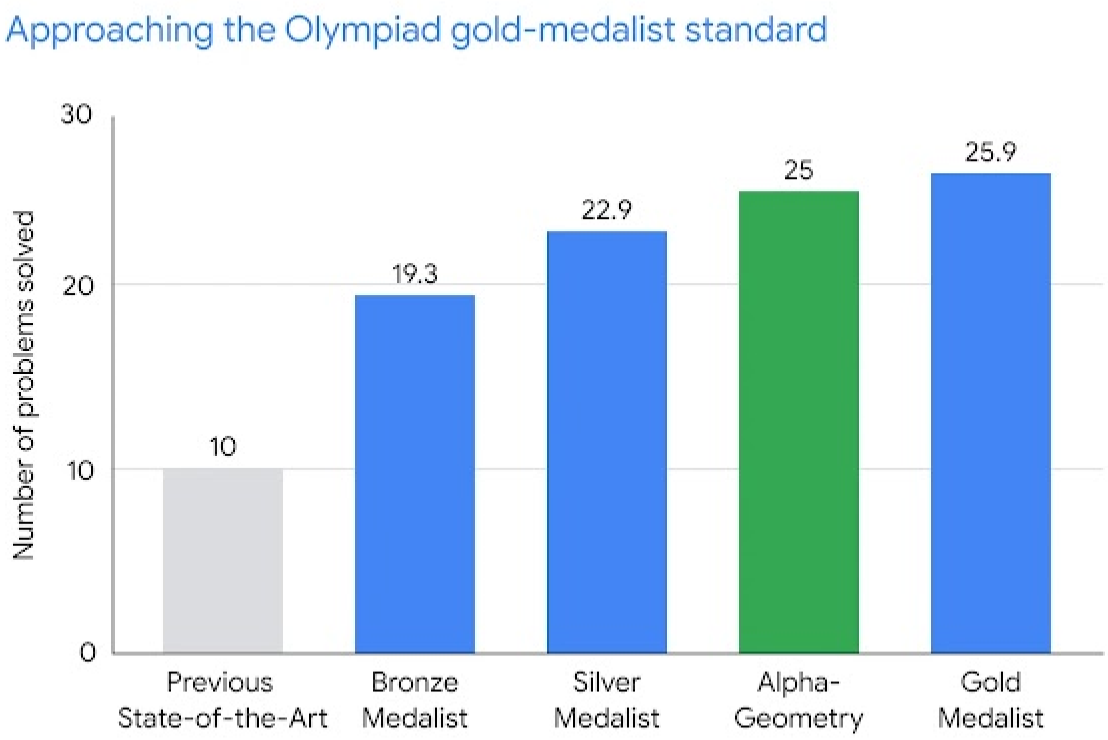
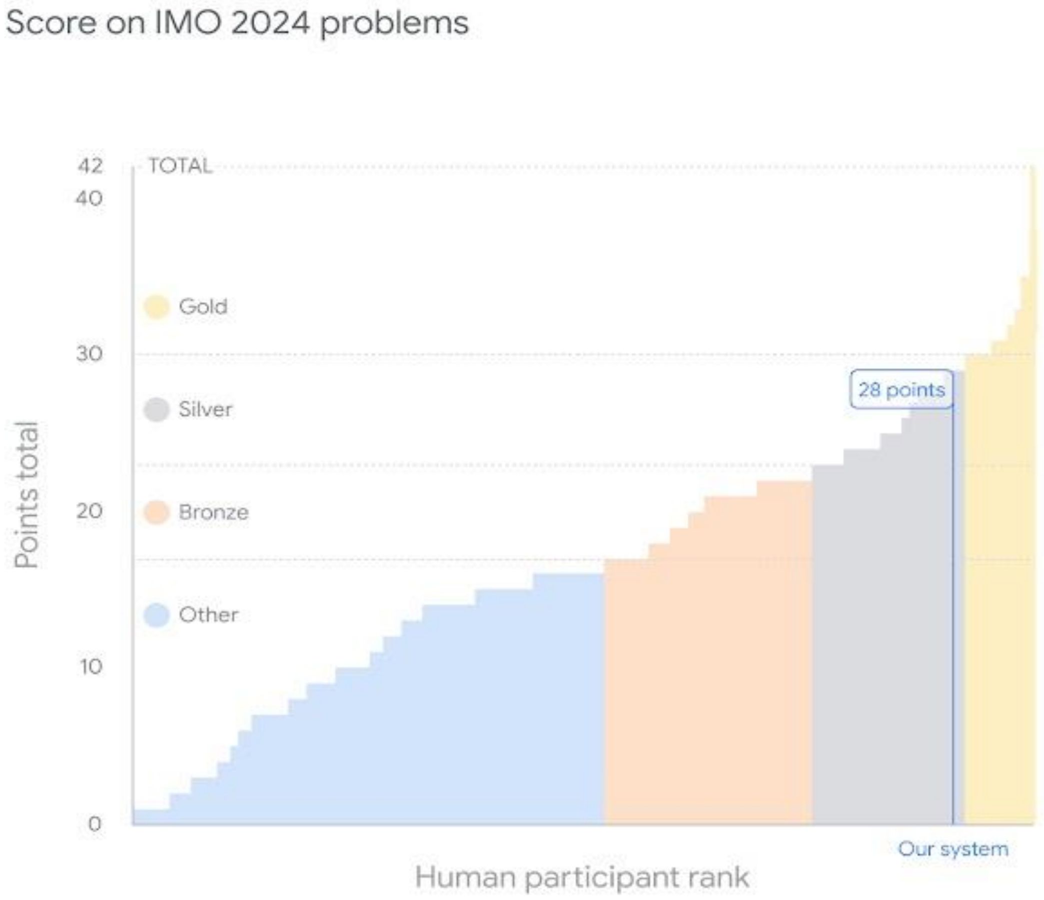
Neuro-Symbolic Theorem Proving with Lean

Peiyang Song
California Institute of Technology
psong@caltech.edu

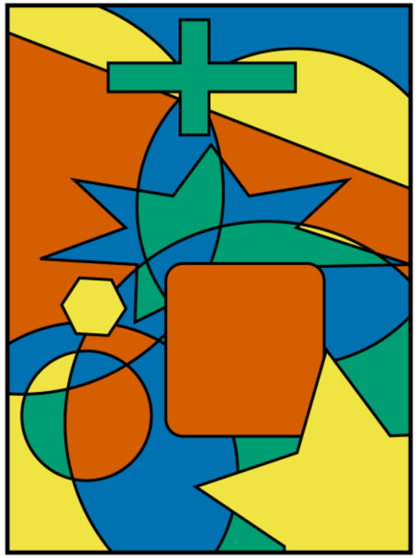
Alpha Geometry 2

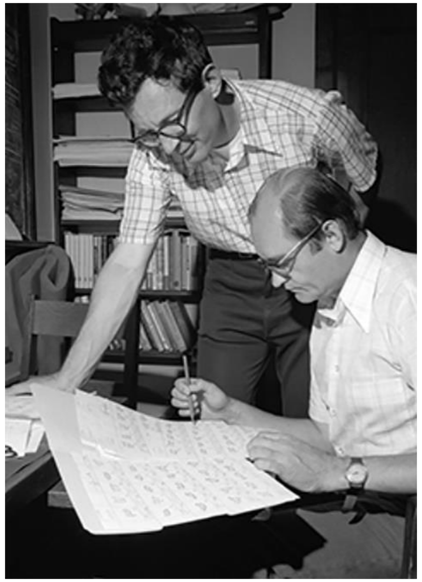


Alpha Proof



Computer-Aided Proofs in Mathematics

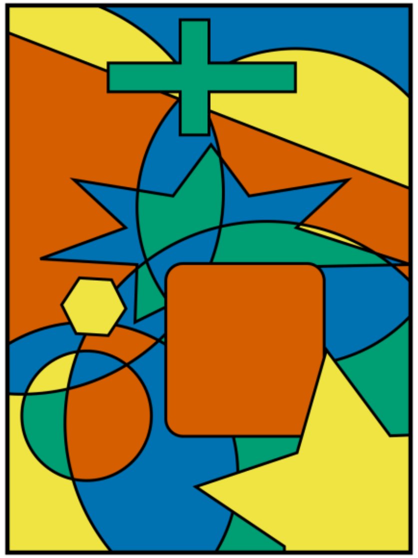


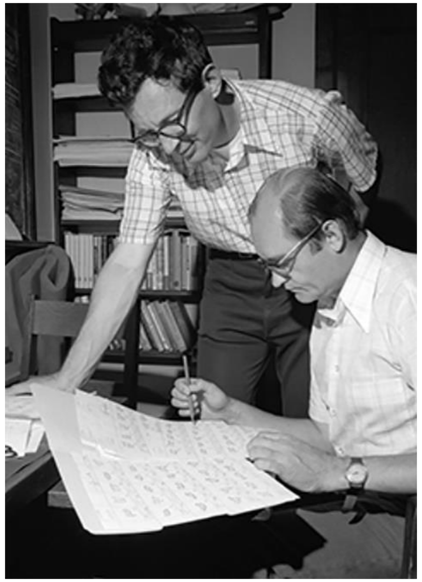


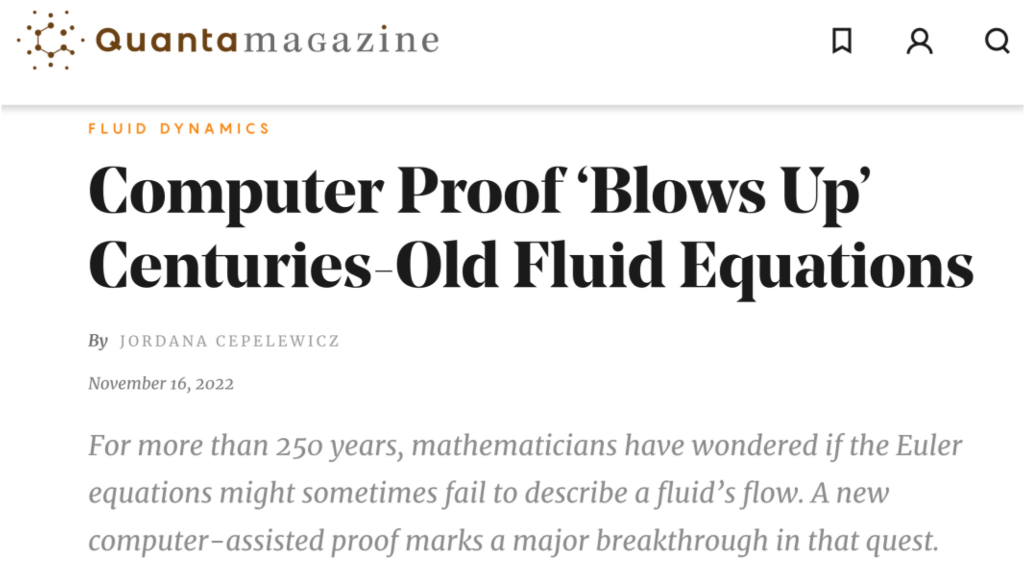
Four Color Theorem
Computers check 1000+ configurations

[Appel and Haken, "Every Planar Map Is Four Colorable", 1976]

Computer-Aided Proofs in Mathematics






Quantamagazine

FLUID DYNAMICS

Computer Proof 'Blows Up' Centuries-Old Fluid Equations

By JORDANA CEPELEWICZ

November 16, 2022

For more than 250 years, mathematicians have wondered if the Euler equations might sometimes fail to describe a fluid's flow. A new computer-assisted proof marks a major breakthrough in that quest.

Four Color Theorem

Computers check 1000+ configurations

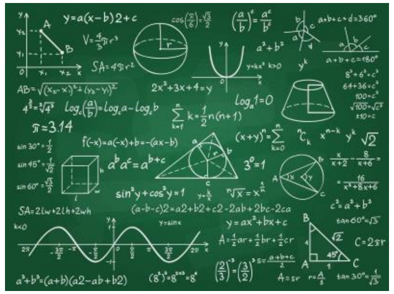
[Appel and Haken, "Every Planar Map Is Four Colorable", 1976]

Blowup of the Euler Equations

Computers calculate bounds of integrals

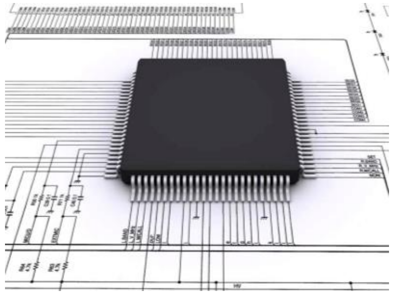
[Chen and Thomas, "Stable Nearly Self-similar Blowup Of The 2D Boussinesq And 3D Euler Equations With Smooth Data", 2022]

Automated Reasoning and Formal Proofs



Formal mathematics

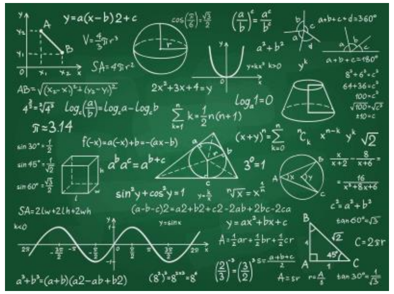
Software verification



Hardware verification

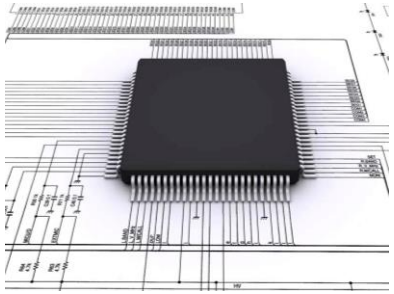
Cyber-physical systems

Automated Reasoning and Formal Proofs



Formal mathematics

Software verification



Hardware verification

Cyber-physical systems

Automated Reasoning and Formal Proofs

- **Automated theorem proving**
  - SMT solvers, model checkers, ATP systems in first-order logic, etc.
  - Minimal efforts from humans
  - Limited expressiveness
  - Difficult to scale

Automated Reasoning and Formal Proofs

- **Interactive theorem proving**
  - Proof assistants such as Coq, Isabelle, and Lean
  - Expressive logic, e.g., dependent type theory
  - Successfully used in large formalization projects
  - Lots of efforts from humans to write proofs

[Hales et al., "A Formal Proof of the Kepler Conjecture", 2017]
[Klein et al., "seL4: Formal Verification of an OS Kernel", 2009]
[Leroy, "Formal Verification of a Realistic Compiler", 2008]

Automated Reasoning and Formal Proofs

- **Interactive theorem proving**
  - Proof assistants such as Coq, Isabelle, and Lean
  - Expressive logic, e.g., dependent type theory
  - Successfully used in large formalization projects
  - **Lots of efforts from humans to write proofs**
  - **Proof automation is critical for wider adoption**

[Hales et al., "A Formal Proof of the Kepler Conjecture", 2017]
[Klein et al., "seL4: Formal Verification of an OS Kernel", 2009]
[Leroy, "Formal Verification of a Realistic Compiler", 2008]

Automated Reasoning and Formal Proofs

- **Interactive theorem proving**
  - Proof assistants such as Coq, Isabelle, and Lean
  - Expressive logic, e.g., dependent type theory
  - Successfully used in large formalization projects
  - **Lots of efforts from humans to write proofs**
  - **Proof automation is critical for wider adoption**

- **Automated theorem proving**
  - SMT solvers, model checkers, ATP systems in first-order logic, etc.
  - Minimal efforts from humans
  - Limited expressiveness
  - Difficult to scale

Automated Theorem Proving

$$1 + 2 + \cdots + n = \frac{(n+1)n}{2}$$

- Generate the proof fully automatically

Automated Theorem Proving

$$1 + 2 + \cdots + n = \frac{(n+1)n}{2}$$

$$\neg q \vee p \vee \neg r$$

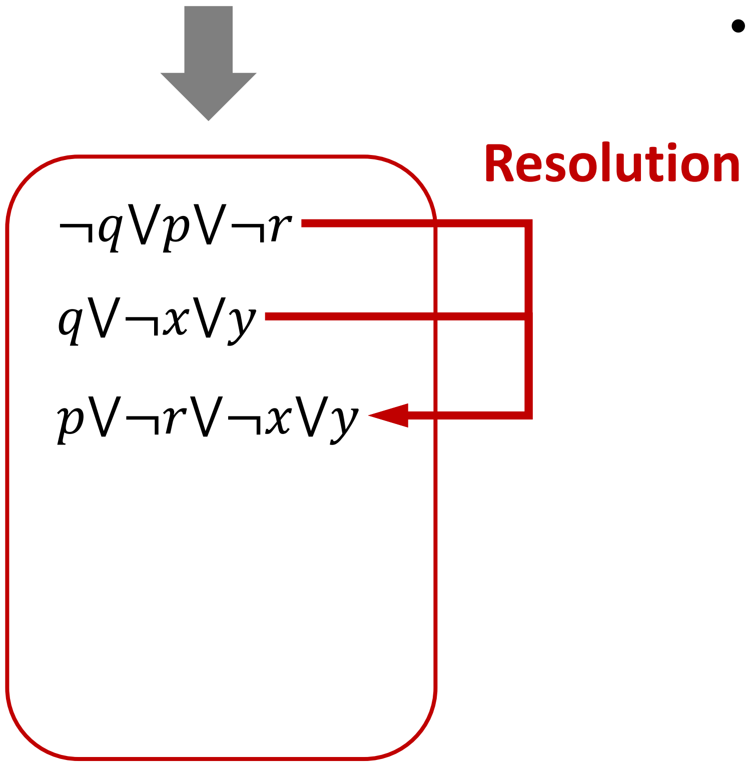
$$q \vee \neg x \vee y$$

Conjunctive normal form (CNF)

- Generate the proof fully automatically
- Low-level: First-order logic, CNFs, and resolution

