

# **Bridging Informal and Formal Mathematical Reasoning with AI**

---

Sean Welleck

April 14, 2025

Carnegie Mellon University

# AI agents in expert domains

## 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  $(r - 1)(r + 2)(r + 6)(r + 3)$ .

LEMMA 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.

# Formal mathematics

## 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.

# Formal mathematics

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

# Formal mathematics (Demo)

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](https://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](https://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?<sup>1</sup>

- **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!

---

<sup>1</sup>See e.g., *Mathematics and the formal turn*, AFM Aims and Scope

# How is formal math used in practice?

Why?<sup>1</sup>

- **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**

---

<sup>1</sup>See e.g., *Mathematics and the formal turn*, AFM Aims and Scope

# How is formal math used in practice?

Why?<sup>1</sup>

- **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**

---

<sup>1</sup>See e.g., *Mathematics and the formal turn*, AFM Aims and Scope

# How is formal math used in practice?

Why?<sup>1</sup>

- **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**
- ...

---

<sup>1</sup>See 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](mailto:spolu@openai.com)

Ilya Sutskever

OpenAI

[ilyasu@openai.com](mailto:ilyasu@openai.com)

*gpt-f* (2020)

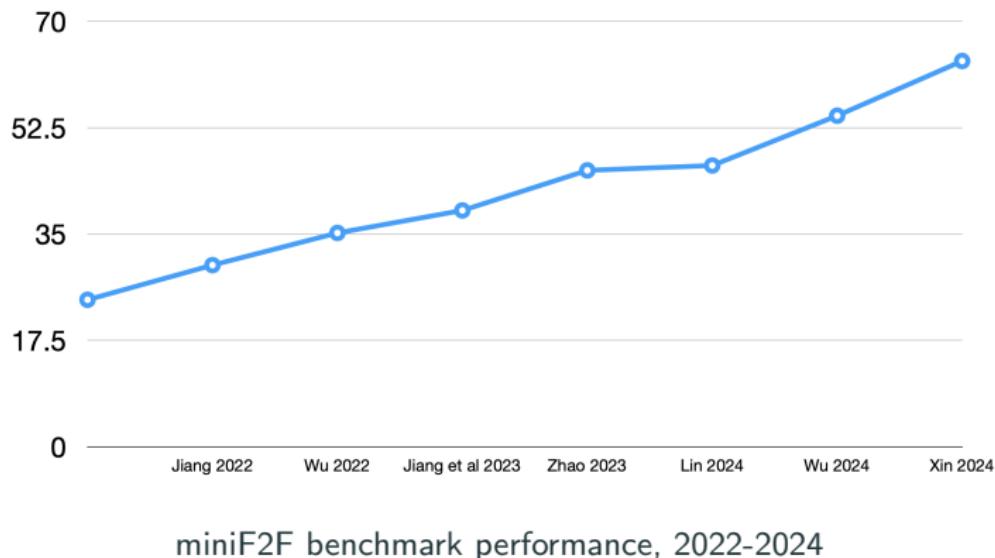
# LLMs $\cap$ formal math

The screenshot shows a GitHub pull request page for the repository 'metamath / set.mm'. The title of the pull request is 'Shortening of various proofs based on OpenAI's provers'. A note indicates that it is '1 Merged' and shows a merge commit from 'nmegill' into 'metamath:develop' from 'spolu:openai-shorten' on March 27, 2020. Two comments are visible:

- A comment by 'spolu' stating: "Any ML-based system is impressive if it can find many shorter proofs than the ones we already have. Nice work."
- A comment by 'gpt-f' (2020) stating: "The shorter proof is easier to translate. It's more symmetric in that it treats A and B identically. It's philosophically more concise in that it doesn't rely on the existence of a universal class of all sets."

*gpt-f (2020)*

Rapid progress in methods based on language models:



