Formal Proof:

```
theorem imo_1959_p1:
```

```
  fixes n :: nat
```

```
  shows "gcd (21*n + 4) (14*n + 3) = 1"
```

```
proof -
```

```
  (* The Euclidean algorithm gives
```

```
  21n+4=1\cdot(14n+3)+7n+1
```

```
  14n+3=2\cdot(7n+1)+1. *)
```

```
  have c0: "21*n + 4 = 1*(14*n + 3) + 7*n + 1"
```

```
    by auto [ATP]
```

```
  have c1: "14*n + 3 = 2*(7*n + 1) + 1" using c0
```

```
    by auto [ATP]
```

```
  (* Since \gcd(7n+1,1)=1, we have \gcd(21n+4,14n+3)=1. *)
```

```
  then have "gcd (7*n + 1) 1 = 1"
```

```
    using c1
```

```
    by auto [ATP]
```

```
  then have "gcd (21*n + 4) (14*n + 3) = 1"
```

```
    using c1
```

```
    by (smt (z3) BitM_plus_one ab_semigroup_add_class.add_ac(1)
```

```
    add.assoc c0 gcd.commute gcd_add2 gcd_add_mult mult_numeral_1
```

```
    numeral_One numeral_eq_Suc numerals(1) semiring_norm(3)) [ATP]
```

```
  then show ?thesis
```

```
    using c1
```

```
    by blast [ATP]
```

```
qed
```


Draft, Sketch, Prove



The screenshot shows a Jupyter Notebook window titled "ll_dsp_part2_dsp.ipynb". The interface includes a toolbar with icons for saving, adding, deleting, and running cells, as well as a "Markdown" dropdown menu. The notebook is running on "Python 3 (ipykernel)".

The first cell is a code cell containing the following Python function definition:

```
#### Language cascades | part 2: Draft, Sketch, Prove ...  
  
[1]: def dsp(xi, xf, f_draft, f_sketch, f_proof):  
      yi = f_draft(xi)  
      zf = f_sketch(yi, xi, xf)  
      yf = f_proof(xf, zf)  
      return yf
```

The second cell is a text cell containing the following text:

Next, we will discuss how to implement these three modules.

We start by introducing the [Isabelle proof assistant](#), since it is relevant to the implementation.

Demo notebook: github.com/cmu-l3/ntptutorial-II

Next: building a low-level prover for Lean

Recap:

- Draft-Sketch-Prove: generate high-level sketches and fill in gaps
- Isabelle's *Sledgehammer* calls out to external provers to fill in gaps

Next: building a low-level prover for Lean

Recap:

- Draft-Sketch-Prove: generate high-level sketches and fill in gaps
- Isabelle's *Sledgehammer* calls out to external provers to fill in gaps

Next: can we build a Sledgehammer for Lean?

Premise Selection for a Lean Hammer

Thomas Zhu, Joshua Clune

Jeremy Avigad, Albert Q. Jiang, Sean Welleck

Under Review 2025

What is a hammer?

A **hammer** integrates an automated theorem prover into an interactive theorem prover

What is a hammer?

A **hammer** integrates an automated theorem prover into an interactive theorem prover

- Interactive theorem prover: Lean, Isabelle, Coq
 - A programming language that checks proofs

What is a hammer?

A **hammer** integrates an automated theorem prover into an interactive theorem prover

- Interactive theorem prover: Lean, Isabelle, Coq
 - A programming language that checks proofs
- Automated theorem prover: higher-order logic provers, SMT solvers
 - Tries to automatically find proofs

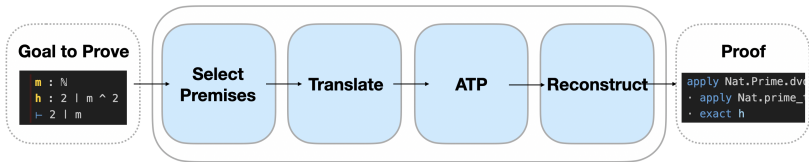
Key challenge: Premise selection

Automated theorem provers (ATPs) struggle with the large search space of possible proofs.