Automated Theorem Proving

$$1 + 2 + \cdots + n = \frac{(n+1)n}{2}$$



Conjunctive normal form (CNF)

- Generate the proof fully automatically
- Low-level: First-order logic, CNFs, and resolution

Automated Theorem Proving

$$1 + 2 + \cdots + n = \frac{(n+1)n}{2}$$

$\neg q \vee p \vee \neg r$

$q \vee \neg x \vee y$

$p \vee \neg r \vee \neg x \vee y$

…

Conjunctive normal form (CNF)

- Generate the proof fully automatically
- Low-level: First-order logic, CNFs, and resolution

Automated Theorem Proving

$$1 + 2 + \cdots + n = \frac{(n+1)n}{2}$$

$\neg q \vee p \vee \neg r$

$q \vee \neg x \vee y$

$p \vee \neg r \vee \neg x \vee y$

…

…

Conjunctive normal form (CNF)

- Generate the proof fully automatically
- Low-level: First-order logic, CNFs, and resolution

Automated Theorem Proving

$$1 + 2 + \cdots + n = \frac{(n+1)n}{2}$$

$\neg q \vee p \vee \neg r$

$q \vee \neg x \vee y$

$p \vee \neg r \vee \neg x \vee y$

…

…

$\emptyset$

Conjunctive normal form (CNF)

- Generate the proof fully automatically
- Low-level: First-order logic, CNFs, and resolution

Automated Theorem Proving

$$1 + 2 + \cdots + n = \frac{(n+1)n}{2}$$

$\neg q \vee p \vee \neg r$

$q \vee \neg x \vee y$

$p \vee \neg r \vee \neg x \vee y$

…

…

$\emptyset$

Conjunctive normal form (CNF)

- Generate the proof fully automatically
- Low-level: First-order logic, CNFs, and resolution
- Main challenge: Large search space

[Haken, "The Intractability of Resolution", Theoretical Computer Science, 1985]

- Heuristics for pruning the search space

[Kovács and Voronkov, CAV 2013]
[Schulz et al. CADE 2019].
[Korovin, IJCAR 2008]
[Urban et al. TABLEAUX 2011]
[Loos et al. LPAR-21]
[Kaliszyk et al. NeurIPS 2018]

- Successful examples: Robbins Conjecture

[McCune, "Solution of the Robbins Problem", 1997]

- Intractable for most theorems

Automated Theorem Proving

$$1 + 2 + \cdots + n = \frac{(n+1)n}{2}$$

$\neg q \vee p \vee \neg r$

$q \vee \neg x \vee y$

$p \vee \neg r \vee \neg x \vee y$

…

…

$\emptyset$

Conjunctive normal form (CNF)

- Generate the proof fully automatically
- Low-level: First-order logic, CNFs, and resolution
- Main challenge: Large search space

[Haken, "The Intractability of Resolution", Theoretical Computer Science, 1985]

- Heuristics for pruning the search space

[Kovács and Voronkov, CAV 2013] [Urban et al. TABLEAUX 2011]
[Schulz et al. CADE 2019]. [Loos et al. LPAR-21]
[Korovin, IJCAR 2008] [Kaliszyk et al. NeurIPS 2018]

- Successful examples: Robbins Conjecture

[McCune, "Solution of the Robbins Problem", 1997]

- Intractable for most theorems in math
- **Lack high-level intuitions of mathematicians**

Interactive Theorem Proving

Theorem

⬇

Proof

Interactive Theorem Proving

Theorem

```
theorem set_inter_comm (s t : Set α) : s ∩ t = t ∩ s
```

⬇

Proof

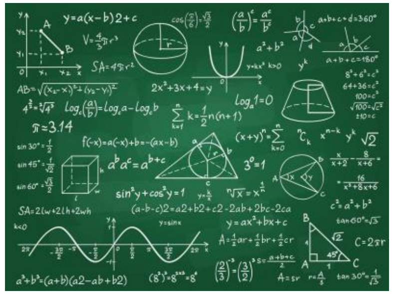
```
ext x
simp [Set.mem_inter_iff]
constructor
· rintro ⟨xs, xt⟩
  exact ⟨xt, xs⟩
. rintro ⟨xt, xs⟩
  exact ⟨xs, xt⟩
```

- Theorems/proofs represented formally as programs

[Hales et al., "A Formal Proof of the Kepler Conjecture", 2017]
[Leroy et al., "CompCert - A Formally Verified Optimizing Compiler", 2016]

Interactive Theorem Proving

Formalize



Mathematics

Software

Theorem

```
theorem set_inter_comm (s t : Set α) : s ∩ t = t ∩ s
```

Proof

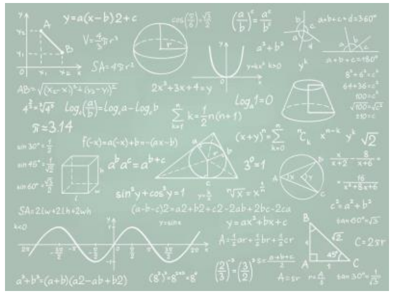
```
ext x
simp [Set.mem_inter_iff]
constructor
· rintro ⟨xs, xt⟩
  exact ⟨xt, xs⟩
. rintro ⟨xt, xs⟩
  exact ⟨xs, xt⟩
```

- Theorems/proofs represented formally as programs

[Hales et al., "A Formal Proof of the Kepler Conjecture", 2017]
[Leroy et al., "CompCert - A Formally Verified Optimizing Compiler", 2016]

Interactive Theorem Proving

Formalize



Mathematics

Software

Theorem

```
theorem set_inter_comm (s t : Set α) : s ∩ t = t ∩ s
```

Proof

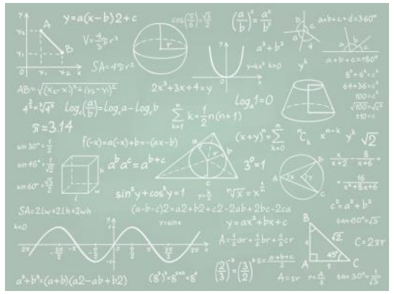
```
ext x
simp [Set.mem_inter_iff]
constructor
· rintro ⟨xs, xt⟩
  exact ⟨xt, xs⟩
. rintro ⟨xt, xs⟩
  exact ⟨xs, xt⟩
```

- Theorems/proofs represented formally as programs
- Proofs can be checked easily

[Hales et al., "A Formal Proof of the Kepler Conjecture", 2017]
[Leroy et al., "CompCert - A Formally Verified Optimizing Compiler", 2016]

Interactive Theorem Proving

Formalize



Mathematics

Software

Theorem

```
theorem set_inter_comm (s t : Set α) : s ∩ t = t ∩ s
```

Proof

```
ext x
simp [Set.mem_inter_iff]
constructor
· rintro ⟨xs, xt⟩
  exact ⟨xt, xs⟩
. rintro ⟨xt, xs⟩
  exact ⟨xs, xt⟩
```

✓

```
ext x
simp [Set.mem_inter_iff]
rintro ⟨xs, xt⟩
exact ⟨xt, xs⟩
```

✗

- Theorems/proofs represented formally as programs
- Proofs can be checked easily

Theorem Proving in Proof Assistants

Human

```
theorem gcd_self (n : nat) : gcd n n = n :=
```

n : ℕ
⊢ gcd n n = n

Proof assistant

Theorem Proving in Proof Assistants

Human

```
theorem gcd_self (n : nat) : gcd n n = n :=
begin
  cases n,
```

n : ℕ
⊢ gcd n n = n

cases n

⊢ gcd 0 0 = 0

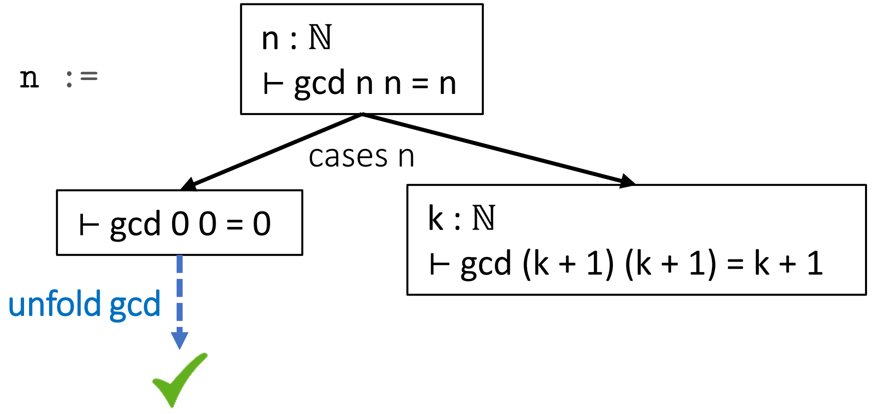
k : ℕ
⊢ gcd (k + 1) (k + 1) = k + 1

Proof assistant

Theorem Proving in Proof Assistants

Human

```
theorem gcd_self (n : nat) : gcd n n = n :=
begin
  cases n,
  { unfold gcd },
```

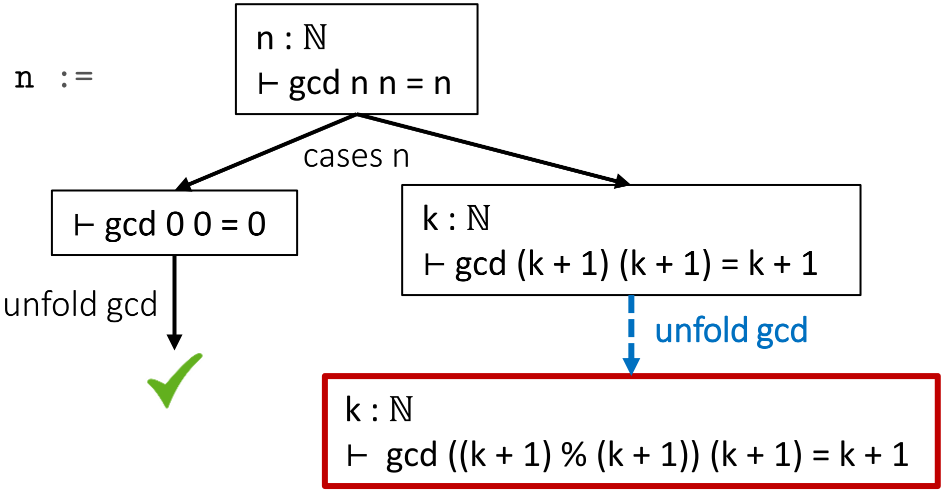


Proof assistant

Theorem Proving in Proof Assistants

Human

```
theorem gcd_self (n : nat) : gcd n n = n :=
begin
  cases n,
  { unfold gcd },
  unfold gcd,
```

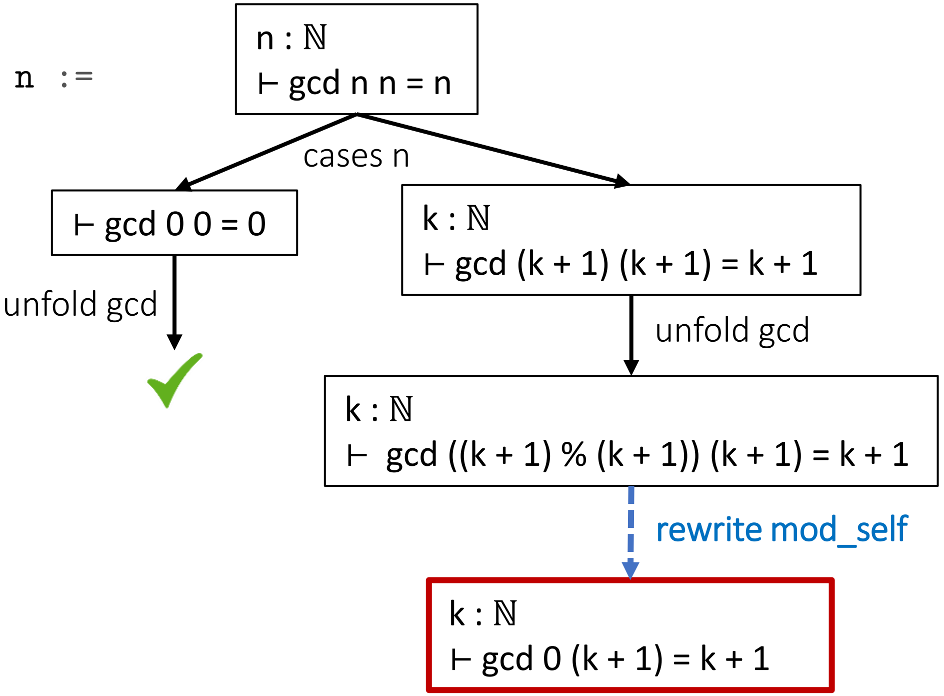


Proof assistant

Theorem Proving in Proof Assistants

Human

```
theorem gcd_self (n : nat) : gcd n n = n :=
begin
  cases n,
  { unfold gcd },
  unfold gcd,
  rewrite mod_self,
```

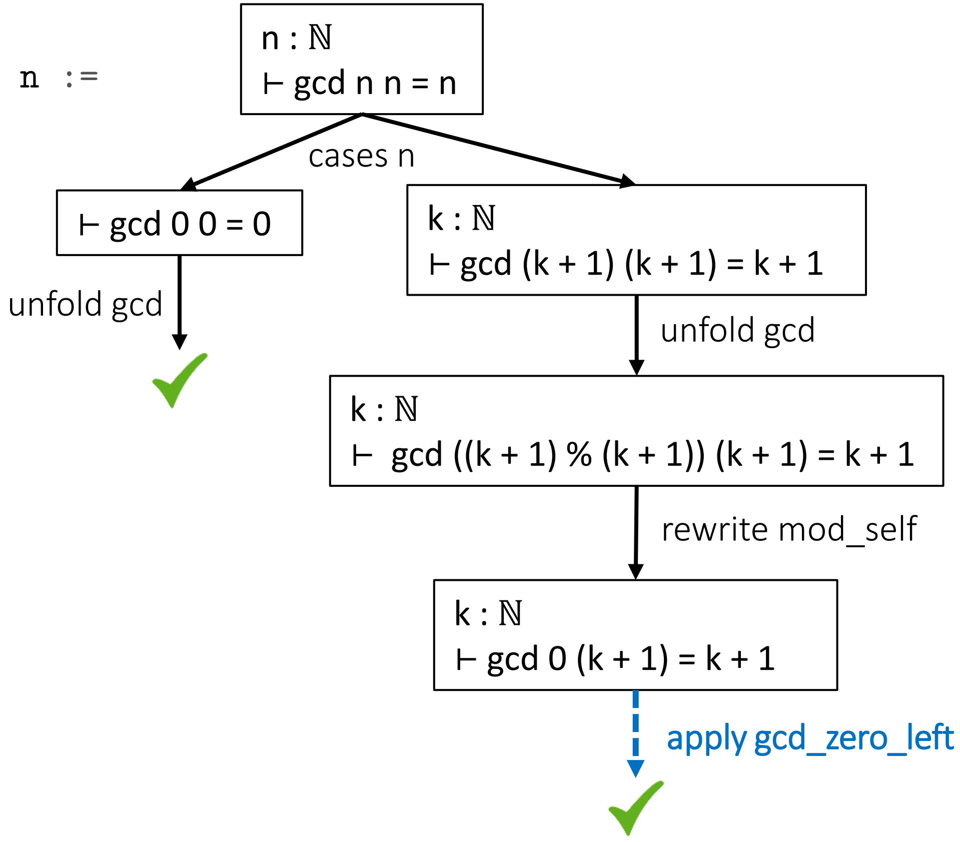


Proof assistant

Theorem Proving in Proof Assistants

Human

```
theorem gcd_self (n : nat) : gcd n n = n :=
begin
  cases n,
  { unfold gcd },
  unfold gcd,
  rewrite mod_self,
  apply gcd_zero_left
end
```

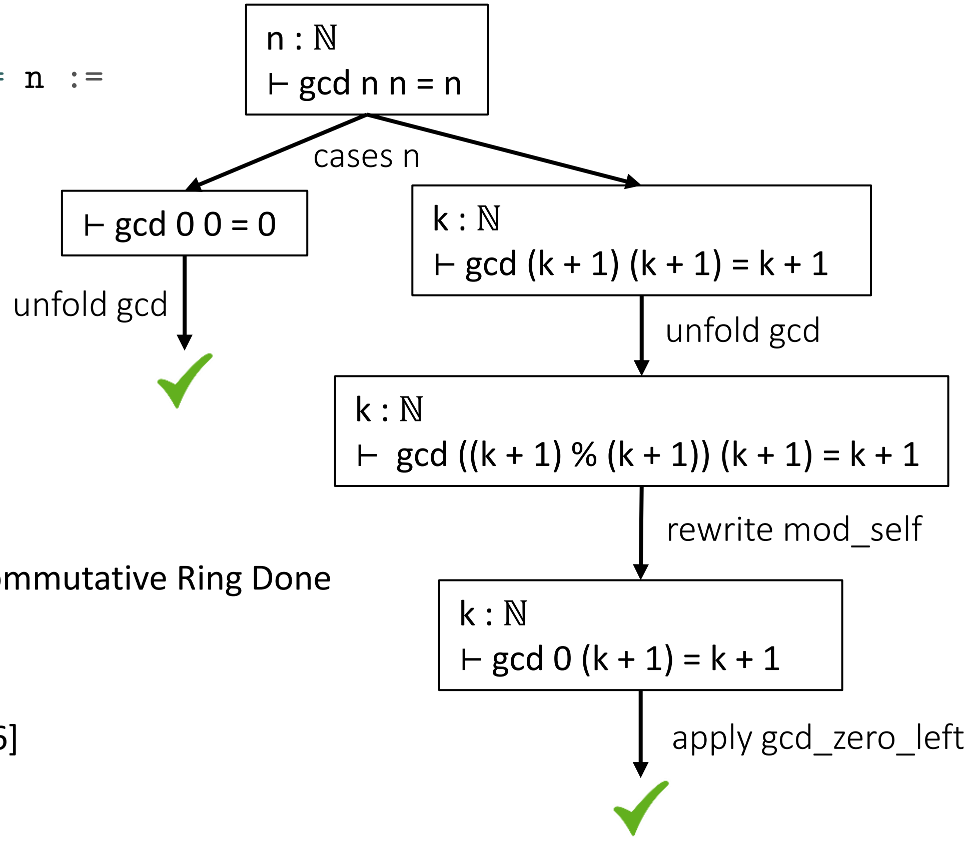


Proof assistant

Theorem Proving in Proof Assistants

Human

```
theorem gcd_self (n : nat) : gcd n n = n :=
begin
  cases n,
  { unfold gcd },
  unfold gcd,
  rewrite mod_self,
  apply gcd_zero_left
end
```