# LLMs ∩ formal math

```
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)

# LLMs $\cap$ formal math



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](https://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)

# LLMs $\cap$ formal math



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](https://arxiv.org/abs/2310.05328) :

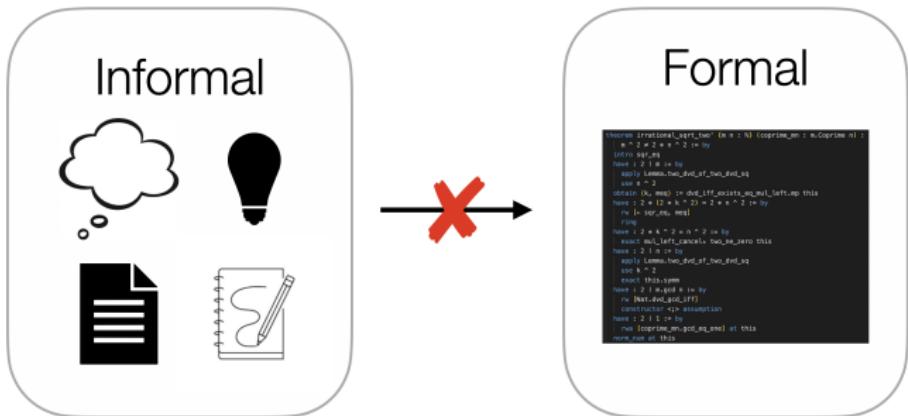
The ability of Github copilot to correctly anticipate multiple lines of code for various routine verifications, and inferring the direction I want to go in from clues such as the names I am giving the theorems, continues to be uncanny.

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:

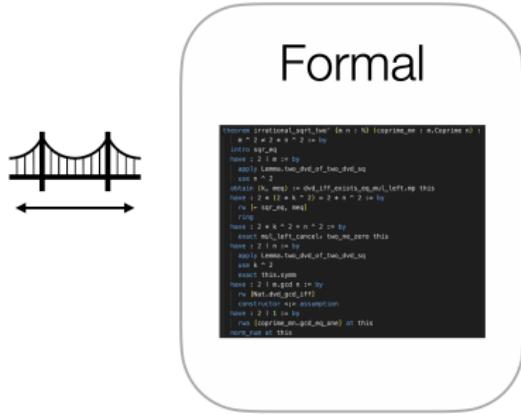


- Each step of reasoning needs to be specified in detail
- Requires a deep knowledge of the formal system

# 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

---

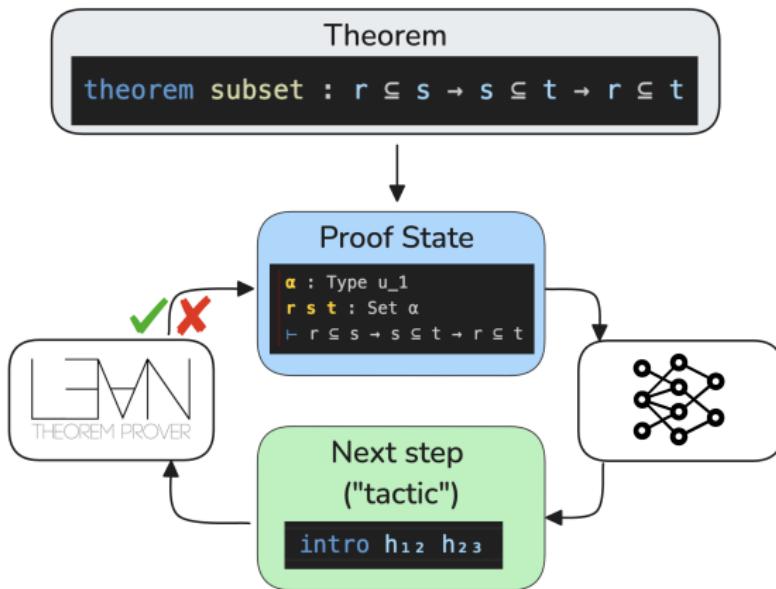
# 1. Training models to “think” — Lean-STaR

---

**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

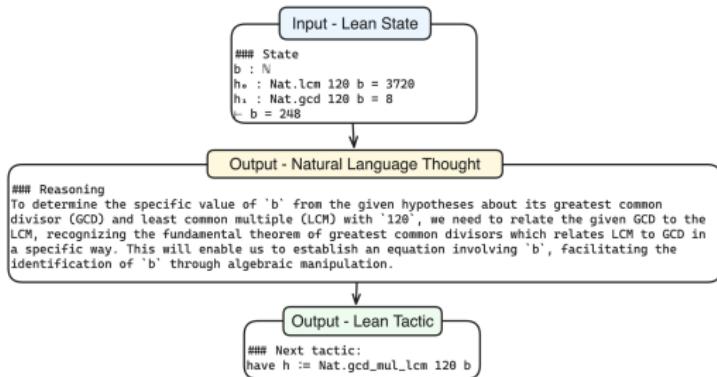
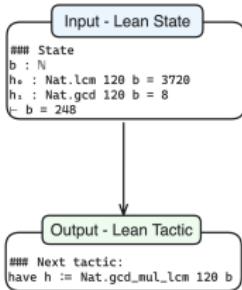
# 1. Training models to “think” — Neural theorem proving

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:

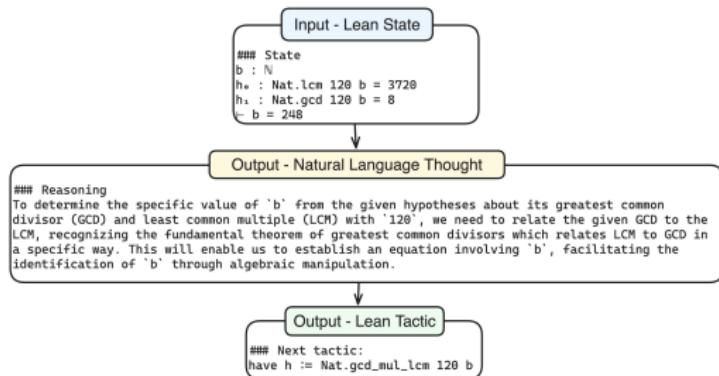
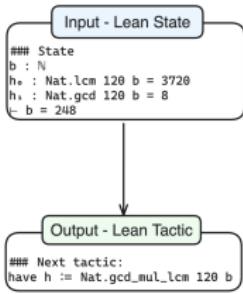


# 1. Training models to “think” — Lean-STaR



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 reasoner<sup>2</sup>)

Learn to generate thoughts via reinforcement learning

1. Initialization
2. Reinforcement learning

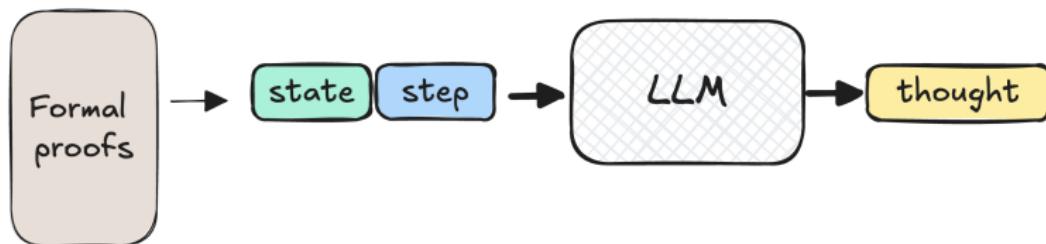
---

<sup>2</sup>Inspired by STaR: Bootstrapping Reasoning with Reasoning, Zelikman et al 2022

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

## 1. Initialization

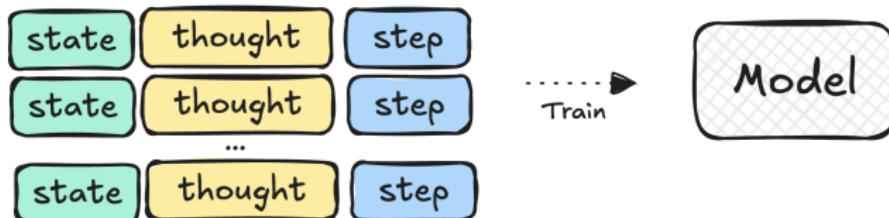
Annotate thoughts “retrospectively”



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

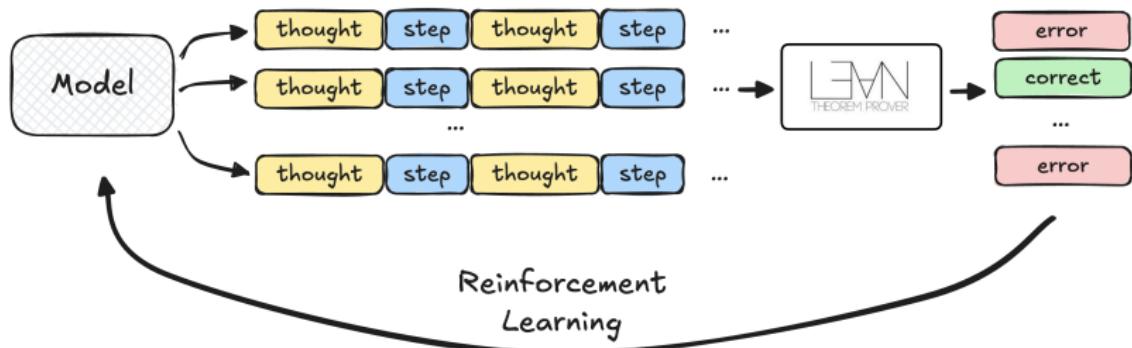
## 1. Initialization

Train initial model on  
(state, thought) → step examples



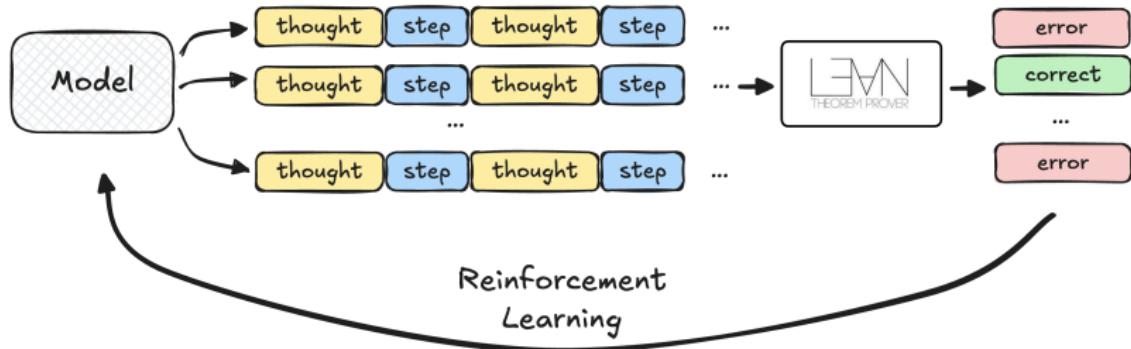
# 1. Training models to “think” — Lean-STaR

## 2: Reinforcement learning



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

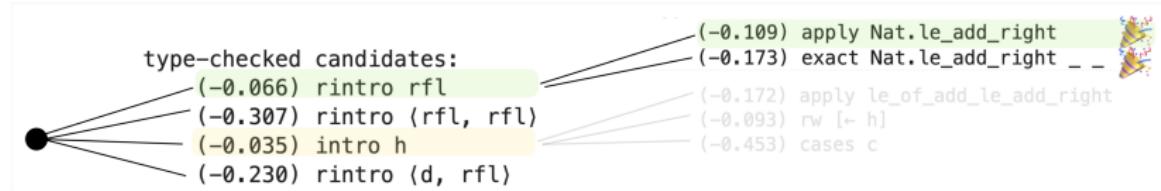
## 2: Reinforcement learning



Need:

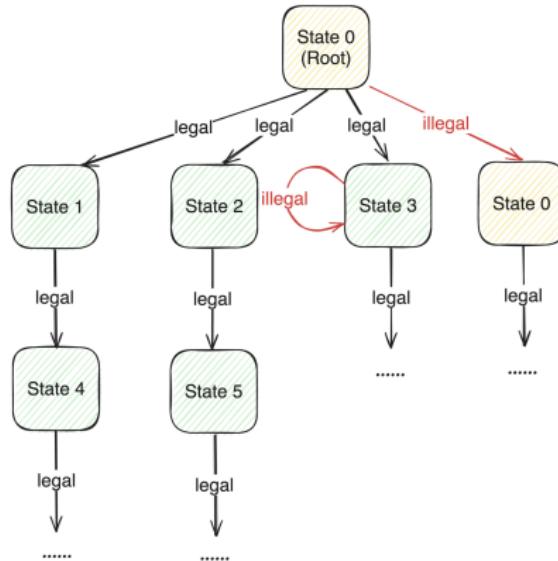
- Method to generate proofs
- Learning algorithm

# 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

## Training models to “think” — Lean-STaR

Algorithm: train on the successful proofs, and repeat:<sup>3</sup>

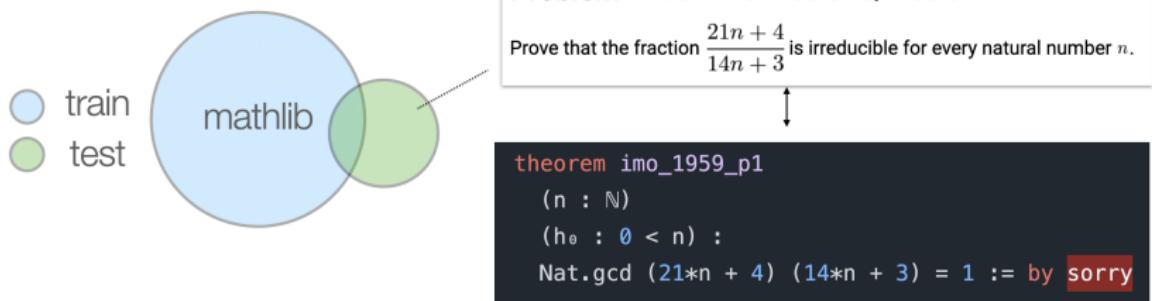
- Collect (state, thought, tactic) from successful proofs
- Train a new model  $p_{\theta}^1(\text{thought}, \text{tactic}|\text{state})$
- Generate proofs
- ...

---

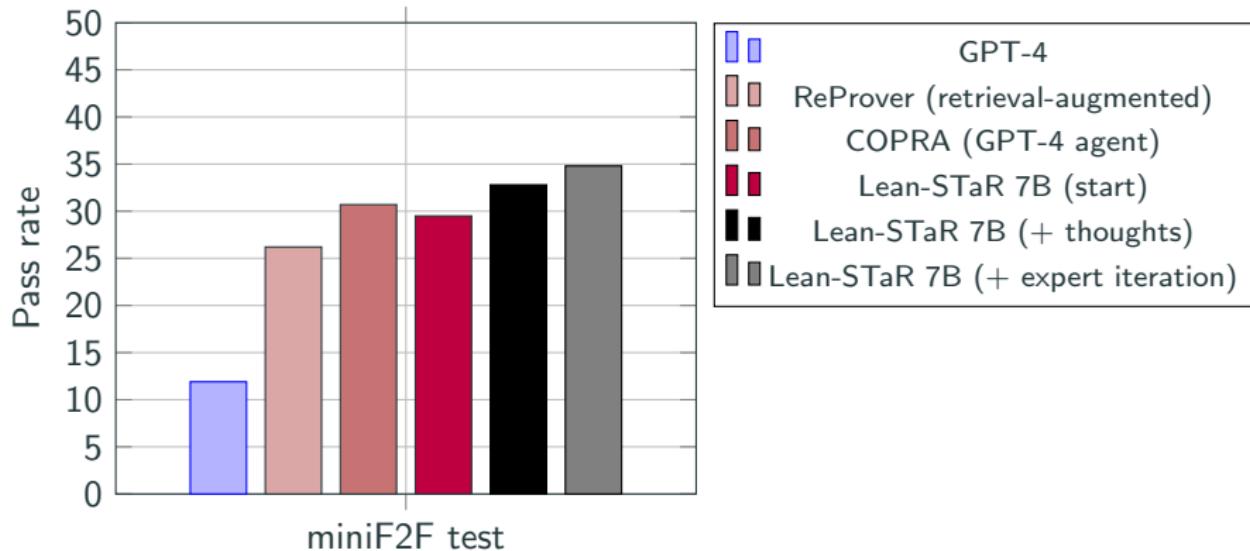
<sup>3</sup>I.e. Expert Iteration [Polu et al 2022], Rest-EM [Singh et al 2024]

# Training models to “think” — Lean-STaR

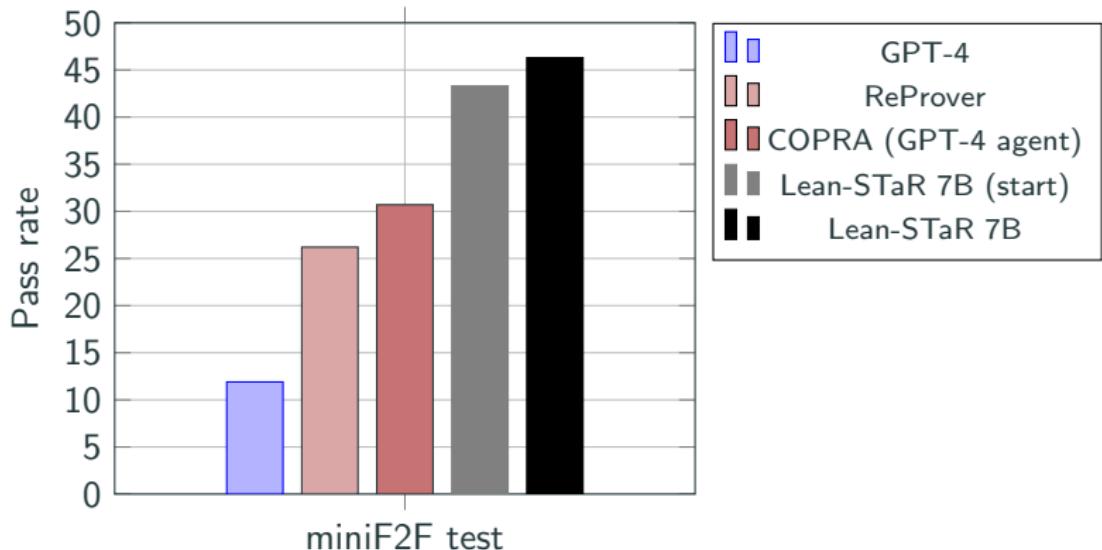
- **miniF2F**: competition problems (AMC, AIME, IMO)



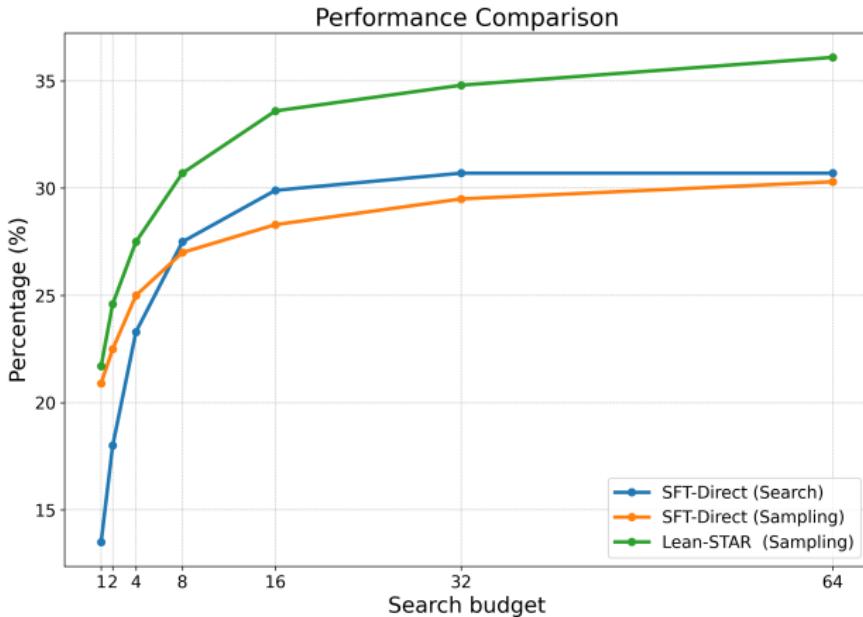
# Training models to “think” — Lean-STaR



## 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  $\lfloor n \rfloor \in \mathbb{N}$ 
such that  $\lfloor (n + 1) * p \rfloor \leq n + 1 + p$  holds for some  $\lfloor p \rfloor \in \mathbb{N}$ , we
recognize that setting  $\lfloor p = 0 \rfloor$  satisfies the inequality by the properties of
zero in arithmetic. This is because  $\lfloor n + 1 > 0 \rfloor$  by assumption, ensuring the
left-hand side is non-negative and the right-hand side simplifies to  $\lfloor n + 1 \rfloor$ ,
making the inequality trivially true.
```