- **Premise selection:** select a small subset of theorems and definitions that are likely to be useful for proving a given theorem.
 - Around 250,000 premises in Lean's Mathlib
 - Cuts down the prover's search space

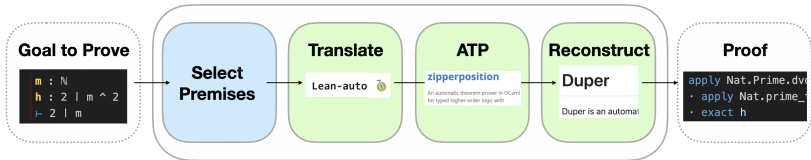
A hammer pipeline

A standard hammer pipeline:



A hammer pipeline

A standard hammer pipeline:



Pre-existing components:

- **Translation:** LeanAuto [Qian et al 2025]
- **ATP:** Zipperposition [Cruanes et al 2015]
- **Reconstruction:** Duper [Clune et al 2024]

A hammer pipeline

A standard hammer pipeline:

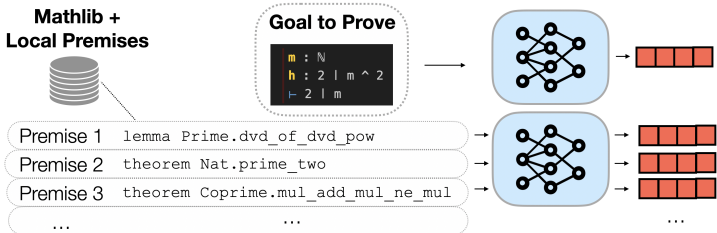


Our challenge:

- **Premise selection**
- **Put it all together** to create LeanHammer

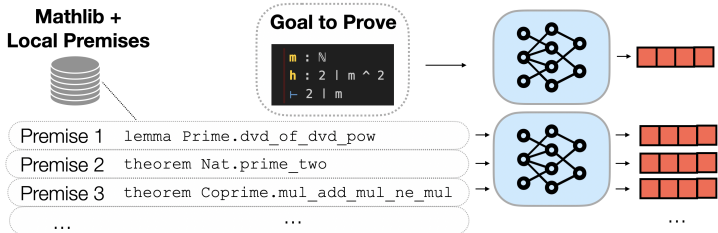
LeanHammer — Neural premise selection

Idea: frame premise selection as retrieval with a neural language model



LeanHammer — Neural premise selection

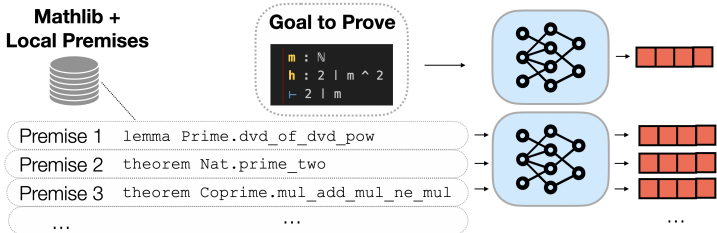
Idea: frame premise selection as retrieval with a neural language model



- Transformer encoder embeds the state and candidate premises

LeanHammer — Neural premise selection

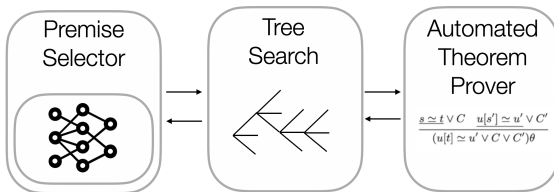
Idea: frame premise selection as retrieval with a neural language model



- Transformer encoder embeds the state and candidate premises
- Contrastive loss on $(\text{state}, \{\text{premise}^+\}, \{\text{premise}^-\})$ examples
 - Nuance in how to collect and format examples

LeanHammer — Putting it all together

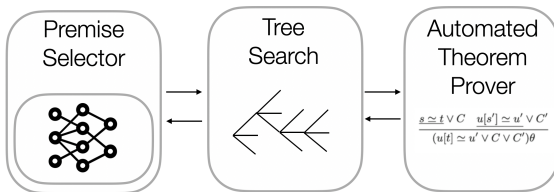
Idea: combine the premise selector and ATP with a tree search