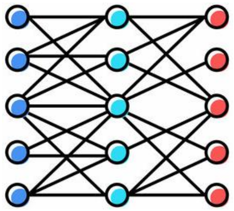


Proof assistant

- Efficient decision procedures
  - [Grégoire and Mahboubi, "Proving Equalities in a Commutative Ring Done Right in Coq", 2005]
- Hammers: outsource to external ATP systems
  - [Blanchette, et al., "Hammering towards QED", 2016]
- Proof search within the proof assistant
  - Coq's auto tactic
  - [Limperg and From, "Aesop: White-Box Best-First Proof Search for Lean", 2023]

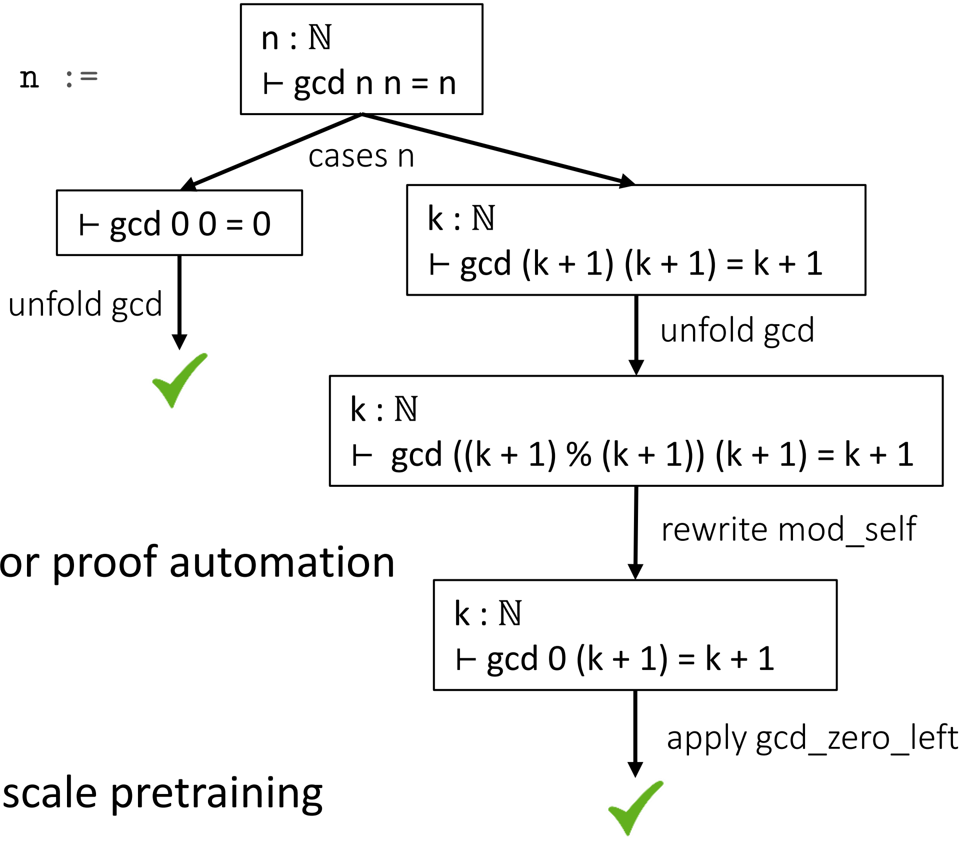
Theorem Proving in Proof Assistants

Human



Machine learning

```
theorem gcd_self (n : nat) : gcd n n = n :=
begin
  cases n,
  { unfold gcd },
  unfold gcd,
  rewrite mod_self,
  apply gcd_zero_left
end
```



Proof assistant

- Complementary to formal methods for proof automation
- Learning from human-written proofs
  - ~100K proofs in Lean
  - More in Coq and Isabelle
- Math knowledge learned from large-scale pretraining
  - Large language models (LLMs)

Why is Theorem Proving Important for AI?

The Era of Large Language Models (LLMs)

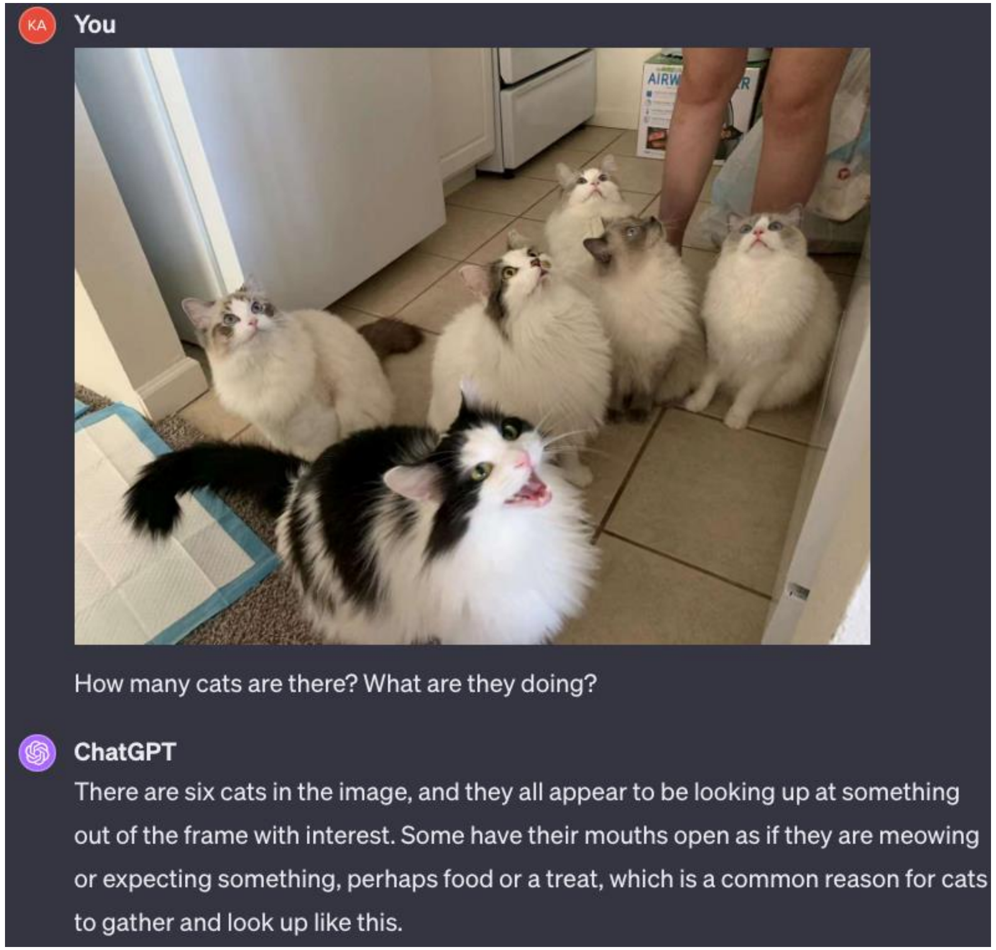
GPT-4

Claude

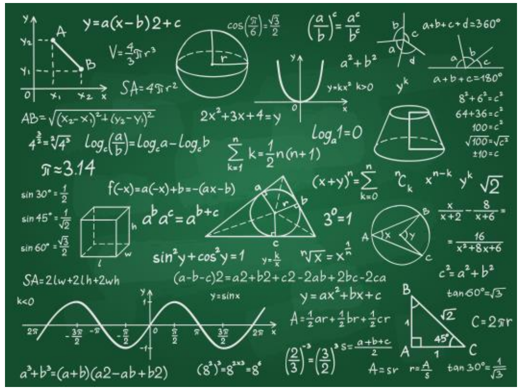
GitHub Copilot

[Ma et al., Eureka, 2023]

[Wang et al., Voyager, 2023]



Theorem Proving and LLMs

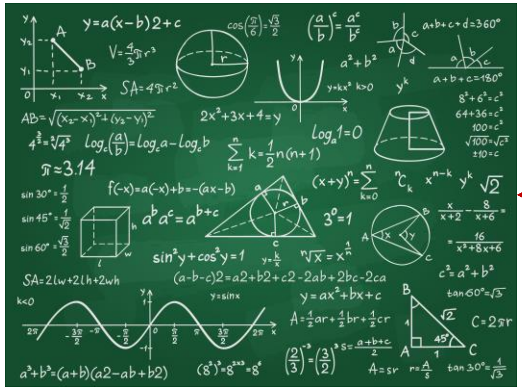


Mathematical reasoning
with LLMs

Theorem proving

Code generation
with LLMs

Theorem Proving and LLMs



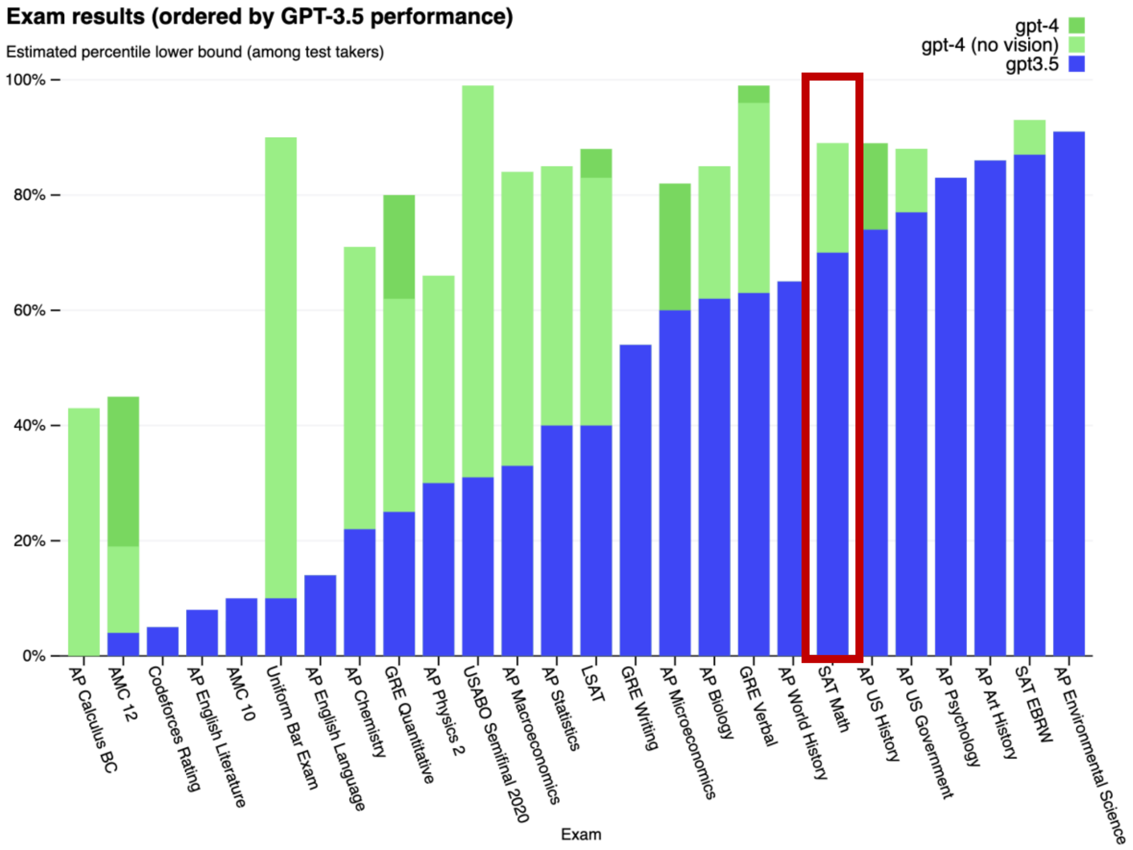
Mathematical reasoning with LLMs

Theorem proving

Code generation with LLMs

Mathematical Reasoning with LLMs

- GPT-4 scored 89th percentile on SAT Math



Mathematical Reasoning with LLMs

- GPT-4 scored 89th percentile on SAT Math
- Specialized math LLMs: Minerva, MetaMath, WizardMath, MAmmoTH, Llemma

Question: For every $a, b, b \neq a$ prove that

$$\frac{a^2+b^2}{2} > \left(\frac{a+b}{2}\right)^2.$$

Model output:

$$\begin{aligned}
\frac{a^2+b^2}{2} &> \left(\frac{a+b}{2}\right)^2 \\
\iff \frac{a^2+b^2}{2} &> \frac{a^2+b^2+2ab}{4} \\
\iff a^2+b^2 &> \frac{a^2+b^2+2ab}{2} \\
\iff 2a^2+2b^2 &> a^2+b^2+2ab \\
\iff a^2+b^2 &> 2ab \\
\iff a^2+b^2-2ab &> 0 \\
\iff (a-b)^2 &> 0
\end{aligned}$$

which is true, because the square of a real number is positive.

[Lewkowycz et al., **Minerva**, 2022]

Input:
Let $f(r) = \sum_{j=2}^{2008} \frac{1}{j^r} = \frac{1}{2^r} + \frac{1}{3^r} + \cdots + \frac{1}{2008^r}$. Find $\sum_{k=2}^{\infty} f(k)$.

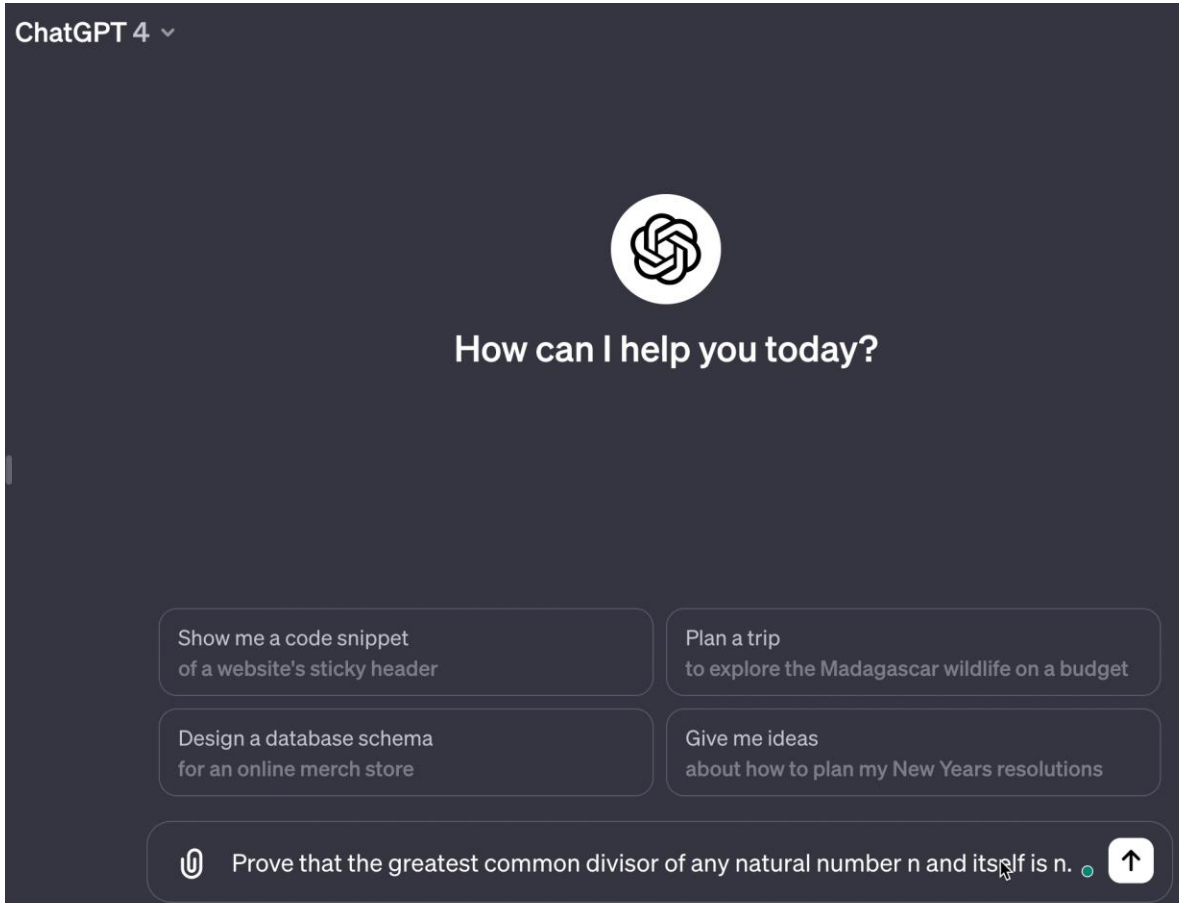
LLEMMA 34B solution:
We have

$$\begin{aligned}
\sum_{k=2}^{\infty} f(k) &= \sum_{k=2}^{\infty}\sum_{j=2}^{2008} \frac{1}{j^k} = \sum_{j=2}^{2008}\sum_{k=2}^{\infty} \frac{1}{j^k} = \sum_{j=2}^{2008} \frac{1}{j^2}\sum_{k=0}^{\infty} \frac{1}{j^k} = \sum_{j=2}^{2008} \frac{1}{j^2}\frac{1}{1-\frac{1}{j}} \\
&= \sum_{j=2}^{2008} \frac{1}{j(j-1)} = \sum_{j=2}^{2008} \left(\frac{1}{j-1} - \frac{1}{j}\right) \\
&= \left(\frac{1}{1} - \frac{1}{2}\right) + \left(\frac{1}{2} - \frac{1}{3}\right) + \cdots + \left(\frac{1}{2007} - \frac{1}{2008}\right) \\
&= 1 - \frac{1}{2008} \\
&= \boxed{\frac{2007}{2008}}.
\end{aligned}$$

Final Answer: The final answer is $\frac{2007}{2008}$.