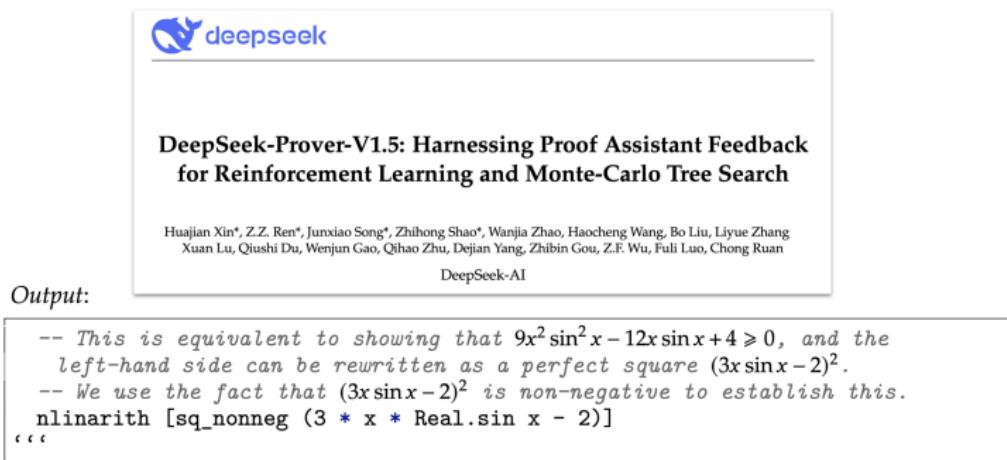
Example generated thoughts and proof from Lean-STaR