Tree search: *Aesop* [Limperg & From 2023]

LeanHammer — Putting it all together

Idea: combine the premise selector and ATP with a tree search

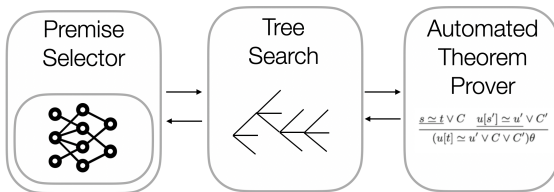


Tree search: *Aesop* [Limperg & From 2023]

1. Queries the automated theorem prover using the premises
2. Applies tactics (e.g. `apply`, `simp_all`) using the premises

LeanHammer — Putting it all together

Idea: combine the premise selector and ATP with a tree search



Tree search: *Aesop* [Limperg & From 2023]

1. Queries the automated theorem prover using the premises
2. Applies tactics (e.g. `apply`, `simp_all`) using the premises

Goes beyond the standard hammer pipeline!

As a user, simply issue `hammer` at any step of a proof:

```
theorem two_dvd_of_two_dvd_sq {m : ℕ}
  (h : 2 ∣ m ^ 2) : 2 ∣ m := by
  hammer
```

Try this:

```
apply Nat.Prime.dvd_of_dvd_pow
· apply Nat.prime_two
· exact h
```

LeanHammer in action

Demo: start with human-written proof sketch (from *Mathematics in Lean*)

```
/-- Theorem taken from Mathematics in Lean -/  
theorem irrational_sqrt_two {m n : ℕ} (coprime_mn : m.Coprime n) :  
  | m ^ 2 ≠ 2 * n ^ 2 := by  
  intro sqr_eq  
  have : 2 ∣ m := by  
    | sorry  
  obtain (k, meq) := dvd_iff_exists_eq_mul_left.mp this  
  have : 2 * (2 * k ^ 2) = 2 * n ^ 2 := by  
    | sorry  
  have : 2 * k ^ 2 = n ^ 2 := by  
    | sorry  
  have : 2 ∣ n := by  
    | sorry  
  have : 2 ∣ m.gcd n := by  
    | sorry  
  have : 2 ∣ 1 := by  
    | sorry  
  sorry
```

Demo: fill in the gaps (sorrys) with LeanHammer

Human
Written

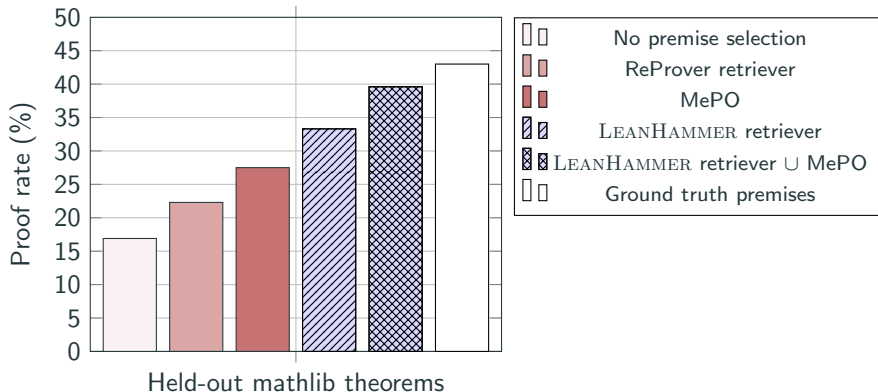
LEAN



```
theorem irrational_sqrt_two {m n : ℕ} (coprime_mn : m.Coprime n) :  
  | m ^ 2 ≠ 2 * n ^ 2 := by  
  intro sqreq  
  have : 2 ∣ m := by  
    apply Lemma.two_dvd_of_two_dvd_sq  
    simp_all only [dvd_mul_right]  
  obtain (k, meq) := dvd_iff_exists_eq_mul_left.mp this  
  have : 2 * (2 * k ^ 2) = 2 * n ^ 2 := by  
    subst meq  
    simp_all only [dvd_mul_left, mul_eq_mul_left_iff, OfNat.ofNat_ne_zero, or_false]  
    (linarith)  
  have : 2 * k ^ 2 = n ^ 2 := by  
    subst meq  
    simp_all only [mul_eq_mul_left_iff, OfNat.ofNat_ne_zero, or_false, dvd_mul_left]  
  have : 2 ∣ n := by  
    subst meq  
    simp_all only [dvd_mul_left]  
    (hammerCore [] [*, dvd_mul_right, Nat.modEq_zero_iff_dvd, Nat.ModEq.symm, Nat.modEq_iff_dvd',  
      Nat.Coprime.mul_add_mul_ne_mul, frobeniusNumber_pair, add_le_mul, FrobeniusNumber, Nat.ModEq.trans,  
      Nat.cast_mul, sq, Nat.Coprime.dvd_of_dvd_mul_left, AddSubmonoid.mem_closure_pair, Nat.chineseRemainder_lt_mul,  
      Nat.Coprime.eq_one_of_dvd, Lemma.two_dvd_of_two_dvd_sq] {simpTarget := no_target})  
  have : 2 ∣ m.gcd n := by  
    clear coprime_mn  
    subst meq  
    simp_all only [dvd_mul_left]  
    apply Nat.dvd_gcd  
    · simp_all only [dvd_mul_left]  
    · simp_all only  
  have : 2 ∣ 1 := by  
    subst meq  
    simp_all only [dvd_mul_left, Nat.dvd_one, OfNat.ofNat_ne_one]  
  subst meq  
  simp_all only [Nat.dvd_one, OfNat.ofNat_ne_one]
```