[Azerbayev et al., **Llemma**, 2023]

Informal vs. Formal Mathematical Reasoning




```
theorem gcd_self (n : ℕ) : gcd n n = n
```

```
cases n
· unfold gcd
  rfl
· unfold gcd
  rw [mod_self]
  unfold gcd
  rfl
```

✓

```
cases n
· simp [gcd]
· rw [mod_self]
  unfold gcd
  rfl
```

✗

Important for LLMs to tackle advanced mathematics

- Grounded in environments that can provide feedback
- Simple and rigorous evaluation: formal proofs can be checked (no hallucination)

Informal ⟷ **Formal**

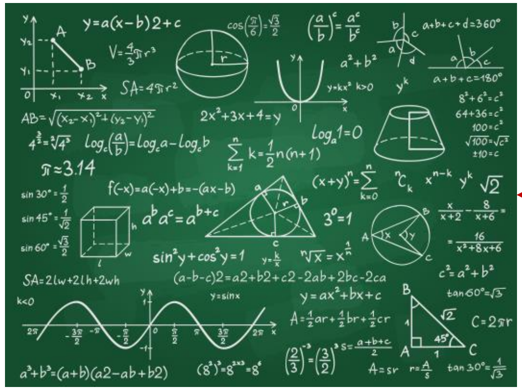
Checking Mathematical Proofs is Hard for Humans

Quantamagazine

Titans of Mathematics Clash Over Epic Proof of ABC Conjecture

Two mathematicians have found what they say is a hole at the heart of a proof that has convulsed the mathematics community for nearly six years.

Theorem Proving and LLMs



Mathematical reasoning
with LLMs

Theorem proving

Code generation
with LLMs

Theorem Proving and LLMs

Mathematical reasoning with LLMs

Theorem proving

Code generation with LLMs

Code Generation with LLMs

```python
def gcd (x : int, y : int) -> int:
    """Compute the greatest common divisor of ``x`` and ``y``.
    >>> gcd(10, 5)
    5
    >>> gcd(2, 3)
    1
    >>> gcd(8, 12)
    4
    """
    if x == 0:
        return y
    if y == 0:
        return x
    if x < y:
        return gcd(x, y % x)
    return gcd(x % y, y)
```

Passing a few testing examples ≠ correctness

Code Generation with LLMs

```
def gcd (x : int, y : int) -> int:
    """Compute the greatest common divisor of ``x`` and ``y``.
    >>> gcd(10, 5)
    5
    >>> gcd(2, 3)
    1
    >>> gcd(8, 12)
    4
    """
    if x == 0:
        return y
    if y == 0:
        return x
    if x < y:
        return gcd(x, y % x)
    return gcd(x % y, y)
```

GitHub Copilot

```
In [3]: gcd(-10, -5)
```

What if x and y are negative?

Passing a few testing examples ≠ correctness

Code Generation with LLMs

```python
def gcd (x : int, y : int) -> int:
    """Compute the greatest common divisor of ``x`` and ``y``.
    >>> gcd(10, 5)
    5
    >>> gcd(2, 3)
    1
    >>> gcd(8, 12)
    4
    """
    if x == 0:
        return y
    if y == 0:
        return x
    if x < y:
        return gcd(x, y % x)
    return gcd(x % y, y)
```

GitHub Copilot

```
In [3]: gcd(-10, -5)
---------------------------------------------------------------------------
RecursionError                            Traceback (most recent call last)
Cell In[3], line 1
----> 1 gcd(-10, -5)

File ~/LeanDojo/tmp.py:16, in gcd(x, y)
     14         return x
     15 if x < y:
---> 16         return gcd(x, y % x)
     17 return gcd(x % y, y)

File ~/LeanDojo/tmp.py:16, in gcd(x, y)
     14         return x
     15 if x < y:
---> 16         return gcd(x, y % x)
     17 return gcd(x % y, y)

    [... skipping similar frames: gcd at line 16 (2981 times)]

File ~/LeanDojo/tmp.py:16, in gcd(x, y)
     14         return x
     15 if x < y:
---> 16         return gcd(x, y % x)
     17 return gcd(x % y, y)

File ~/LeanDojo/tmp.py:11, in gcd(x, y)
      2 def gcd (x : int, y : int) -> int:
      3     """Compute the greatest common divisor of ``x`` and ``y``.
      4     >>> gcd(10, 5)
      5     5
   (...)
      9     4
     10     """
---> 11     if x == 0:
     12         return y
     13     if y == 0:

RecursionError: maximum recursion depth exceeded in comparison
```

Passing a few testing examples ≠ correctness

How Can We Trust AI-Generated Code?

Freethink

GitHub CEO says Copilot will write 80% of code "sooner than later"

Theorem Proving for Verified Code Generation

- Generate code + formal specification (theorem) + formal proof

[Sun and Sheng et al., "Clover: Closed-Loop Verifiable Code Generation", 2023]

Code

```
def abs (x : ℤ) : ℤ :=
  if x < 0 then -x else x
```

Code generation

Theorem Proving for Verified Code Generation

- Generate code + formal specification (theorem) + formal proof

[Sun and Sheng et al., "Clover: Closed-Loop Verifiable Code Generation", 2023]

Code

```
def abs (x : ℤ) : ℤ :=
  if x < 0 then -x else x
```

Code generation

Specification (Theorem)

```
theorem abs_max (x : ℤ) : abs x = max x (-x)
```

Verified Code generation

Theorem Proving for Verified Code Generation

- Generate code + formal specification (theorem) + formal proof

[Sun and Sheng et al., "Clover: Closed-Loop Verifiable Code Generation", 2023]

Code

```
def abs (x : ℤ) : ℤ :=
  if x < 0 then -x else x
```

Code generation

Specification (Theorem)

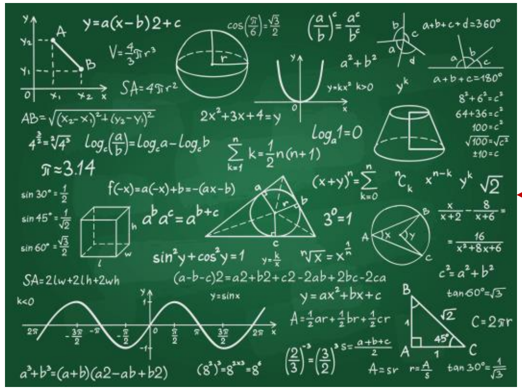
```
theorem abs_max (x : ℤ) : abs x = max x (-x)
```

Proof

```
  by_cases h : x < 0 <;> simp [abs, h]
  · have : x < -x := by simp_all
    rw [max_eq_right_of_lt this]
  · have : -x ≤ x := by simp_all
    rw [max_eq_left this]
```

Verified Code generation

Theorem Proving and LLMs: Takeaways



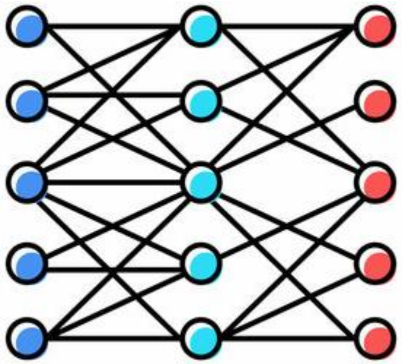
Mathematical reasoning
with LLMs

Theorem proving

Code generation
with LLMs

- Elementary math -> advanced math
- Verified code generation
- Feedback & evaluation at scale: AI mathematicians/programmers

Theorem Proving and LLMs: Takeaways



Machine learning Proof assistants

How to Prove Theorems (with Machine Learning)?

Proof Assistants (Interactive Theorem Provers)

Humans  + Proof assistants

IDEs for writing formal proofs

Theorem

```
theorem set_inter_comm (s t : Set α) : s ∩ t = t ∩ s
```

Proof

```
ext x
simp [Set.mem_inter_iff]
constructor
· rintro ⟨xs, xt⟩
  exact ⟨xt, xs⟩
. rintro ⟨xt, xs⟩
  exact ⟨xs, xt⟩
```

Examples of Proof Assistants

Isabelle
[Nipkow et al., 2002]

- Large formal libraries: ~250K proofs

Coq
[Barras et al., 1997]

- >100K proofs in different repos
- Popular for software verification, e.g., CompCert [Leroy et al., 2016]

Lean
[de Moura et al., 2015]

- ~100K proofs in Mathlib
- Liquid tensor experiment [Commelin, 2022]
- Polynomial Freiman-Ruzsa conjecture (led by Terence Tao)

Examples of Proof Assistants

Isabelle
[Nipkow et al., 2002]

- Large formal libraries: ~250K proofs

[First et al., Baldur, 2023]
[Jiang et al., Thor, 2022]
[Mikuła et al., Magnushammer, 2023]
[Jiang et al., DSP, 2023]
[Wu et al., Autoformalization, 2022]
[Li et al., IsarStep, 2021]
[Wang and Xin et al., LEGO-Prover, 2023]

Coq
[Barras et al., 1997]

- >100K proofs in different repos
- Popular for software verification, e.g., CompCert [Leroy et al., 2016]

[Huang et al., GamePad, 2018]
[Yang and Deng, CoqGym, 2019]
[Sivaraman, et al., Lemma Synthesis, 2022]
[Sanchez-Stern et al., Proverbot9001, 2020]
[Ringer et al., REPLica, 2020]
[Sanchez-Stern and First et al., Passport, 2023]

Lean
[de Moura et al., 2015]

- ~100K proofs in Mathlib
- Liquid tensor experiment [Commelin, 2022]
- Polynomial Freiman-Ruzsa conjecture (led by Terence Tao)

[Han et al., PACT, 2022]
[Polu et al., 2023]
[Lample et al., HTPS 2022]
[Want et al., DT-Solver, 2023]
[Yang et al., LeanDojo, 2023]
[Thakur et al., COPRA, 2023]

Examples of Proof Assistants

Isabelle
[Nipkow et al., 2002]

- Large formal libraries: ~250K proofs

[First et al., Baldur, 2023]
[Jiang et al., Thor, 2022]
[Mikuła et al., Magnushammer, 2023]
[Jiang et al., DSP, 2023]
[Wu et al., Autoformalization, 2022]
[Li et al., IsarStep, 2021]
[Wang and Xin et al., LEGO-Prover, 2023]

Coq
[Barras et al., 1997]

- >100K proofs in different repos
- Popular for software verification, e.g., CompCert [Leroy et al., 2016]

[Huang et al., GamePad, 2018]
[Yang and Deng, CoqGym, 2019]
[Sivaraman, et al., Lemma Synthesis, 2022]
[Sanchez-Stern et al., Proverbot9001, 2020]
[Ringer et al., REPLica, 2020]
[Sanchez-Stern and First et al., Passport, 2023]

Lean
[de Moura et al., 2015]

- ~100K proofs in Mathlib
- Liquid tensor experiment [Commelin, 2022]
- Polynomial Freiman-Ruzsa conjecture (led by Terence Tao)

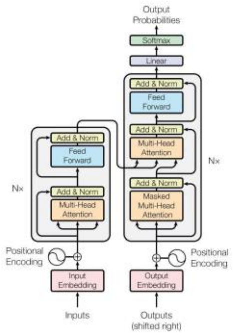
[Han et al., PACT, 2022]
[Polu et al., 2023]
[Lample et al., HTPS 2022]
[Want et al., DT-Solver, 2023]
[Yang et al., LeanDojo, 2023]
[Thakur et al., COPRA, 2023]

Proving Theorems Using Language Models

Input: Theorem

```
theorem add_abc : ∀ a b c : ℕ, a + b + c = a + c + b
```

```
intro a b c
rw [Nat.add_right_comm
```



[Vaswani et al., Transformer, 2017]

Output: Proof

Generating Proof Steps (Tactics)

```
theorem add_abc : ∀ a b c : ℕ, a + b + c = a + c + b := by
  intro a b c
  rw [Nat.add_right_comm]
```

Generating Proof Steps (Tactics)

```
theorem add_abc : ∀ a b c : ℕ, a + b + c = a + c + b := by
  intro a b c
  rw [Nat.add_right_comm]
```

Proof state

$\vdash \forall\ (a\ b\ c : \mathbb{N}),\ a + b + c = a + c + b$

Generating Proof Steps (Tactics)

```
theorem add_abc : ∀ a b c : ℕ, a + b + c = a + c + b := by
  intro a b c
  rw [Nat.add_right_comm]
```

Proof state **Tactic**

$\vdash \forall (a\ b\ c : \mathbb{N}), a + b + c = a + c + b \xrightarrow{\text{intro a b c}}$ a b c: $\mathbb{N}$ $\vdash a + b + c = a + c + b$

Generating Proof Steps (Tactics)

```
theorem add_abc : ∀ a b c : ℕ, a + b + c = a + c + b := by
  intro a b c
  rw [Nat.add_right_comm]
```

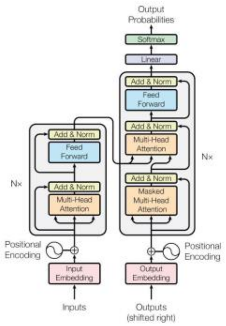
Proof state | **Tactic**

$\vdash \forall (a\ b\ c : \mathbb{N}),\ a + b + c = a + c + b$ $\xrightarrow{\text{intro a b c}}$ $a\ b\ c : \mathbb{N}$, $\vdash a + b + c = a + c + b$ $\xrightarrow{\text{rw [Nat.add\_right\_comm]}}$ ✓

Generating Proof Steps (Tactics)

```
theorem add_abc : ∀ a b c : ℕ, a + b + c = a + c + b := by
  intro a b c
  rw [Nat.add_right_comm]
```

Tactic generator

Input: Proof state 

Output: Tactic

⊢ ∀ (a b c : ℕ), a + b + c = a + c + b

intro a b c

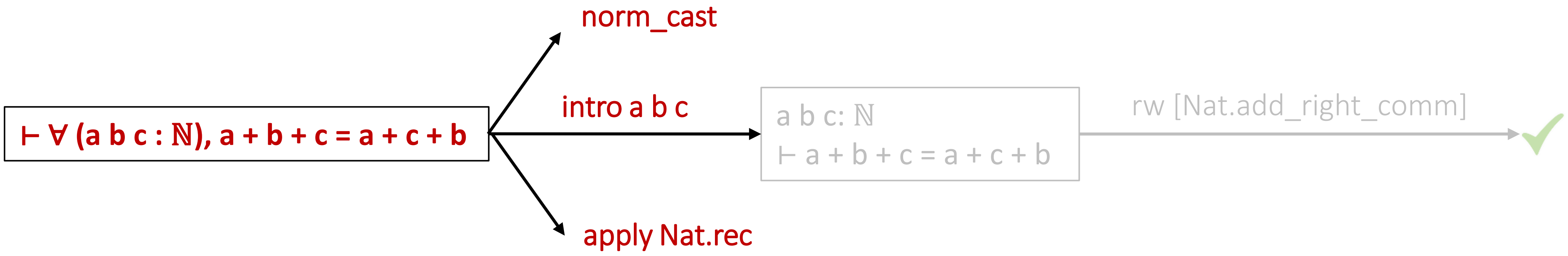
a b c: ℕ
⊢ a + b + c = a + c + b

rw [Nat.add_right_comm]

✓

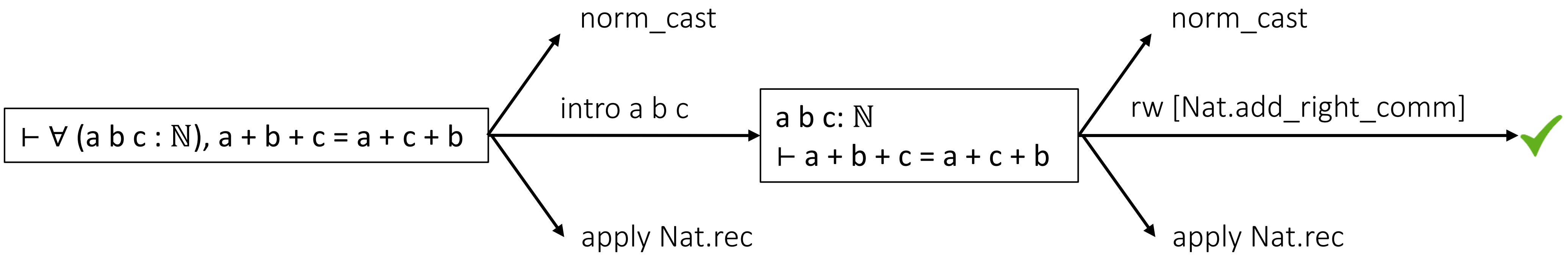
Generating Proof Steps (Tactics)

```
theorem add_abc : ∀ a b c : ℕ, a + b + c = a + c + b := by
  intro a b c
  rw [Nat.add_right_comm]
```



Searching for Proofs

```
theorem add_abc : ∀ a b c : ℕ, a + b + c = a + c + b := by
  intro a b c
  rw [Nat.add_right_comm]
```

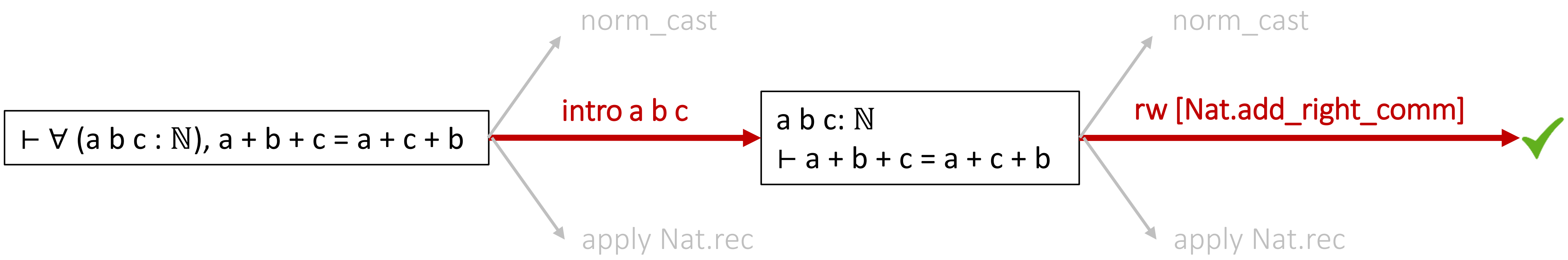


Searching for Proofs

```
theorem add_abc : ∀ a b c : ℕ, a + b + c = a + c + b := by
  intro a b c
  rw [Nat.add_right_comm]
```

Classical proof search algorithms

- Depth first search (DFS)
- Breadth first search (BFS)
- …



We've successfully built a simple prover!
… now what?