# Training models to “think” — Lean-STaR

```
theorem aimes_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 h₂ h₃ -- Reasoning: Given a function |f(x)| defined over the real
numbers, where |f(x)| equals the sum of the absolute values of the
differences |(x - p)|, |(x - 15)|, and |(x - p - 15)|, along with conditions
on |(p)| and |(x)| that ensure |(x)| lies between |(p)| and |(15)| and |(x)| is
non-positive, we observe that |f(x)| ≥ |(p)| (geq 15). To demonstrate this, we
consider all possible cases for the possible values of |(x)| relative to |(p)| and
|(15)|, specifically at |(p)| itself, between |(p)| and |(15)|, and at |(15)|.
By evaluating |f(x)| for each case, we can establish a lower bound of
|f(x)| that is greater than or equal to |(15)|.
simp [h₀, abs_of_nonneg (sub_nonneg_2 h₁), abs_of_nonneg (sub_nonneg_2 h₂), *]
-- Reasoning: The objective is to show that |f(x)| is at least 15 for |(x)| within the bounds |(p| ≤ |x| ≤ |15)|, where |(p)| is positive but less than
|15|. The provided expression for |f(x)| suggests the properties of absolute
value and inequalities apply, and since |f(x)| ≥ |(p)|, we apply simplification
rules for absolute values and inequalities, along with the
given bounds, we aim to demonstrate the desired inequality |f(x)| ≥ |(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 |(x)|. 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
expression to align both sides of the inequality.
simp only [abs_of_nonneg, sub_add_eq_sub_sub] at * -- Reasoning: The objective
is to demonstrate that the expression "15 ≤ x - p + |x - 15| + |x - p - 15|"
holds given the conditions on |p|, |x|, and the function |f|. 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 on both sides.
congr 1 -- Reasoning: To establish the inequality "15 ≤ x - |x - 15| + |x - p - 15|
- 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_nonneg (sub_nonneg_of_le h₂), abs_of_nonneg (sub_nonneg_of_le h₃)]
-- Reasoning: The objective is to demonstrate the 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 recognising the
algebraic manipulation and cancellation of the known bounds "h₁" and "h₂", as
well as the non-negativity of "p".
rw [abs_of_nonpos] -- Reasoning: The objective is to demonstrate that "15 ≤ 15
+ |x - p - 15| - p" under the given constraints on |p| and |x|. Given "p < 15"
and "p ≤ x ≤ 15", it's apparent that "p - 15" 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.
abs_geq_lemma -- Reasoning: This lemma provides a simple way of 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.
```

# Training models to “think” — After Lean-STaR

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



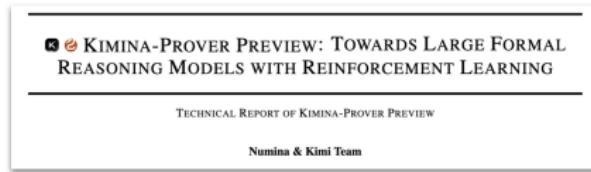
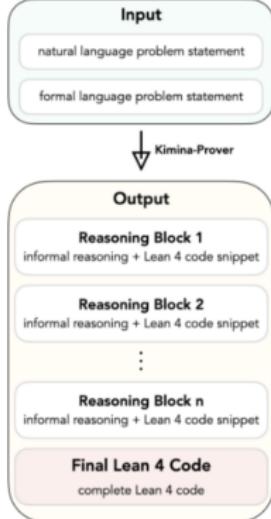
The screenshot shows the DeepSeek interface. At the top left is the DeepSeek logo (a blue stylized 'd' icon) and the word "deepseek". Below it is a horizontal line. In the center, the title "DeepSeek-Prover-V1.5: Harnessing Proof Assistant Feedback for Reinforcement Learning and Monte-Carlo Tree Search" is displayed. Below the title, the authors' names are listed: 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. Underneath the author list is the text "DeepSeek-AI". A large text box contains the proof code, which includes comments explaining the steps. The comments are:  
-- 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:



Next, we'll substitute  $d = \frac{15}{2}$  back into one of the equations to solve for  $a$ . Let's use  $h_0$ :  $a + 6d = 30$

$$a + 6\left(\frac{15}{2}\right) = 30$$

$$a + 45 = 30$$

$$a = 30 - 45 = -15$$

In Lean, once we have  $d$ , we can determine  $a$ :

```
have ha : a = -15 := by  
linarith [h₀, hd]
```

Now we need to compute  $a + 20d$ :  $a + 20d = -15 + 20\left(\frac{15}{2}\right) = -15 + 150 = 135$

In Lean, we can directly compute  $a + 20d$  using the values of  $a$  and  $d$ :

```
linarith [ha, hd]
```

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:

September 12, 2024 Release

## Learning to reason with LLMs

We are introducing OpenAI o1, a new large language model trained with reinforcement learning to perform complex reasoning. It thinks before it answers —it can produce a long internal chain of thought before responding to the user.

OpenAI o1 reasoning model



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

DeepSeek-AI  
research@deepseek.com

DeepSeek R1 reasoning model

# Training models to “think” — Lean-STaR

## 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

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



**Sledgehammer** [Paulson 2010] calls out to external automated provers.

- First-order logic, higher-order logic, SMT

# Low-level provers: Sledgehammer

```
theorem algebra.abs_id_left_abs_left[sledgehammer]:  
  fixes x :: real  
  shows "abs (x + 1) = x + 2"  
  shows "0 < 0"
```

Theorem



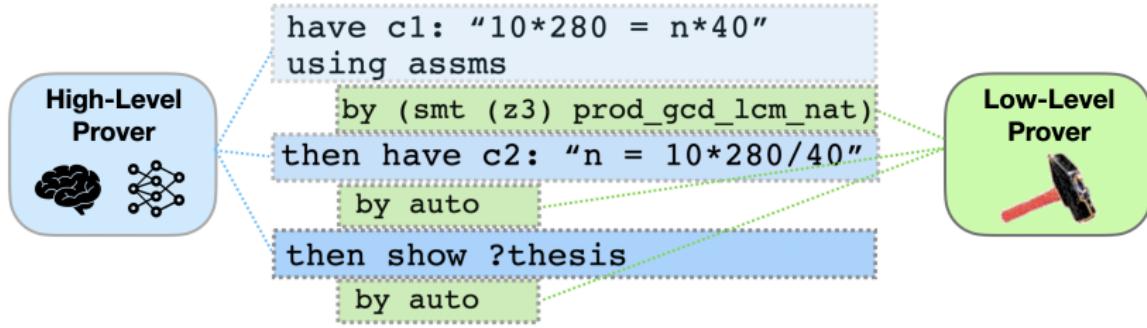
```
theorem algebra.abs_id_left_abs_left[sledgehammer]:  
  fixes x :: real assumes "0 < x - 1" + abs x + abs (x + 1) = x + 2"  
  shows "0 < x" + abs x + abs (x + 1) = x + 2"  
proof -  
  have c1: "x > 0" by linarith  
  have c2: "x > 0" using c1 by auto  
  assume c3: "x < 0" assumes c4: "x < 0"  
  have c5: "abs(x-1) + abs x + abs(x+1) = -(x-1) - x + (x+1)" using c1 c2  
  have c6: "0 < 0" using c3 c4 by auto  
  then have c7: "x = 0" by auto  
  then show ?thesis using c5 c6 by auto  
qed  
have c1: "x < 1" by linarith  
have c2: "x < 0" using c1 by auto  
assume c3: "x < 0" assumes c4: "x < 0"  
have c5: "abs(x-1) + abs x + abs(x+1) = 2-x" by auto  
have c6: "2-x = x" using c3 c4 by auto  
then have c7: "x = 0" using c5 c6 by auto  
then show ?thesis using c7 by auto  
qed  
have c1: "x > 1" by linarith  
have c2: "x < 0" using c1 by auto  
assume c3: "x < 0" assumes c4: "x < 0"  
have c5: "abs(x-1) + abs x + abs(x+1) = x-1 + x + (x+1)" using c1 c2  
have c6: "x-1 + x + (x+1) = 3*x" by auto  
then have c7: "3*x = x" using c3 c4 by auto  
then have c8: "x = 0" using c5 c6 by auto  
then show ?thesis using c8 by auto  
qed  
then show ?thesis using c5 c6 c7 c8 by fastforce  
qed
```

complex proof

Sledgehammer [Paulson 2010]

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)*

# Draft-Sketch-Prove

---

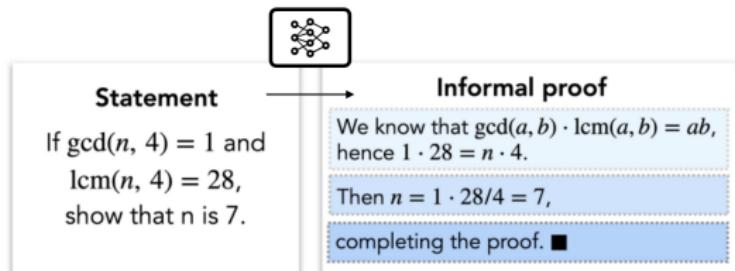
## Idea:

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

# Draft, Sketch, Prove

Given informal theorem  $x_I$ ,  
formal theorem  $x_F$

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

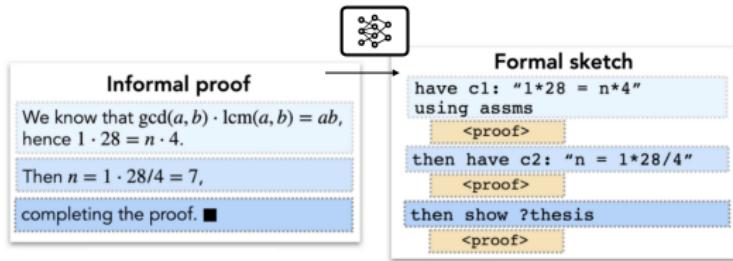


Human-written or LLM-generated draft

# Draft, Sketch, Prove

Given informal theorem  $x_I$ ,  
formal theorem  $x_F$

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

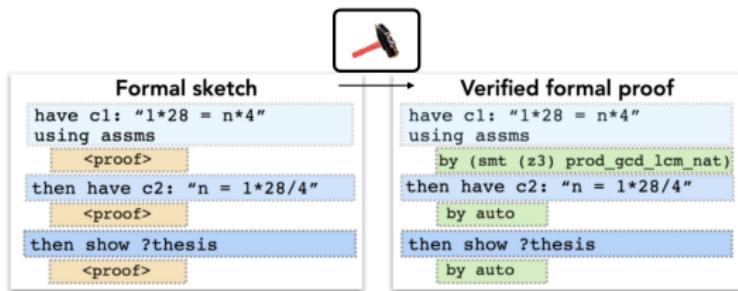


LLM-generated sketch

# Draft, Sketch, Prove

Given informal theorem  $x_I$ ,  
formal theorem  $x_F$

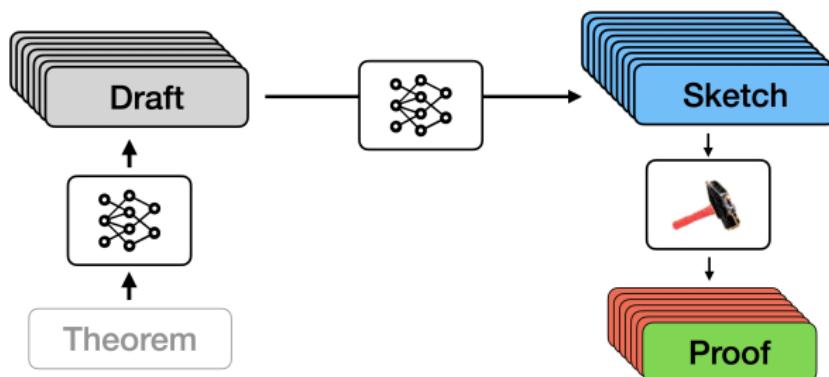
1. Draft  $y_I \sim p(\cdot | x_I)$
2. Sketch  $z_F \sim p(\cdot | x_F, x_I, y_I)$
3. **Prove**  $y_F = f(x_F, z_F)$



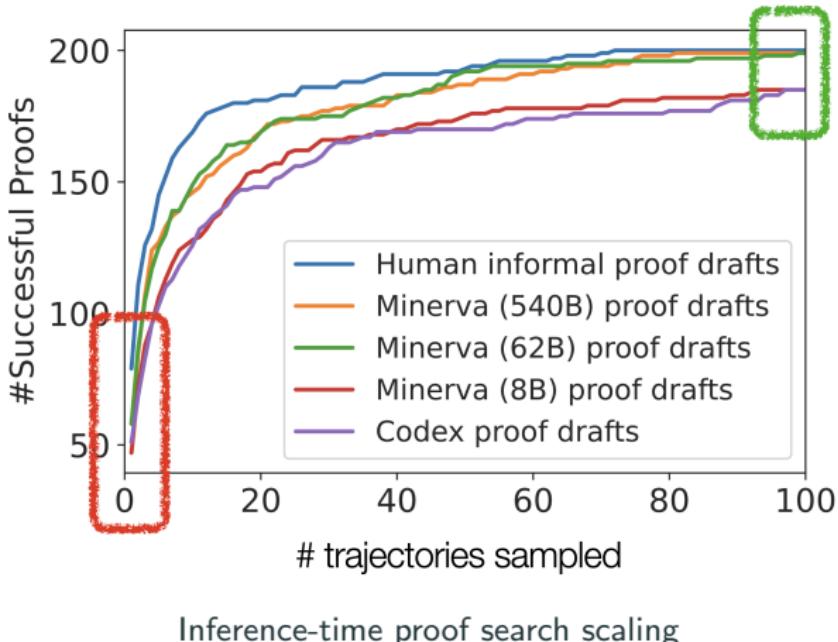
Low-level prover (*Sledgehammer*) fills in the gaps

# Draft-sketch-prove

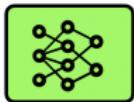
## Proof search



# Draft-sketch-prove



# Draft-sketch-prove



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

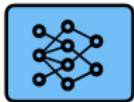
**Informal Proof (Minerva 62B):**

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

The Euclidean algorithm gives

$$\begin{aligned}21n + 4 &= 1 \cdot (14n + 3) + 7n + 1 \\14n + 3 &= 2 \cdot (7n + 1) + 1.\end{aligned}$$

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_numerals_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 interface with the following details:

- Title Bar:** Shows the notebook title "II\_dsp\_\_part2\_dsp.ipynb" and a tab labeled "Markdown".
- Toolbar:** Includes standard notebook controls like back/forward, search, and file operations.
- Code Cell:** Contains Python code for a function `dsp`:

```
#### 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
```
- Text Cell:** A block of text explaining the next steps:

Next, we will discuss how to implement these three modules.
- Text Cell:** Another block of text introducing the [Isabelle proof assistant](#):

We start by introducing the [Isabelle proof assistant](#), since it is relevant to the implementation.
- Right Panel:** Shows navigation icons for the notebook.

Demo notebook: [github.com/cmu-l3/ntptutorial-II](https://github.com/cmu-l3/ntptutorial-II)

### 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

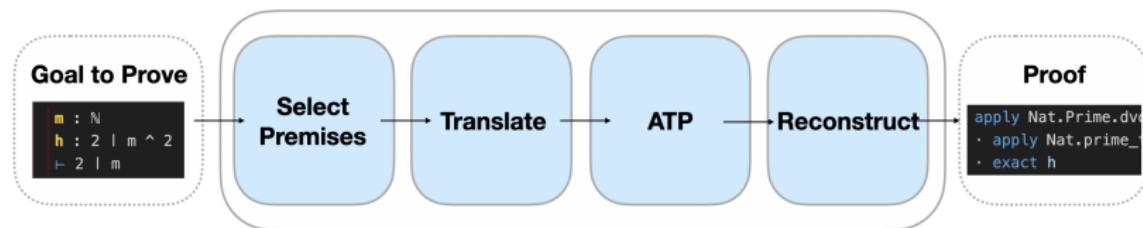
---

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

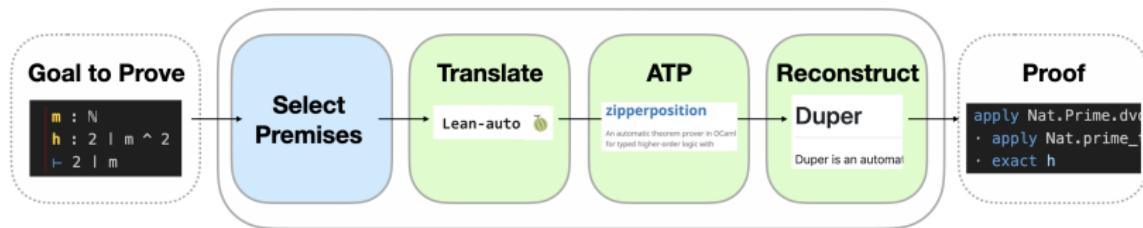
# A hammer pipeline

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:

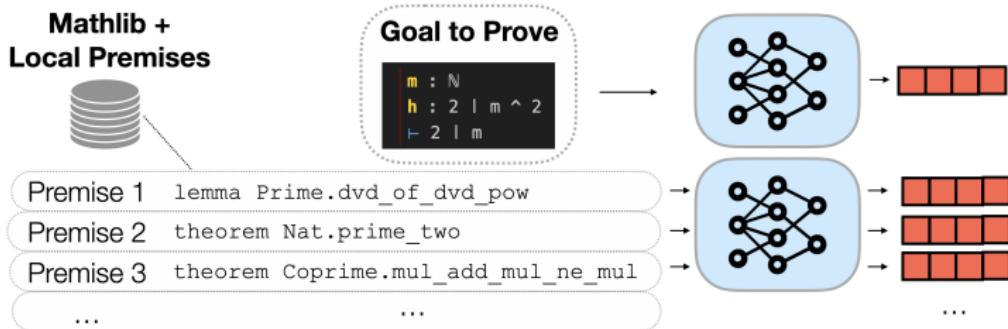


Our challenge:

- Premise selection
- Put it all together to create LeanHammer

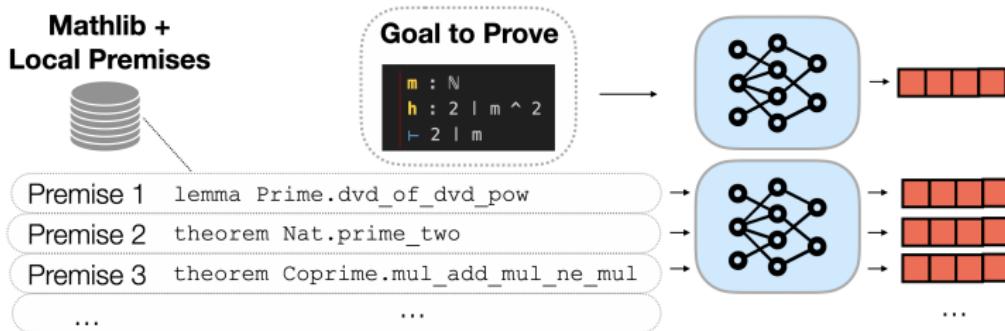
# LeanHammer — Neural premise selection

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



# LeanHammer — Neural premise selection

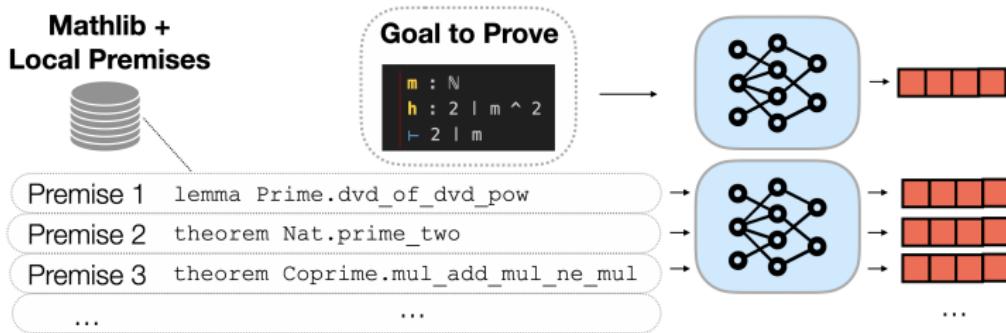
Idea: frame premise selection as retrieval with a neural language model



- Transformer encoder embeds the state and candidate premises

# LeanHammer — Neural premise selection

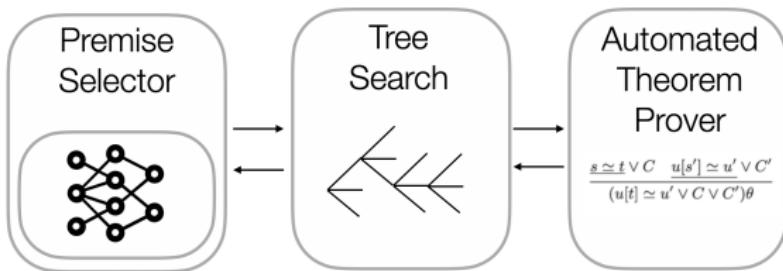
Idea: frame premise selection as retrieval with a neural language model



- 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

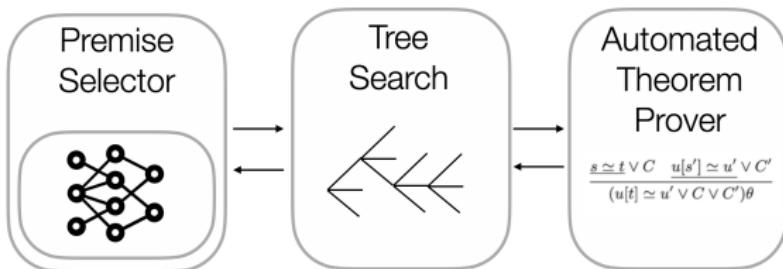
Idea: combine the premise selector and ATP with a tree search



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

# LeanHammer — Putting it all together

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

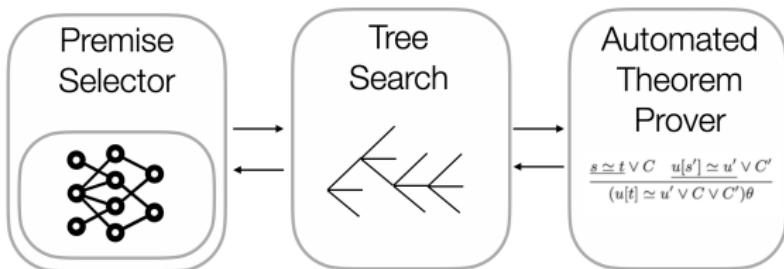


Tree search: *Aesop* [Limperg & From 2023]

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
```

# LeanHammer — Demo

Demo: fill in the gaps (sorrrys) with LeanHammer

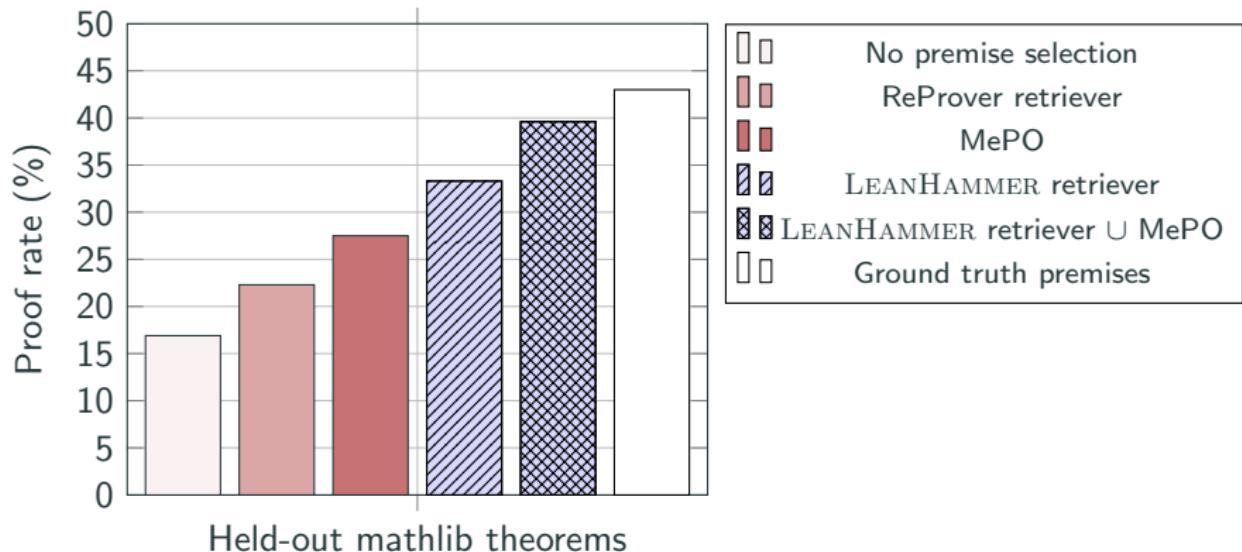
Human  
Written



```
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
  apply Lemma.two_dvd_of_two_dvd_sq
  simp_all only [dvd_mul_right]
obtain ⟨k, meq⟩ := dvd_ifd_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

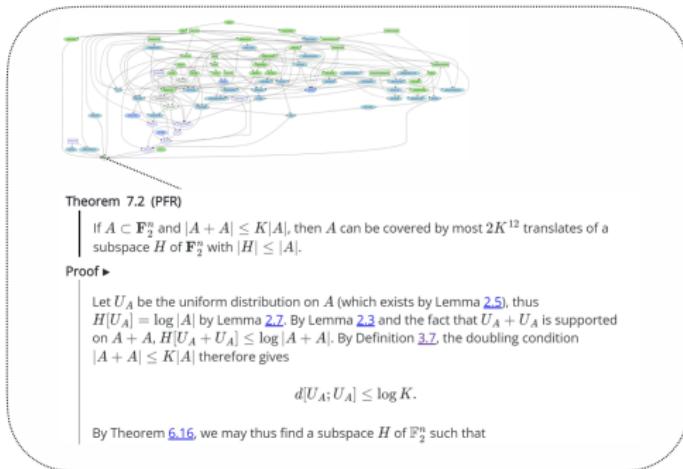
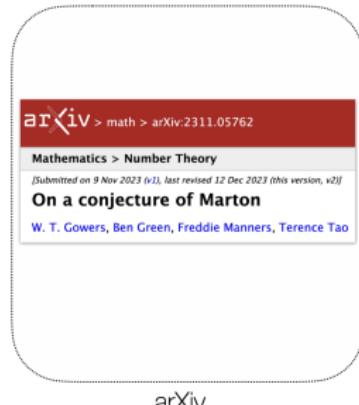
Even small neural networks are powerful! (retriever: < 100M params)