LeanHammer — Quantitative results

Varying the premise selector within LEANHAMMER:



Two approaches for combining informal and formal provers:

Two approaches for combining informal and formal provers:

- **Draft-Sketch-Prove (DSP)**
 - LLM drafts informal proof, generates formal sketch
 - Low-level prover (Sledgehammer) fills in low-level details

Two approaches for combining informal and formal provers:

- **Draft-Sketch-Prove (DSP)**

- LLM drafts informal proof, generates formal sketch
- Low-level prover (Sledgehammer) fills in low-level details

- **LeanHammer**

- Brings hammer functionality to Lean
- Neural premise selection + tree search + automated theorem proving
- Enables filling in proof sketches with `hammer` command

Two approaches for combining informal and formal provers:

- **Draft-Sketch-Prove (DSP)**

- LLM drafts informal proof, generates formal sketch
- Low-level prover (Sledgehammer) fills in low-level details

- **LeanHammer**

- Brings hammer functionality to Lean
- Neural premise selection + tree search + automated theorem proving
- Enables filling in proof sketches with `hammer` command

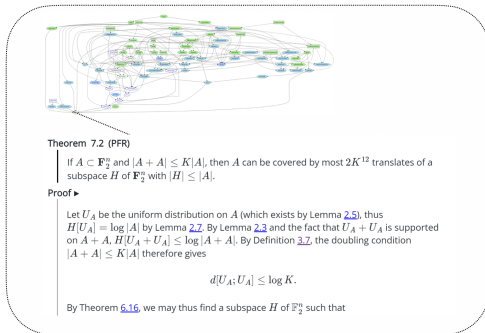
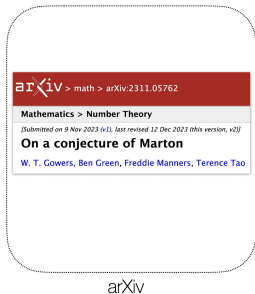
Even small neural networks are powerful! (retriever: $< 100\text{M}$ params)

This talk: Bridging Informal and Formal

1. Informal thoughts
2. Informal provers
3. **Research-level mathematics**
 - Assisting in research-level projects
 - Practical tools
 - MiniCTX