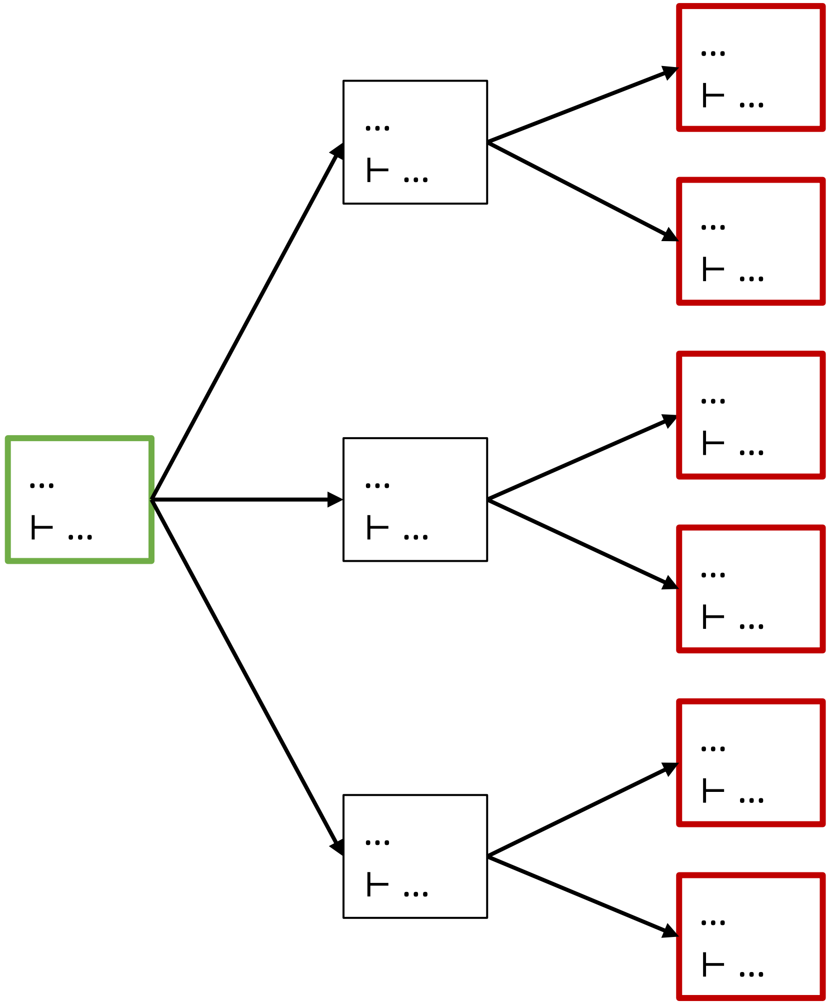
Proof search

Premise selection

Reinforcement Learning

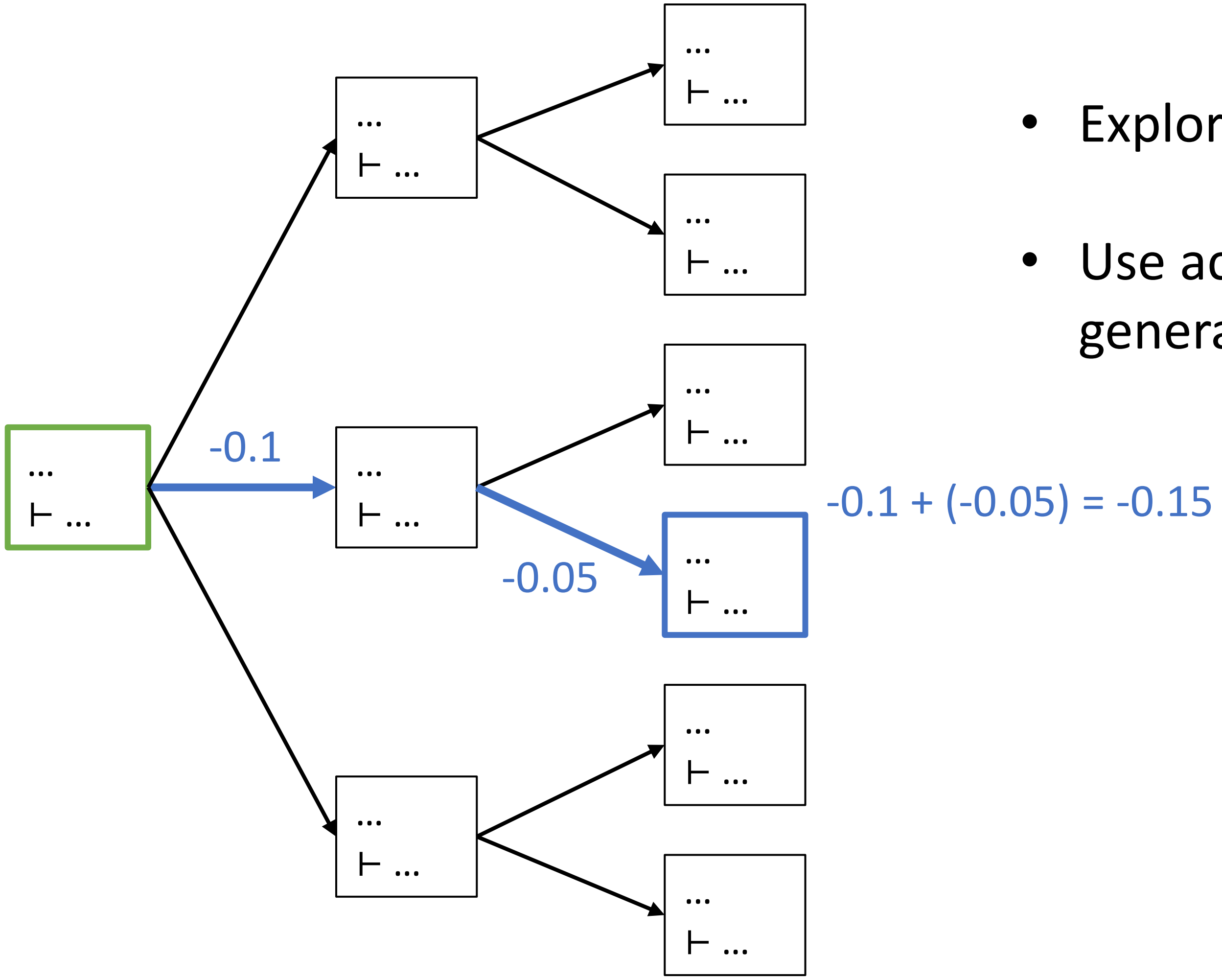
Synthetic data generation

Best First Search



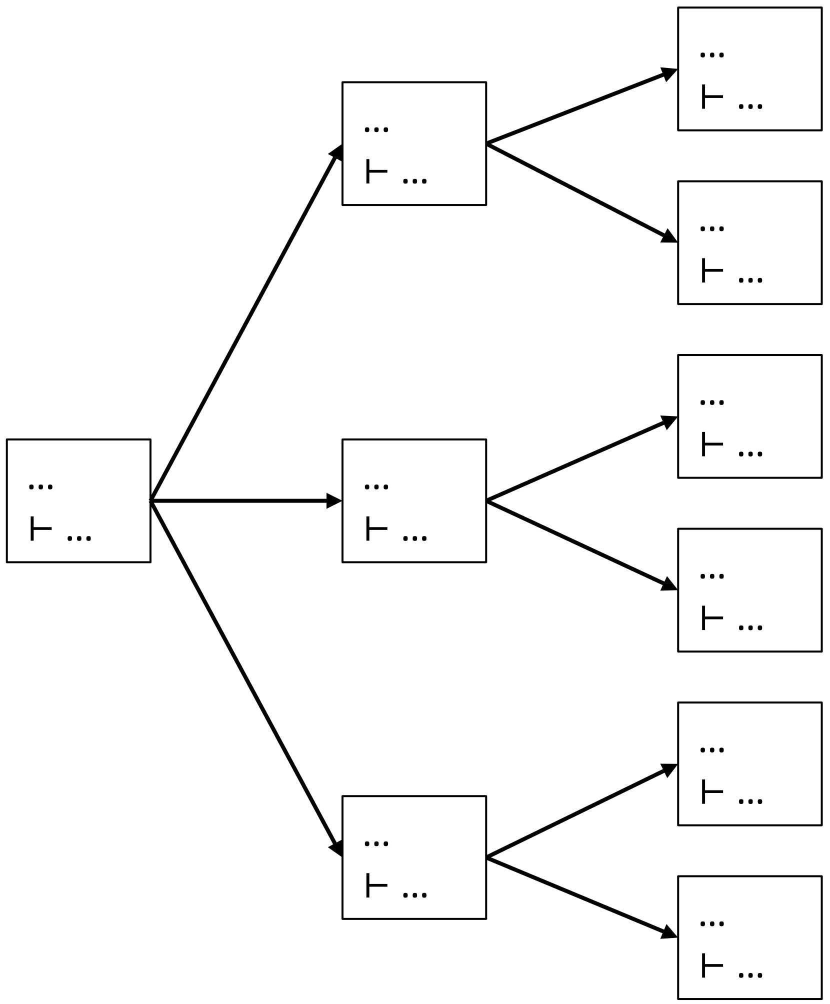
- Explore the most promising node
- Use accumulated scores from the tactic generator to rank the nodes

Best First Search



- Explore the most promising node
- Use accumulated scores from the tactic generator to rank the nodes

Best First Search



- Explore the most promising node
- Use accumulated scores from the tactic generator to rank the nodes
- **Simple and widely used**

[Han et al., PACT, ICLR 2022]
[Polu et al., ICLR 2023]
[Jiang et al., Thor, NeurIPS 2022]
[Yang et al., LeanDojo, NeurIPS 2023]

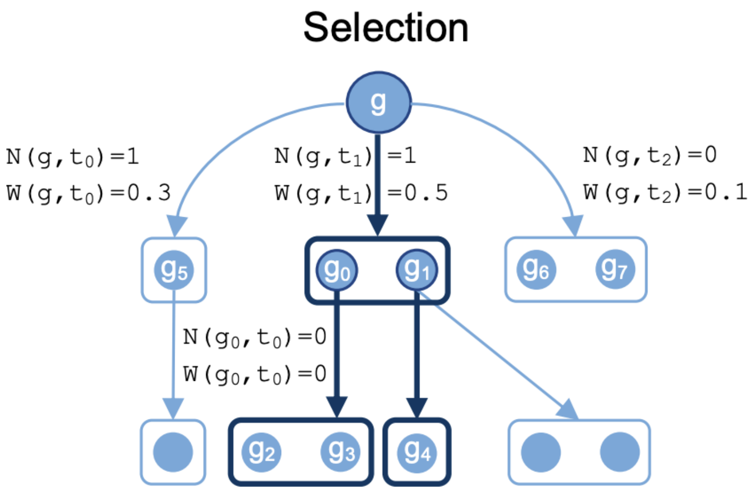
[Lampe et al., "HyperTree Proof Search for Neural Theorem Proving", NeurIPS 2022]

Hyper Tree Proof Search

- Inspired by Monte Carlo Tree Search (MCTS)
- Update visit counts and estimated values for each node

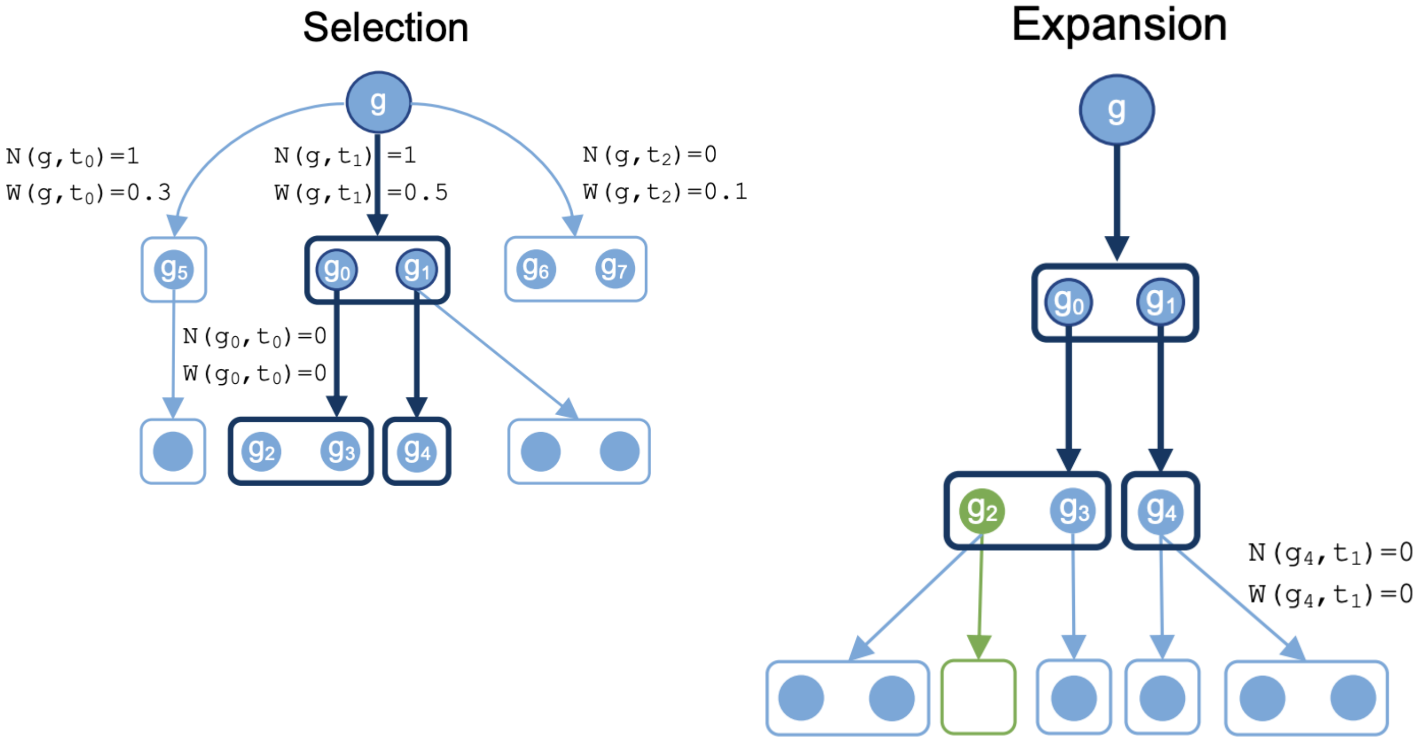
Hyper Tree Proof Search

- Inspired by Monte Carlo Tree Search (MCTS)
- Update visit counts and estimated values for each node



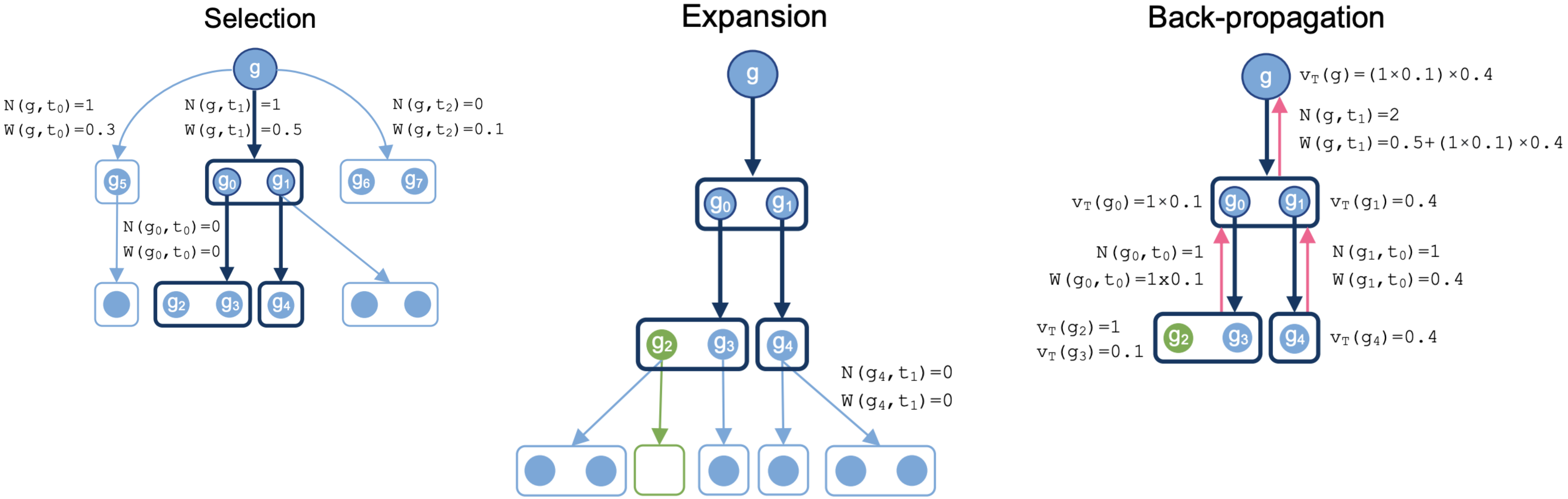
Hyper Tree Proof Search

- Inspired by Monte Carlo Tree Search (MCTS)
- Update visit counts and estimated values for each node