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

### **III: Research-level mathematics**

---

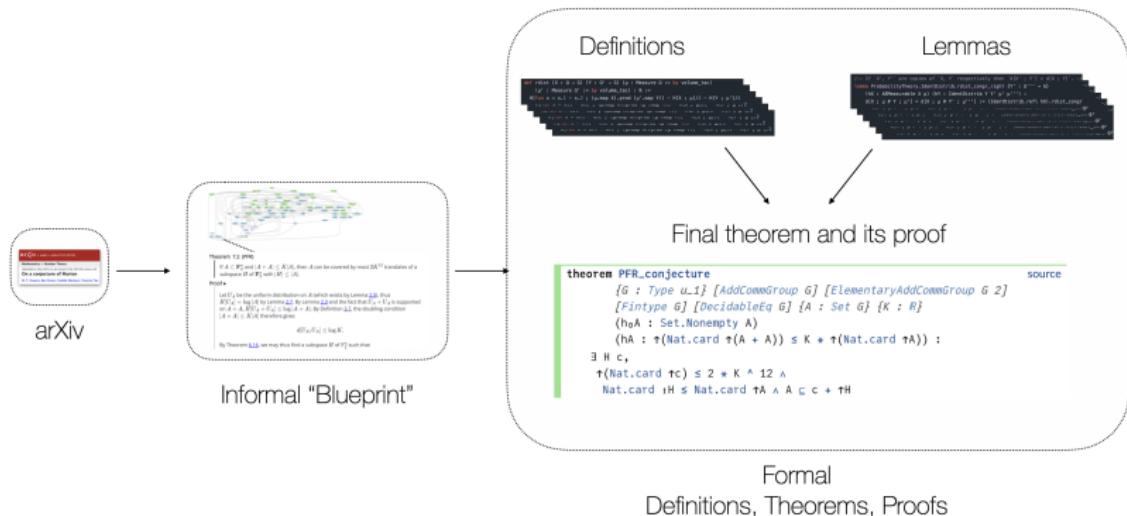
# What does it look like to formalize research-level math?<sup>4</sup>



Informal “Blueprint”

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

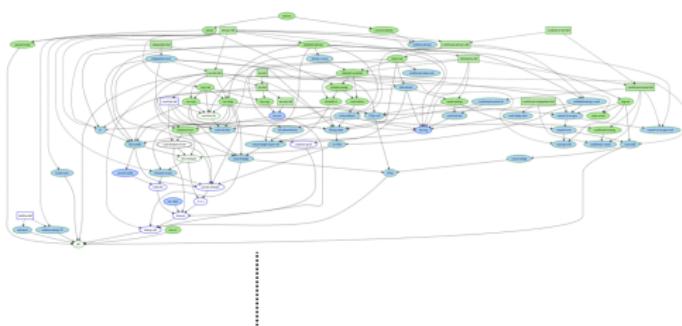
# What does it look like to formalize research-level math?<sup>4</sup>



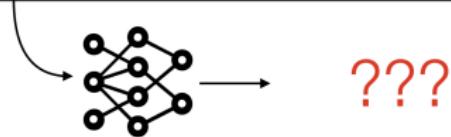
<sup>4</sup> Formalizing the proof of PFR in Lean4 using Blueprint: a short tour by Terence Tao

# Where can AI help?

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



```
-- `H[X | Y=y] = ∑_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] = ∑' 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, ...)

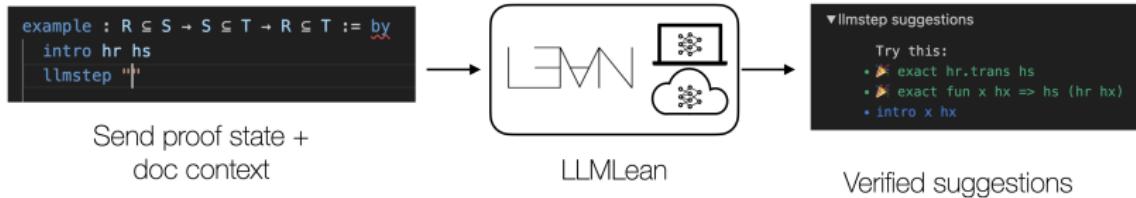
# Where can AI help? — Challenges

## Accessibility gap:

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

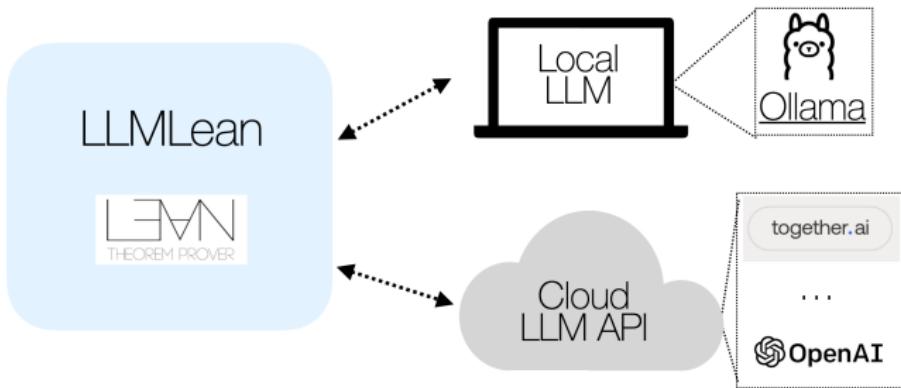
However, there **are** already tools available!

# Where can AI help? — Existing tools



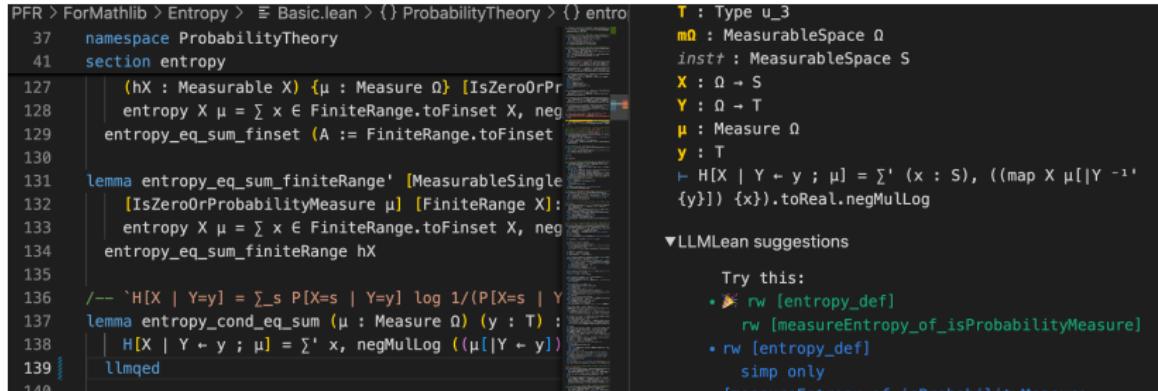
**LLMLean:** <https://github.com/cmu-l3/llmlean>

## 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 > entropy

```
37 namespace ProbabilityTheory
41 section entropy
127   (hX : Measurable X) {μ : Measure Ω} [IsZeroOrProbabilityMeasure μ]
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=y])
137   lemma entropy_cond_eq_sum (μ : Measure Ω) (y : T) :
138     H[X | Y = y ; μ] = ∑' x, negMulLog ((μ[|Y = y]) )
139   llimqed
```

T : Type u\_3  
m $\Omega$  : MeasurableSpace Ω  
inst $t$  : MeasurableSpace S  
X : Ω → S  
Y : Ω → T  
μ : Measure Ω  
y : T  
 $\vdash H[X \mid Y = y ; \mu] = \sum' (x : S), ((\text{map } X \mu[|Y^{-1}(y)]) \{x\}).\text{toReal}.\text{negMulLog}$

▼ LLMLean suggestions

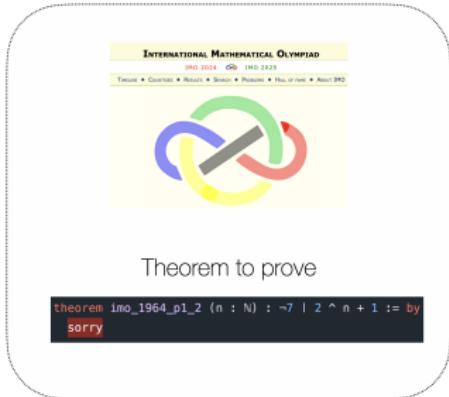
Try this:

- 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

# 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
```

## Real projects



Definitions

```
def entropy [measurable_space α] (μ : measure[α]) : ℝ :=  
  -μ.map log (μ.map (λ x, μ{!x}))
```