III: Research-level mathematics

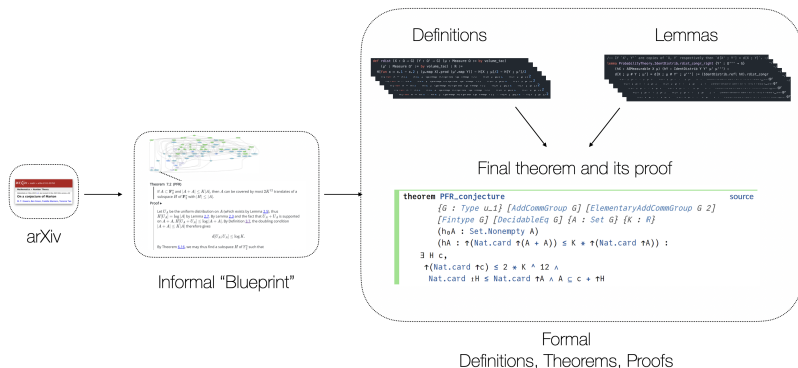
What does it look like to formalize research-level math?⁴



Informal “Blueprint”

⁴Formalizing the proof of PFR in Lean4 using Blueprint: a short tour by Terence Tao

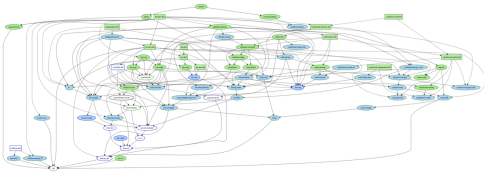
What does it look like to formalize research-level math?⁴



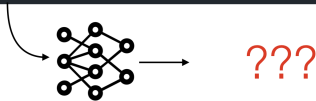
⁴Formalizing the proof of PFR in Lean4 using Blueprint: a short tour by Terence Tao

Where can AI help?

As a start, can AI help with filling in small parts of the blueprint?



```
/-- `H[X | Y=y] = \sum_s P[X=s | Y=y] \log 1/(P[X=s | Y=y])`. -/  
lemma entropy_cond_eq_sum (μ : Measure Ω) (y : T) :  
  H[X | Y ← y ; μ] = \sum' x, negMulLog ((μ[|Y ← y]).map X {x}).toReal := by
```



Accessibility gap:

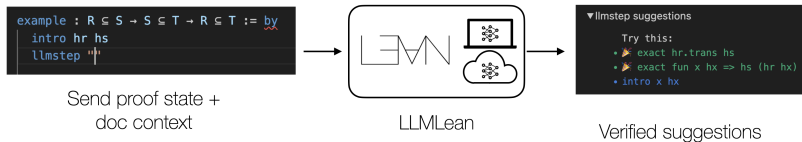
- Some methods are hard to integrate into tools
 - Not open-source (AlphaProof, ...)
 - Expensive to run (MCTS, ...)

Accessibility gap:

- Some methods are hard to integrate into tools
 - Not open-source (AlphaProof, ...)
 - Expensive to run (MCTS, ...)

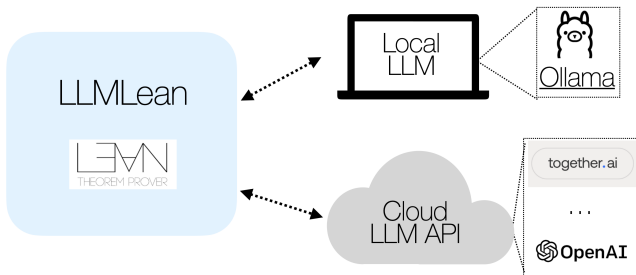
However, there **are** already tools available!