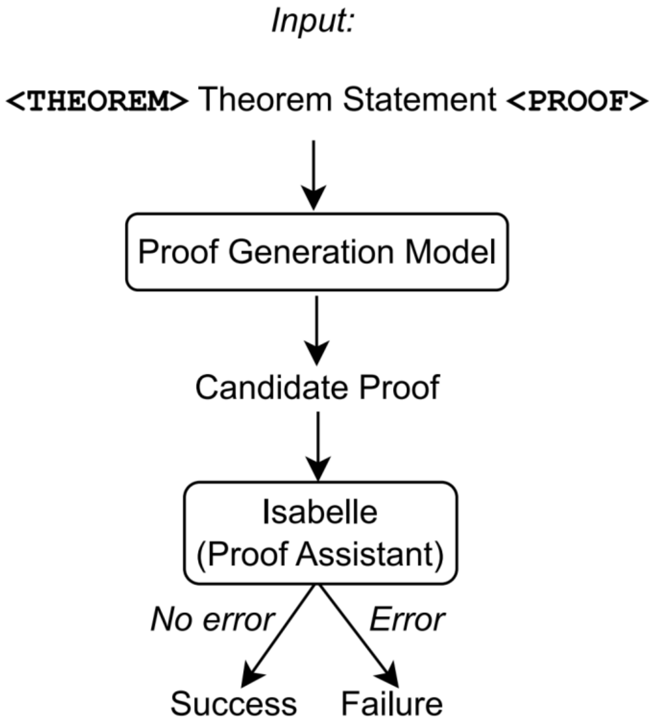
Hyper Tree Proof Search

- Inspired by Monte Carlo Tree Search (MCTS)
- Update visit counts and estimated values for each node



Is Proof Search Really Necessary?

- Baldur: It's possible to build state-of-the-art provers without search
- 6B and 62B models finetuned from Minerva on Isabelle proofs



Premise Selection

```
theorem add_abc : ∀ a b c : ℕ, a + b + c = a + c + b := by
  intro a b c
  rw  Nat.add_right_comm
```

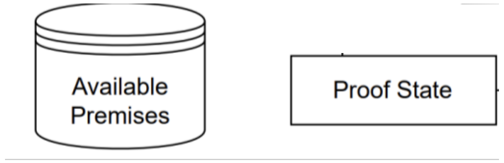
- Premise selection: A key challenge in theorem proving
- Studied as a separate task w/o theorem proving

[Irving et al., DeepMath, NeurIPS 2016]
[Wang et al., "Premise Selection for Theorem Proving by Deep Graph Embedding", NeurIPS 2017]

[Mikuła et al., "Magnushammer: A Transformer-based Approach to Premise Selection", 2023]

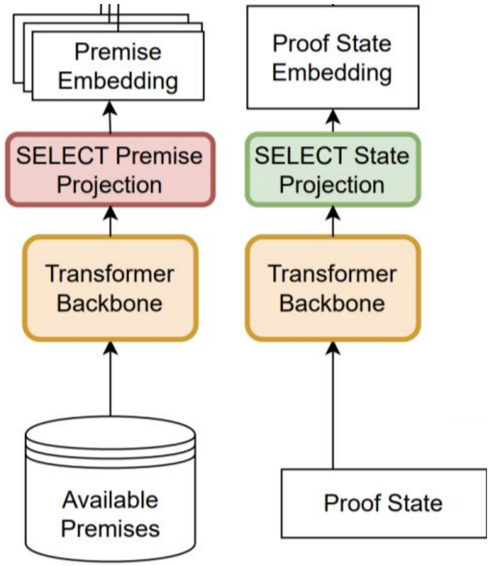
Magnushammer

- Premises selected by Transformer + a simple symbolic prover



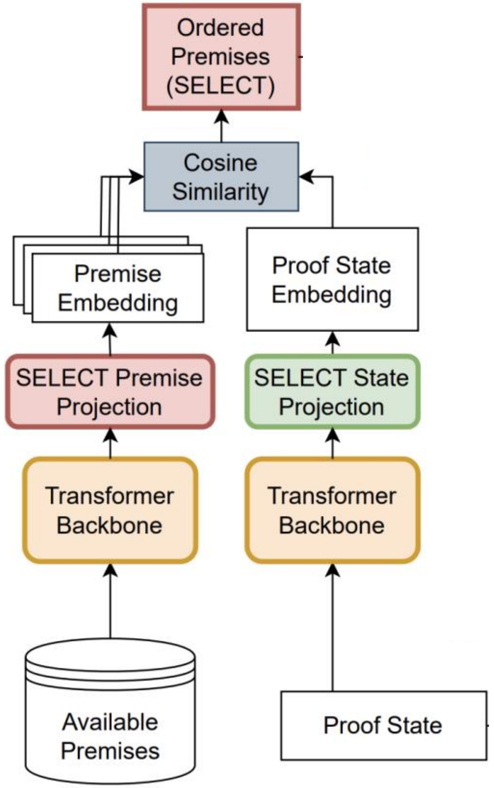
Magnushammer

- Premises selected by Transformer + a simple neuro-symbolic prover



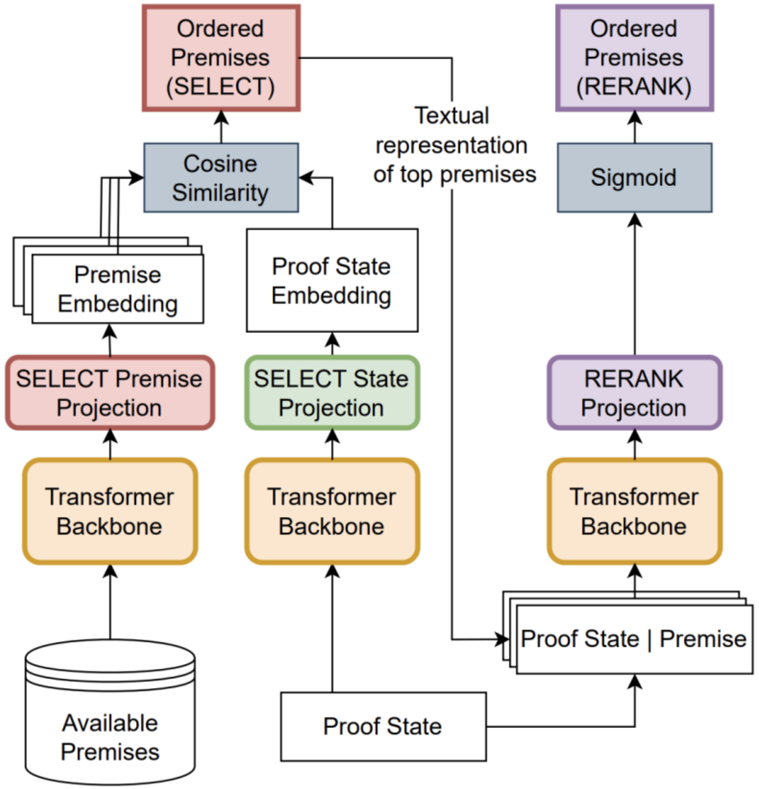
Magnushammer

- Premises selected by Transformer + a simple neuro-symbolic prover



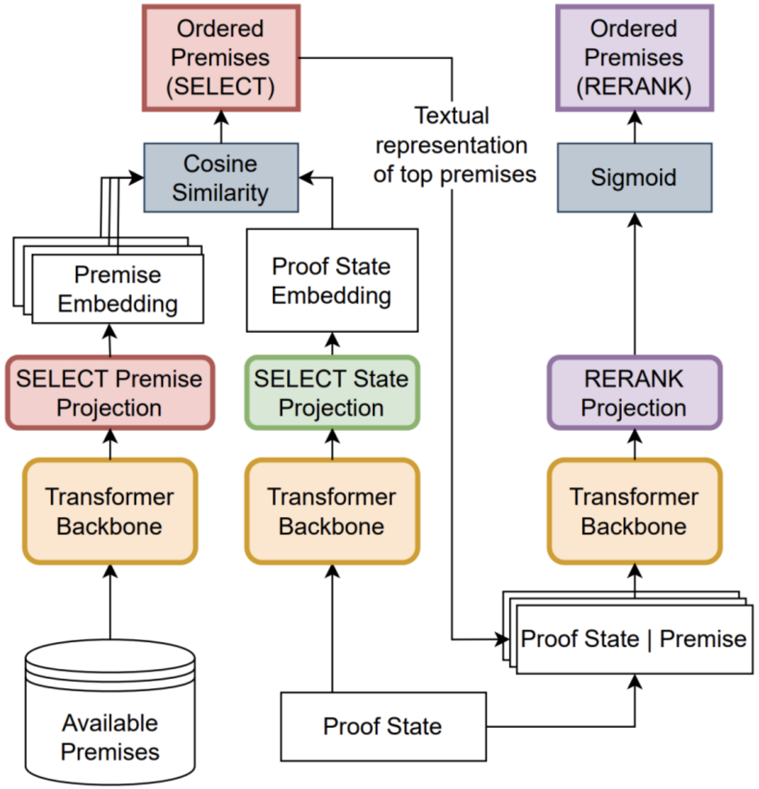
Magnushammer

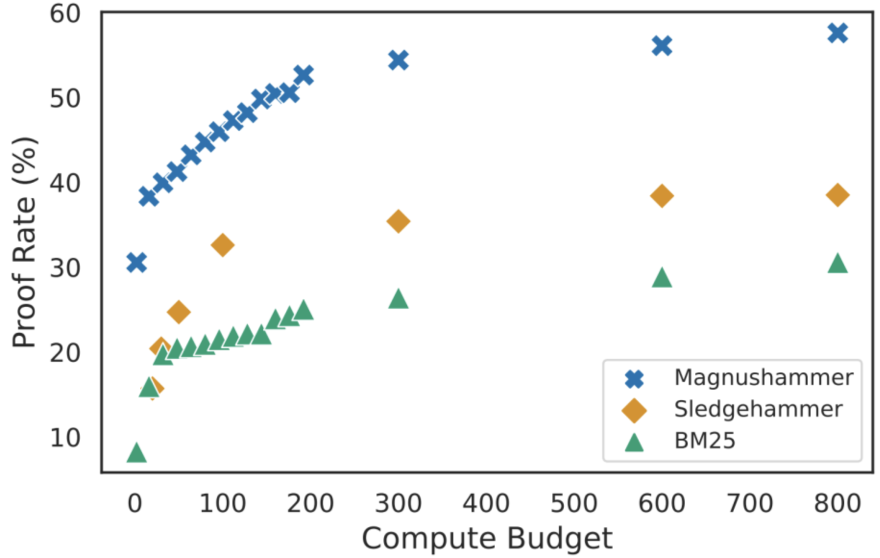
- Premises selected by Transformer + a simple neuro-symbolic prover



Magnushammer

- Premises selected by Transformer + a simple neuro-symbolic prover





[Yang et al., "LeanDojo: Theorem Proving with Retrieval-Augmented Language Models", NeurIPS 2023]

ReProver: Retrieval-Augmented Prover

- Given a state, we retrieve premises from accessible premises

State

$k : \mathbb{N}$
$\vdash \ \mathrm{gcd}\ ((k + 1)\ \%\ (k + 1))\ (k + 1) = k + 1$

All ***accessible premises***
in the math library

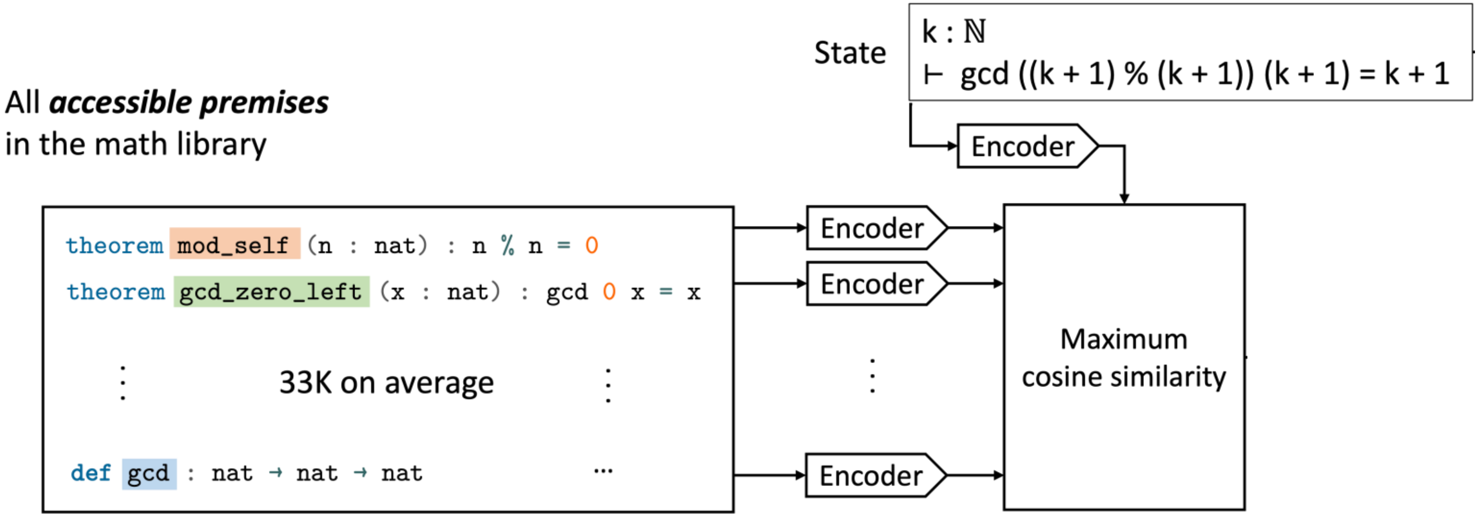
```
theorem mod_self (n : nat) : n % n = 0
theorem gcd_zero_left (x : nat) : gcd 0 x = x

  ⋮            33K on average           ⋮

def gcd : nat → nat → nat               ...
```