Lemmas

```
lemma entropy_eq_sum (μ : measure[α]) (y : T) :  
  H[X | Y = y ; μ] = ∑' x, negMulLog ((μ[Y = y]).map X {x}).toReal := by
```

Theorem to prove

```
/-- `H[X | Y=y] = ∑ s P[X=s | Y=y] log 1/(P[X=s | Y=y])` . --/  
lemma entropy_cond_eq_sum (μ : measure[α]) (y : T) :  
  H[X | Y = y ; μ] = ∑' x, negMulLog ((μ[Y = y]).map X {x}).toReal := by
```

- Self-contained
- Uses standard results
- Part of a project
- Uses new definitions and lemmas

## miniCTX: Neural Theorem Proving with (Long-)Contexts

Jiewen Hu, Thomas Zhu, Sean Welleck

*ICLR 2025* (**Oral**)

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

- (context, theorem) → proof
  - Context: repository of code, new definitions, auxiliary lemmas

## miniCTX:

Test models on real Lean projects:<sup>5</sup>

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

---

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

## miniCTX:

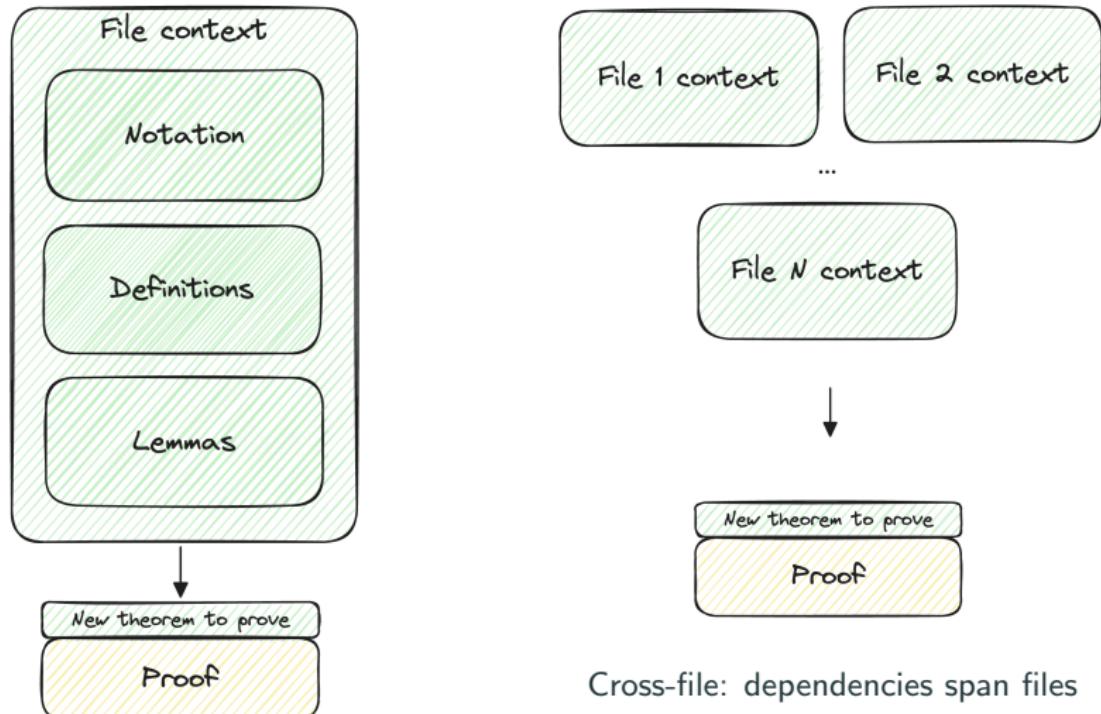
Test models on real Lean projects:<sup>5</sup>

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

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

---

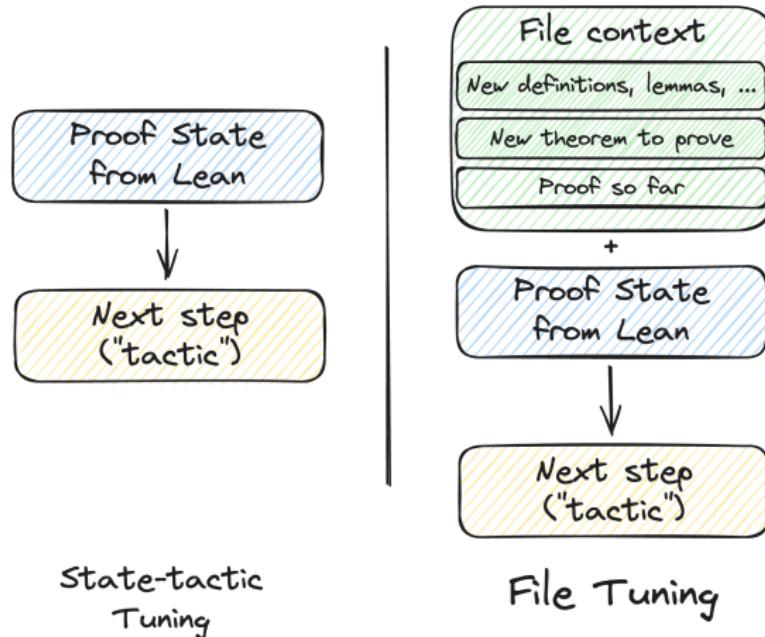
<sup>5</sup>+ tools for easily adding new projects: <https://github.com/cmu-l3/ntp-toolkit>



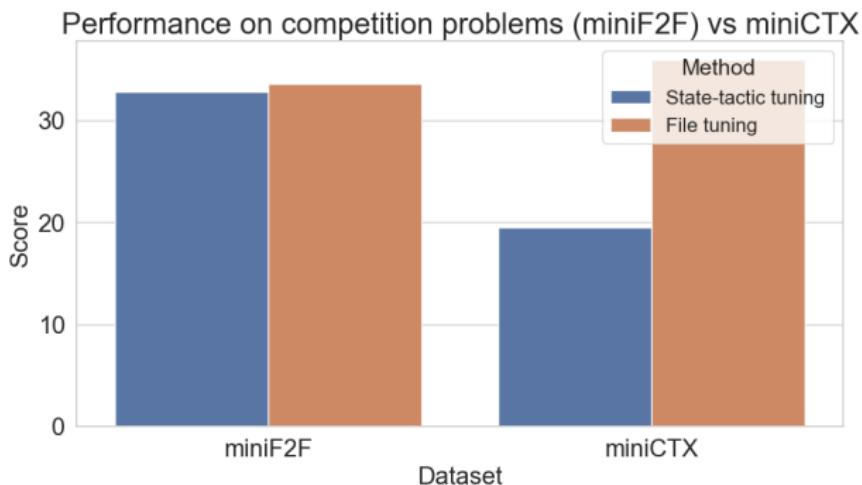
In-file: dependencies appear in the file  
110 / 118

# miniCTX — preceding code context

Does context actually matter? A simple experiment.



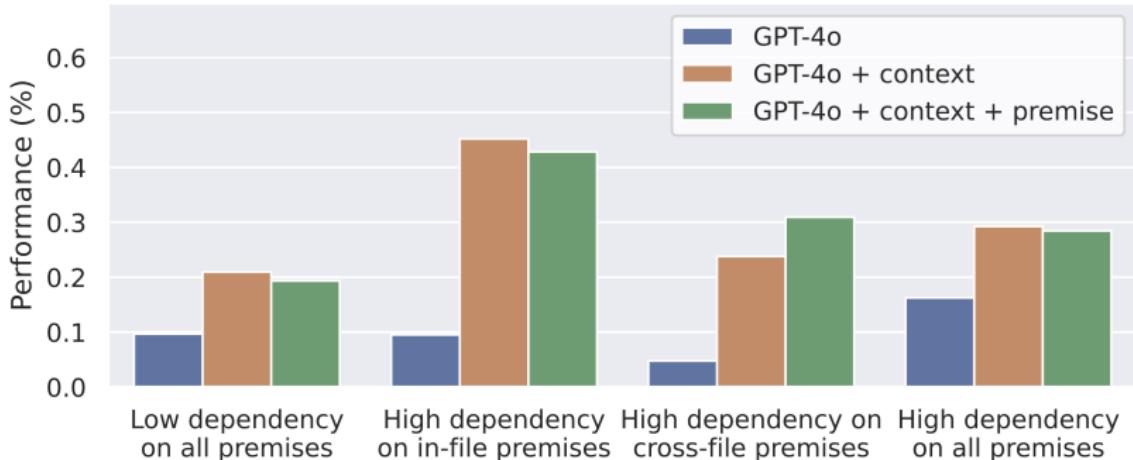
“File tuning”: train on (preceding code, state, next-tactic) examples



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

zy

Performance on miniCTX theorems by dependency level



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-13/ntp-toolkit>
- Evaluation: <https://github.com/cmu-13/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`