Where can AI help? — Existing tools



LLMLean: <https://github.com/cmu-l3/llmlean>

Where can AI help? — Existing tools



LLMLean: <https://github.com/cmu-l3/llmlean>

Where can AI help? — Existing tools

```
PFR > ForMathlib > Entropy > Basic.lean > {} ProbabilityTheory > {} entro
37 namespace ProbabilityTheory
41 section entropy
127 (hX : Measurable X) {μ : Measure Ω} [IsZeroOrPr
128 entropy X μ = ∑ x ∈ FiniteRange.toFinset X, neg
129 entropy_eq_sum_finset (A := FiniteRange.toFinset
130
131 lemma entropy_eq_sum_finiteRange' [MeasurableSingle
132 [IsZeroOrProbabilityMeasure μ] [FiniteRange X]:
133 entropy X μ = ∑ x ∈ FiniteRange.toFinset X, neg
134 entropy_eq_sum_finiteRange hX
135
136 /-- `H[X | Y=y] = ∑_s P[X=s | Y=y] log 1/(P[X=s | Y
137 lemma entropy_cond_eq_sum (μ : Measure Ω) (y : T) :
138 | H[X | Y ← y ; μ] = ∑' x, negMulLog ((μ[Y ← y])
139 llmqed
140
T : Type u_3
mΩ : MeasurableSpace Ω
inst1 : MeasurableSpace S
X : Ω → S
Y : Ω → T
μ : Measure Ω
γ : T
⊢ H[X | Y ← γ ; μ] = ∑' (x : S), ((map X μ[|Y ← γ]
{x})].toReal.negMulLog
▼LLMLean suggestions
Try this:
• rw [entropy_def]
• rw [measureEntropy_of_isProbabilityMeasure]
• rw [entropy_def]
simp only
```

LLMLean example on Polynomial Freiman Rusza Conjecture project

Where can AI help? — Benchmarking gap

Math competition problems



Theorem to prove

```
theorem imo_1964_p1_2 (n : ℕ) : ~7 ∣ 2 ^ n + 1 := by  
  sorry
```

- Self-contained
- Uses standard results

miniCTX: Neural Theorem Proving with (Long-)Contexts

Jiewen Hu, Thomas Zhu, Sean Welleck

*ICLR 2025 (***Oral***)*

Research-level theorems depend on newly-formalized **context**

- $(\text{context}, \text{theorem}) \rightarrow \text{proof}$
 - Context: repository of code, new definitions, auxiliary lemmas

miniCTX:

Test models on real Lean projects:⁵

- “Future mathlib”: theorems added after a time cutoff
- Recent projects: PFR, PrimeNumberTheorem, ...

⁵+ tools for easily adding new projects: <https://github.com/cmu-l3/ntp-toolkit>

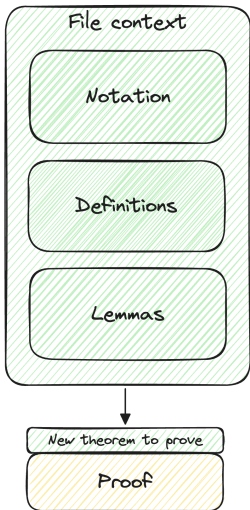
miniCTX:

Test models on real Lean projects:⁵

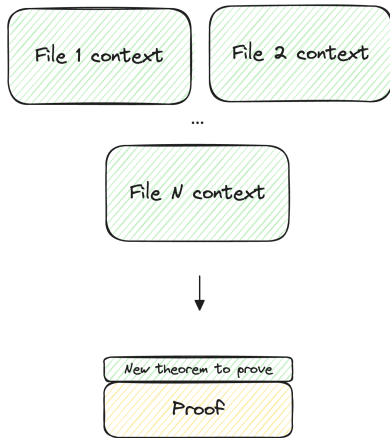
- “Future mathlib”: theorems added after a time cutoff
- Recent projects: PFR, PrimeNumberTheorem, ...

Periodically updated with new projects to stay ahead of LLM training cutoffs.

⁵+ tools for easily adding new projects: <https://github.com/cmu-l3/ntp-toolkit>



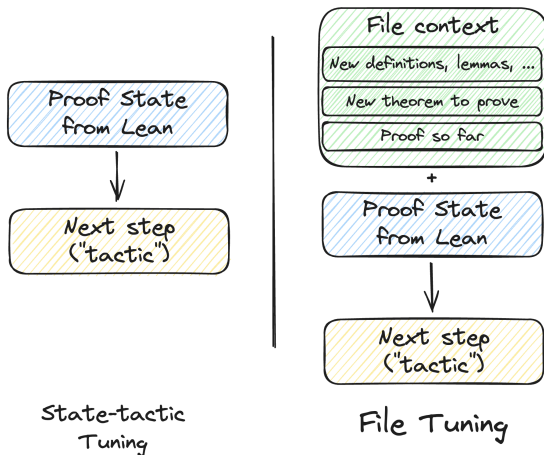
In-file: dependencies appear in the file