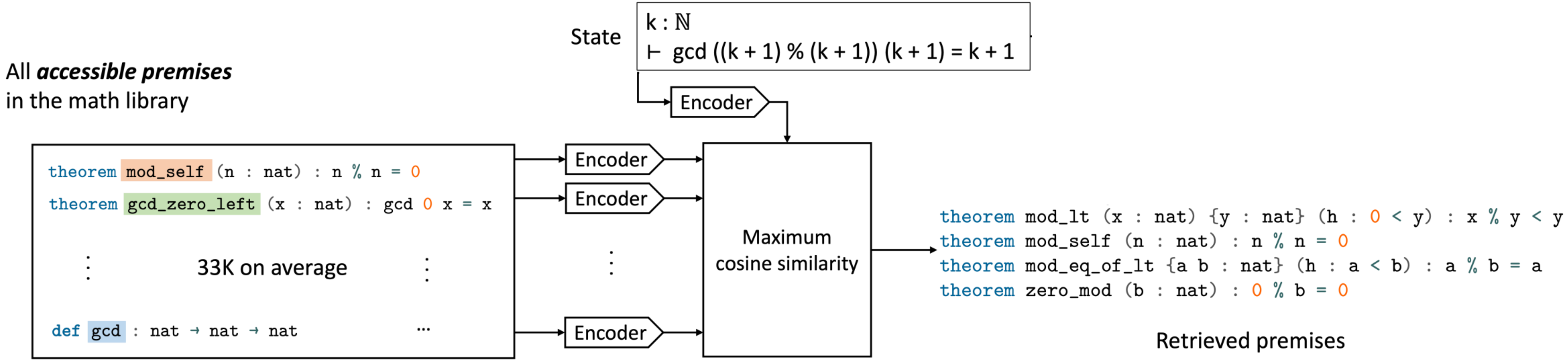
ReProver: Retrieval-Augmented Prover

- Given a state, we retrieve premises from accessible premises



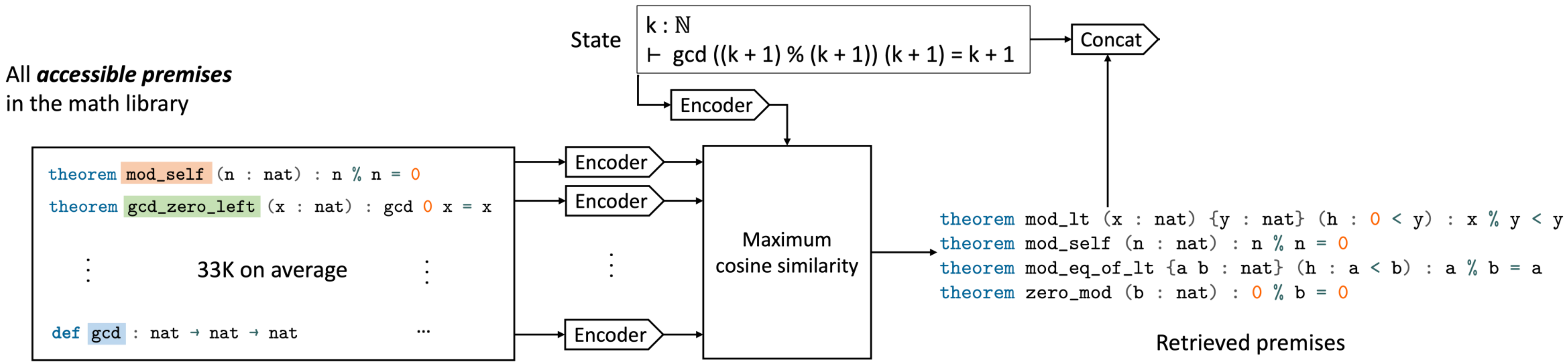
ReProver: Retrieval-Augmented Prover

- Given a state, we retrieve premises from accessible premises



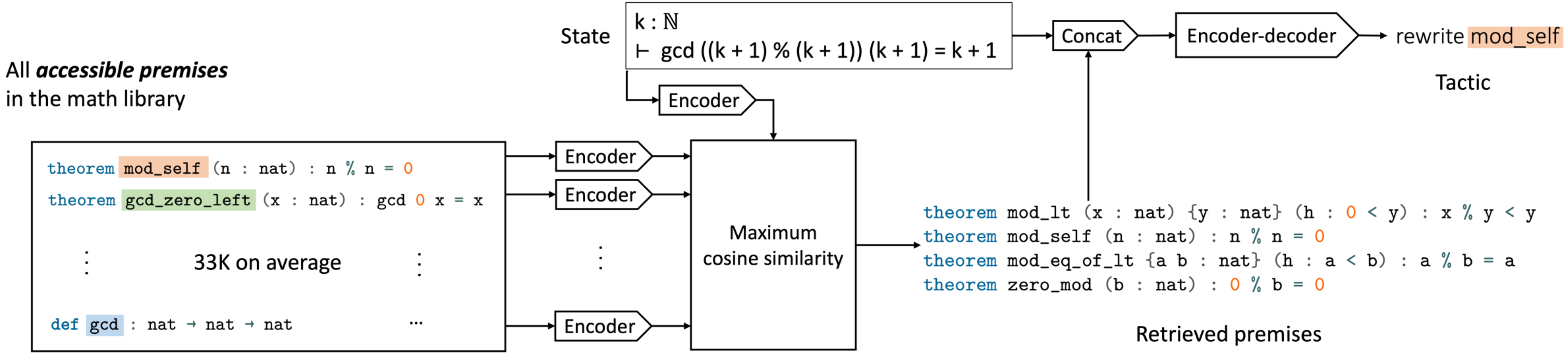
ReProver: Retrieval-Augmented Prover

- Given a state, we retrieve premises from accessible premises
- Retrieved premises are concatenated with the state and used for **tactic generation**

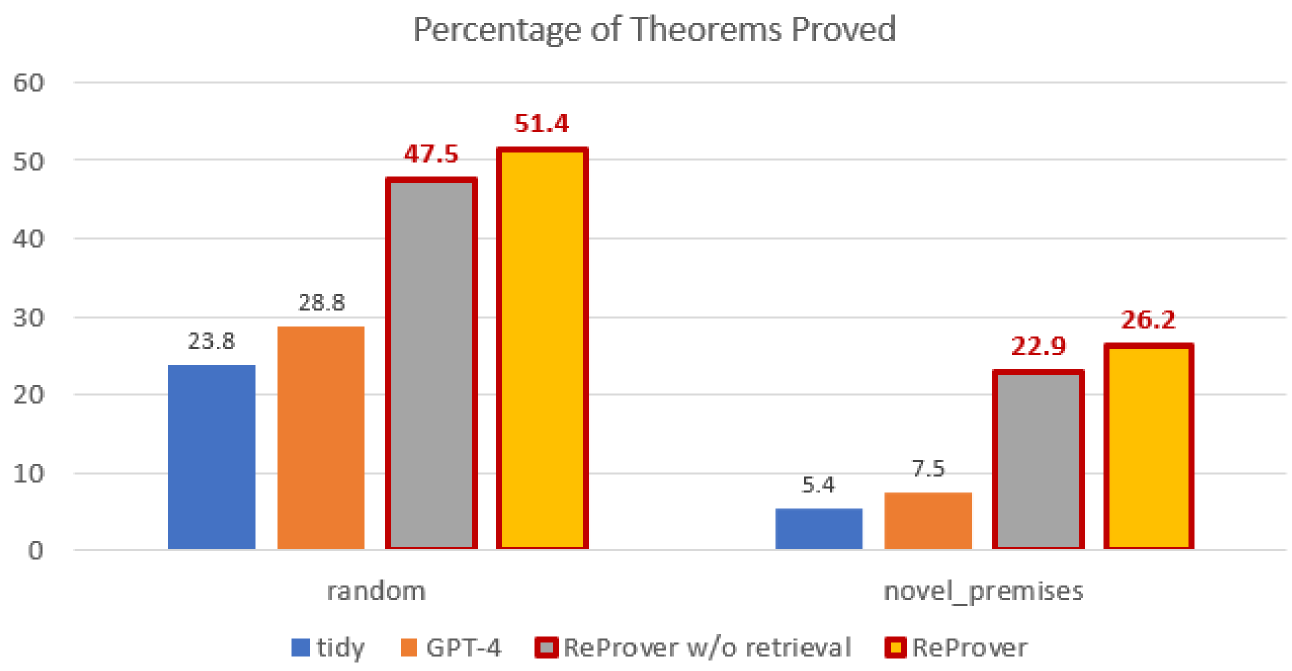


ReProver: Retrieval-Augmented Prover

- Given a state, we retrieve premises from accessible premises
- Retrieved premises are concatenated with the state and used for **tactic generation**



ReProver: Retrieval-Augmented Prover



Method	random	novel_premises
tidy	23.8	5.3
GPT-4	29.0	7.4
ReProver (ours)	**51.2**	**26.3**
w/o retrieval	47.6	23.2

Reinforcement Learning

- Specialized domains without sufficient existing proofs for training, e.g., MiniF2F
- LLMs perform badly on out-of-domain data

[Bansal et al., "Learning to Reason in Large Theories without Imitation“, arXiv 2020]
[Wu et al., "TacticZero: Learning to Prove Theorems from Scratch with Deep Reinforcement Learning“, NeurIPS 2021]
[Lample et al., "HyperTree Proof Search for Neural Theorem Proving", NeurIPS 2022]
[Polu et al., "Formal Mathematics Statement Curriculum Learning", 2023]

Expert Iteration

- Specialized domains without sufficient existing proofs for training, e.g., MiniF2F
- LLMs perform badly on out-of-domain data

[Bansal et al., "Learning to Reason in Large Theories without Imitation“, arXiv 2020]
[Wu et al., "TacticZero: Learning to Prove Theorems from Scratch with Deep Reinforcement Learning“, NeurIPS 2021]
[Lample et al., "HyperTree Proof Search for Neural Theorem Proving", NeurIPS 2022]
[Polu et al., "Formal Mathematics Statement Curriculum Learning", 2023]

[Polu et al., "Formal Mathematics Statement Curriculum Learning", 2023]

OpenAI

Expert Iteration

Solving (some) formal math olympiad problems

- Specialized domains without sufficient existing proofs for training, e.g., MiniF2F
- LLMs perform badly on out-of-domain data
- **Solution: Iteratively improve the prover on the new domain**
  1. Train the prover
  2. Use the prover to find new proofs
  3. Add new proofs to the training data and go back to step 1

Model	*d*	*e*	*pass@1*	*pass@8*
mathlib-valid				
PACT	512	16	48.4%	
θ_0^*	512	16	48.5%	57.6%
θ_0	512	8	46.7%	57.5%
θ_1	512	8	**56.3%**	**66.3%**

Model	*d*	*e*	*pass@1*	*pass@8*
miniF2F-valid				
miniF2F	128	16	23.9%	29.3%
θ_0^*	128	16	27.6%	31.8%
θ_0	512	8	28.4%	33.6%
θ_1	512	8	**28.5%**	**35.5%**

Wait! There’s something left out…

MiniF2F Benchmark

- Math olympiads problems from AMC, AIME, IMO, etc.
- 488 theorems (many w/o proof) for evaluation; no training

MiniF2F Benchmark

- Math olympiads problems from AMC, AIME, IMO, etc.
- 488 theorems (many w/o proof) for evaluation; no training
- Open problems:
  - How to formalize problems asking for numerical answers?
  - How to deal with geometry?

Informal

Solve for a: $\sqrt{4+\sqrt{16+16a}}+\sqrt{1+\sqrt{1+a}}=6$. Show that it is 8.

Lean

```
theorem mathd_algebra_17
  (a : ℝ)
  (h₀ : real.sqrt (4 + real.sqrt (16 + 16 * a)) + real.sqrt (1 + real.sqrt (1 + a))
  a = 8 :=
begin
  sorry
end
```

Datasets & Benchmarks

High quality datasets are available for Lean & Coq

LeanDojo

- 98,641 theorems and proofs
- 217,639 tactics
- 129,162 premises

[Yang et al., "LeanDojo: Theorem Proving with Retrieval-Augmented Language Models", 2023]

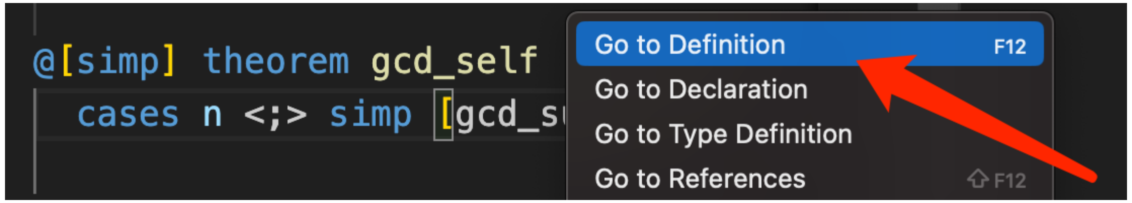
CoqGym

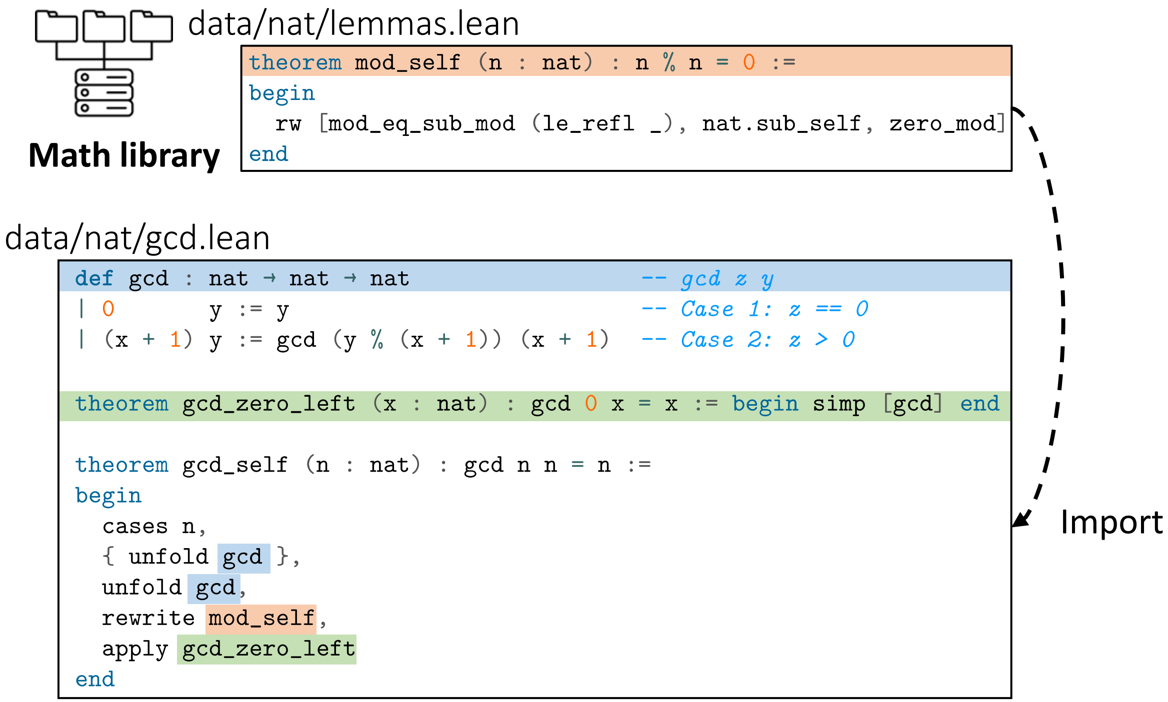
- 71K human-written proofs
- Ranging among 123 projects

[Yang et al., "Learning to Prove Theorems via Interacting with Proof Assistants", 2019]

Data Extraction in LeanDojo

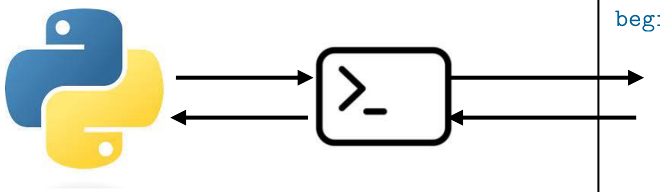
- ASTs, tactics
  - From Lean's parser
- Proof goals
  - From Lean's InfoTree
- Premises
  - Definitions, lemmas, etc.
  - Where they are used/defined
  - Also in the InfoTree





Programmatical Interaction in LeanDojo

- Replace the human-written proof with a single repl tactic
- repl performs IO to provide a command line interface for interacting with Lean
- Wrap the interface in any language, e.g., Python



data/nat/gcd.lean

```
def gcd : nat → nat → nat                       -- gcd z y
| 0       y := y                                 -- Case 1: z == 0
| (x + 1) y := gcd (y % (x + 1)) (x + 1)        -- Case 2: z > 0

theorem gcd_zero_left (x : nat) : gcd 0 x = x := begin simp [gcd] end

theorem gcd_self (n : nat) : gcd n n = n :=
begin

  repl

end
```

[Han et al., "Proof Artifact Co-training for Theorem Proving with Language Models", 2022]

Proof Artifact Co-training

- LLMs are data-hungry, but human-written proofs are limited (~100K proofs in mathlib)
- 9 auxiliary tasks
    - **Next lemma prediction**: Proof goal -> the next lemma to be applied
    - **Type prediction**: Partial proof term -> its type
    - **Theorem naming**: theorem statement -> its name
    - …

Proof Artifact Co-training

- LLMs are data-hungry, but human-written proofs are limited (~100K proofs in mathlib)
- 9 auxiliary tasks
    - **Next lemma prediction**: Proof goal -> the next lemma to be applied
    - **Type prediction**: Partial proof term -> its type
    - **Theorem naming**: theorem statement -> its name
    - …

Model	Tokens elapsed	`mix1`	`mix2`	`tactic`	Pass-rate
Baselines					
`refl`					1.1%
`tidy-bfs`					9.9%
`WebMath > tactic`	1B			1.02	32.2%
Co-training (PACT)					
`WebMath > mix1 + tactic`	18B	*0.08*		0.94	40.0%
`WebMath > mix2 + tactic`	75B		*0.09*	0.93	46.0%
`WebMath > mix1 + mix2 + tactic`	71B	*0.09*	*0.09*	**0.91**	**48.4%**

- **Key insight: Training on tactic generation + auxiliary tasks is better than tactic generation alone**

[Wu et al., "Autoformalization with Large Language Models", 2022]
[Jiang et al., “Multilingual Mathematical Autoformalization”, 2023]

Autoformalization

- LLMs translate informal math into formal math
- Evaluation can be hard

Isabelle statement	GPT-4 informalisation
`lemma eint_minus_le: assumes "(b::eint) < c" shows "c - b > 0"`	The lemma named “eint_minus_le” assumes that an extended integer “b” is less than another extended integer “c”. It then shows that the result of “c” subtracted by “b” is greater than zero.
`lemma closed_superdiagonal: "closed {(x,y) | x y. x ≥ (y::('a::{linorder_topology}))}"`	The set of all pairs of elements (x, y) such that x is greater than or equal to y, is a closed set in the context of a linearly ordered topology.
Lean4 statement	**GPT-4 informalisation**
theorem norm_eq_one_of_pow_eq_one $\{\zeta : \mathbb{C}\}$ $\{n : \mathbb{N}\}$ (h : $\zeta^n = 1$) (hn : $n \neq 0$): $\|\zeta\| = 1$:=	For a complex number ζ and a natural number n, if ζ to the power of n equals 1 and n is not equal to 0, then the norm of ζ is equal to 1.
theorem mul_dvd_mul_iff_left $\{a\ b\ c : \mathbb{N}\}$ (ha : $0 < a$) : $a * b \mid a * c \leftrightarrow b \mid c$:=	For any three natural numbers a, b, and c, where a is greater than 0, a times b divides a times c if and only if b divides c.