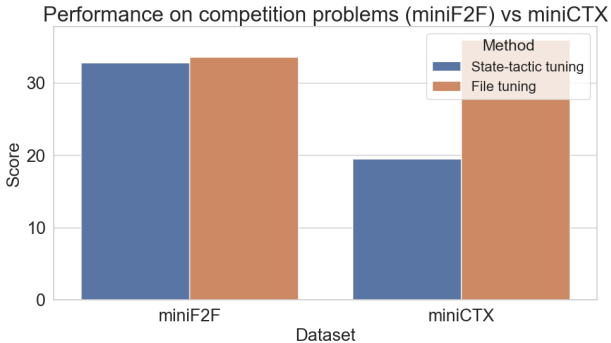
Cross-file: dependencies span files

miniCTX — preceding code context

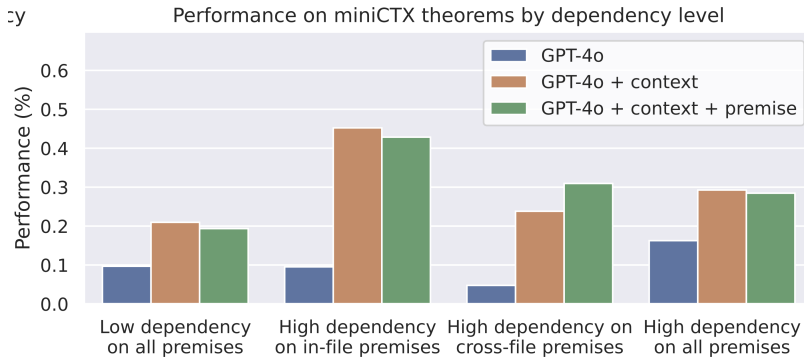
Does context actually matter? A simple experiment.



"File tuning": train on (preceding code, state, next-tactic) examples



Two methods can have similar performance on competition problems, but vastly difference performance on actual projects



Premise selection helps with cross-file dependencies.

File-tuned model is deployed in LLMLean:

LLM on your laptop:

1. Install [ollama](#).
2. Pull a language model:

```
ollama pull wellecks/ntpctx-llama3-8b
```



<https://github.com/cmu-l3/llmlean>

Benchmark, models, code are open-source:

- Data/models: <https://huggingface.co/l3lab>
- Data extraction: <https://github.com/cmu-l3/ntp-toolkit>
- Evaluation: <https://github.com/cmu-l3/minictx-eval>

Recap: Towards AI for Research-Level Formalization

Formalizing research-level math has unique challenges

- **Accessibility gap** exists between AI advances and real-world formalization
 - But we have some tools available!
- **Benchmarking gap** exists between competition problems and real-world formalization
 - miniCTX tests the ability to generalize to new, real-world projects

This talk: Bridging Informal and Formal

1. Informal thoughts

- Training models to think informally
 - Lean-STaR

2. Informal provers

- Sketching proofs and filling in the gaps
 - Draft, Sketch, Prove
 - LeanHammer

3. Research-level mathematics

- Assisting in research-level projects
- Practical tools
- MiniCTX

Thank you!

Collaborators on works in this talk (alphabetical by last name):

- Jeremy Avigad (CMU)
- Joshua Clune (CMU)
- Jiewen Hu (CMU)
- Mateja Jamnik (Cambridge)
- Albert Q. Jiang (Cambridge, Mistral)
- Timothee Lacroix (Meta, Mistral)
- Guillaume Lample (Meta, Mistral)
- Haohan Lin (Tsinghua)
- Wenda Li (Edinburgh)
- Jiacheng Liu (Washington)
- Zhiqing Sun (CMU, OpenAI)
- Yuhuai (Tony) Wu (Google, X.ai)
- Yiming Yang (CMU)
- Jin Peng Zhou (Cornell)
- Thomas Zhu (CMU)

Sean Welleck
CMU School of Computer Science
Learning, Language, and Logic (L3) Lab
wellecks@cmu.edu