Case Study 2 Question:
Natural Language version: "*When all the girls at Madeline's school line up in rows of eight, there are seven left over. If instead they line up in rows of four, how many are left over? The final answer is 3.*" Translate the natural language version to an Isabelle version:

PaLM output:

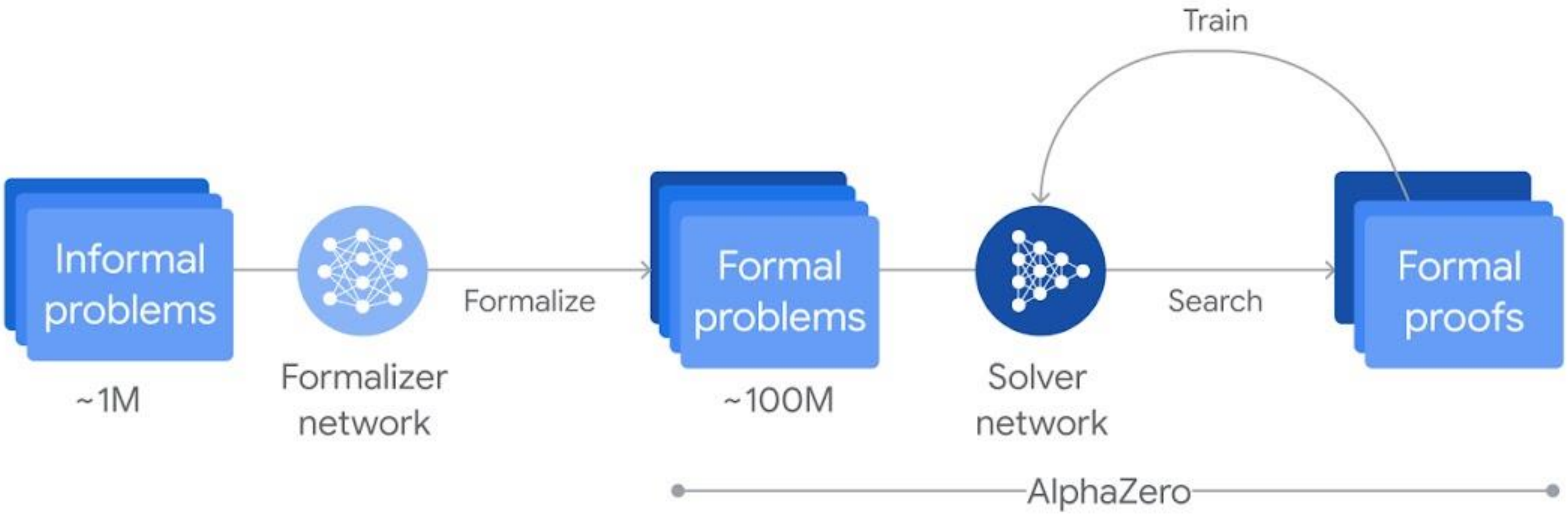
```
theorem
  fixes n::nat
  assumes "n mod 8 = 7"
  shows "n mod 4 = 3"
```

Case Study 3 Question:
Natural language version: "*Let f be a linear function for which $f(6) - f(2) = 12$. What is $f(12) - f(2)$? The final answer is 30.*" Translate the natural language version to an Isabelle version:

Codex output:

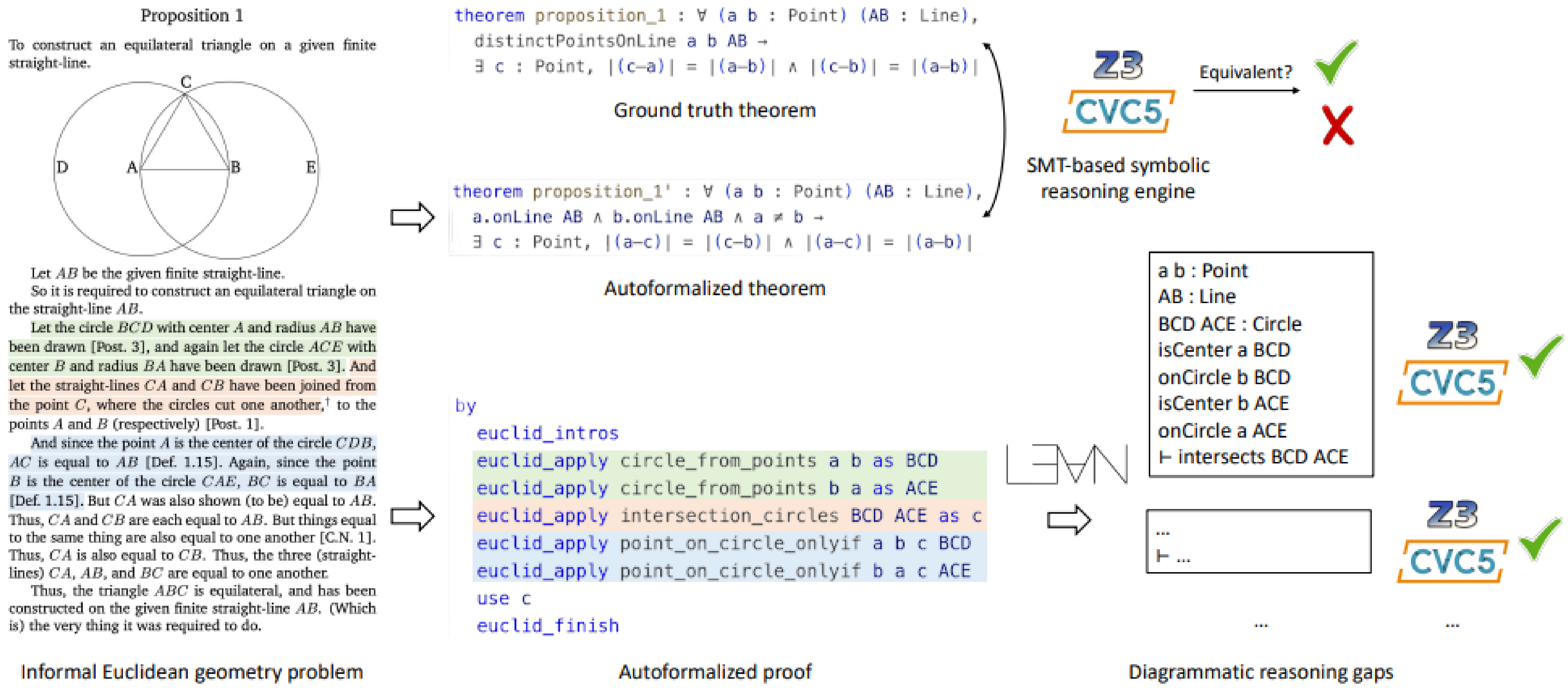
```
theorem
  fixes f :: "real \<Rightarrow> real"
  assumes "linear f"
          "f 6 - f 2 = 12"
  shows "f 12 - f 2 = 30"
```

Alpha Proof

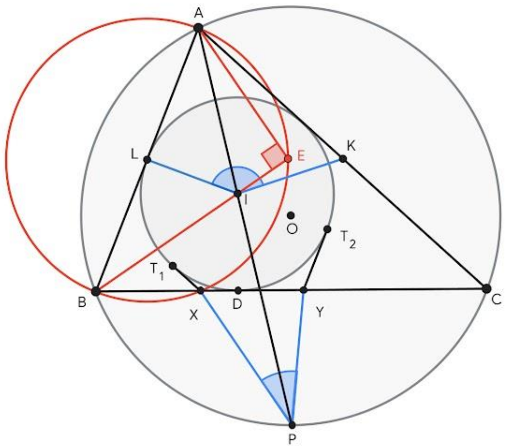

Train
Informal problems
~1M
Formalizer network
Formalize
Formal problems
~100M
Solver network
Search
Formal proofs
AlphaZero

[Murphy et al., "Autoformalizing Euclidean Geometry", 2024]

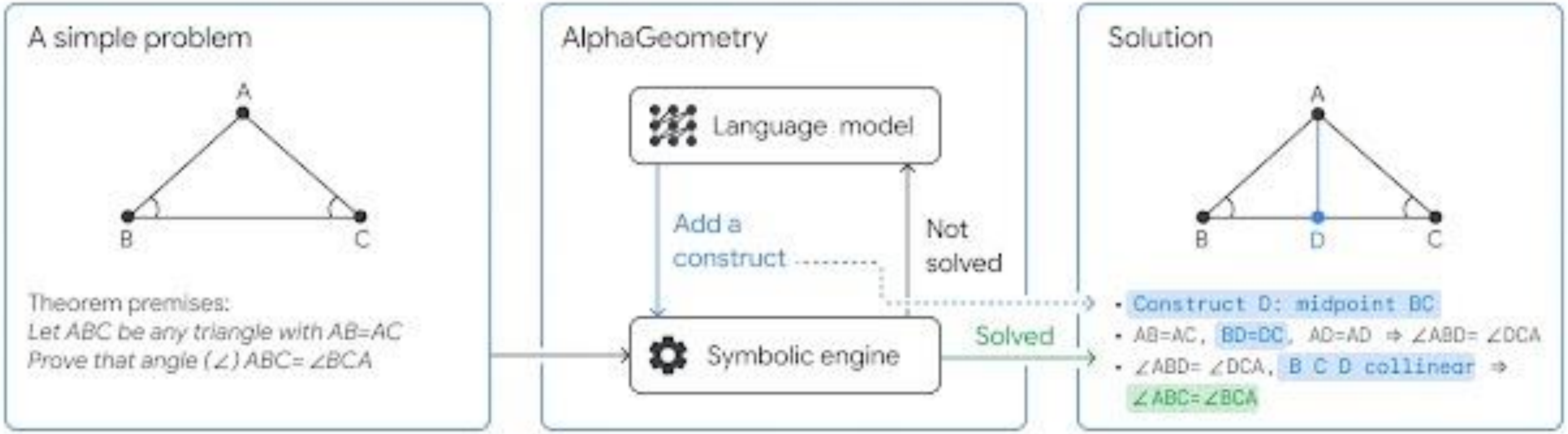
Logic Systems for Specific domains



Logic Systems for Specific domains

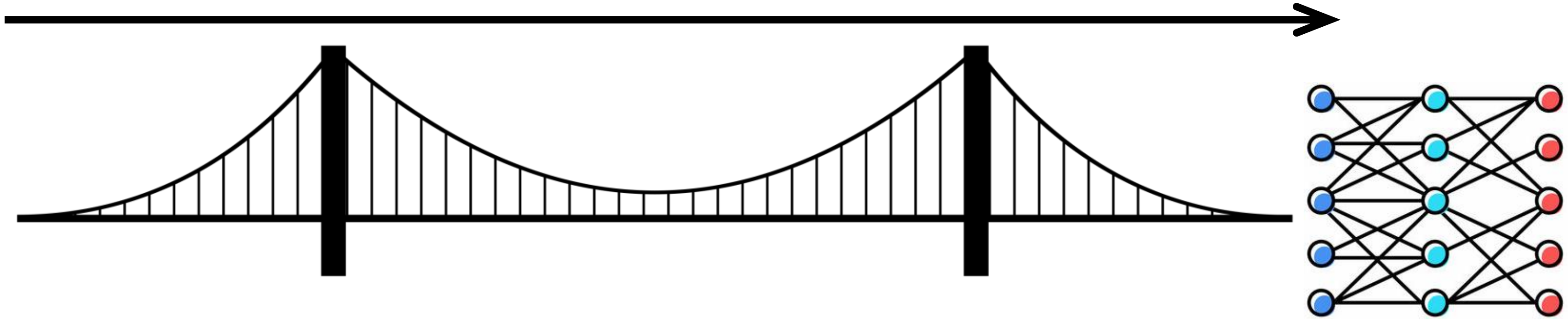


- Alpha Geometry → Alpha Geometry 2



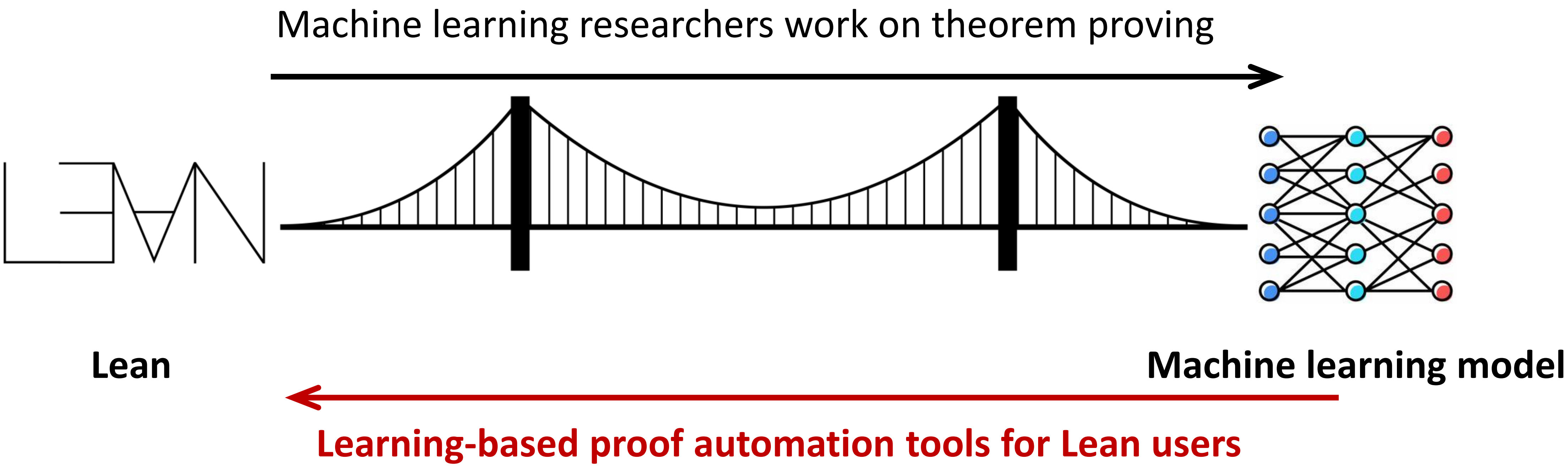
Bridging Machine Learning and Theorem Proving

Machine learning researchers work on theorem proving



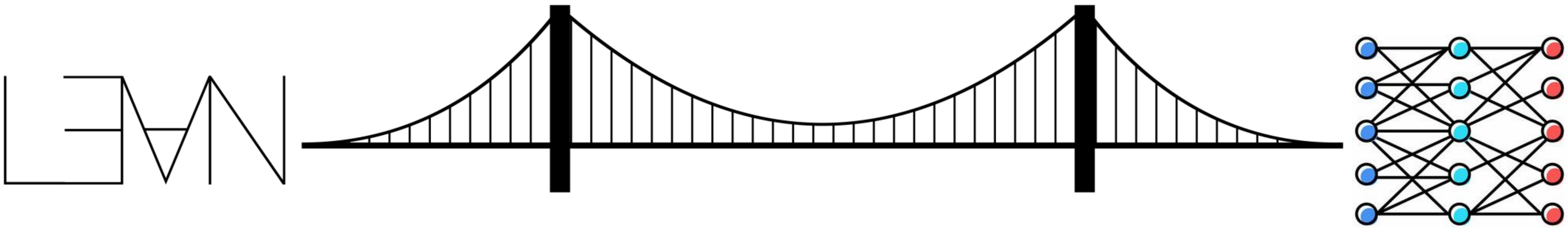
Lean **Machine learning model**

Bridging Machine Learning and Theorem Proving



Bridging Machine Learning and Theorem Proving

Machine learning researchers work on theorem proving

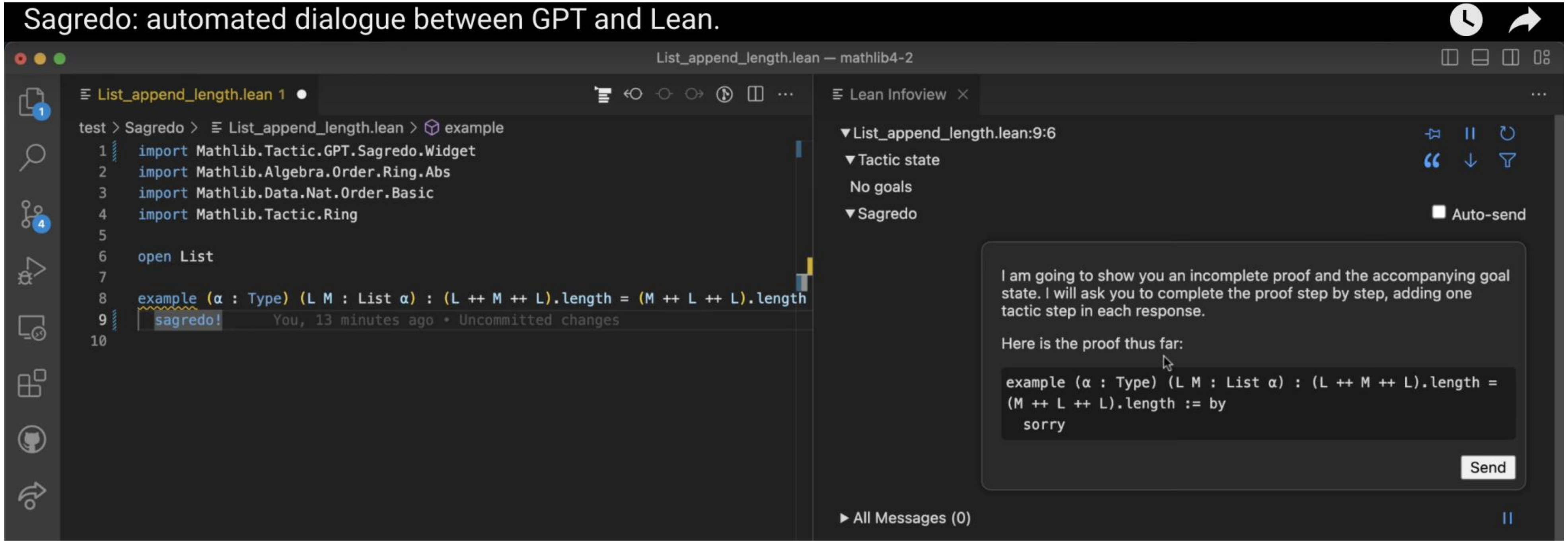


Lean **Machine learning model**

Learning-based proof automation tools for Lean users

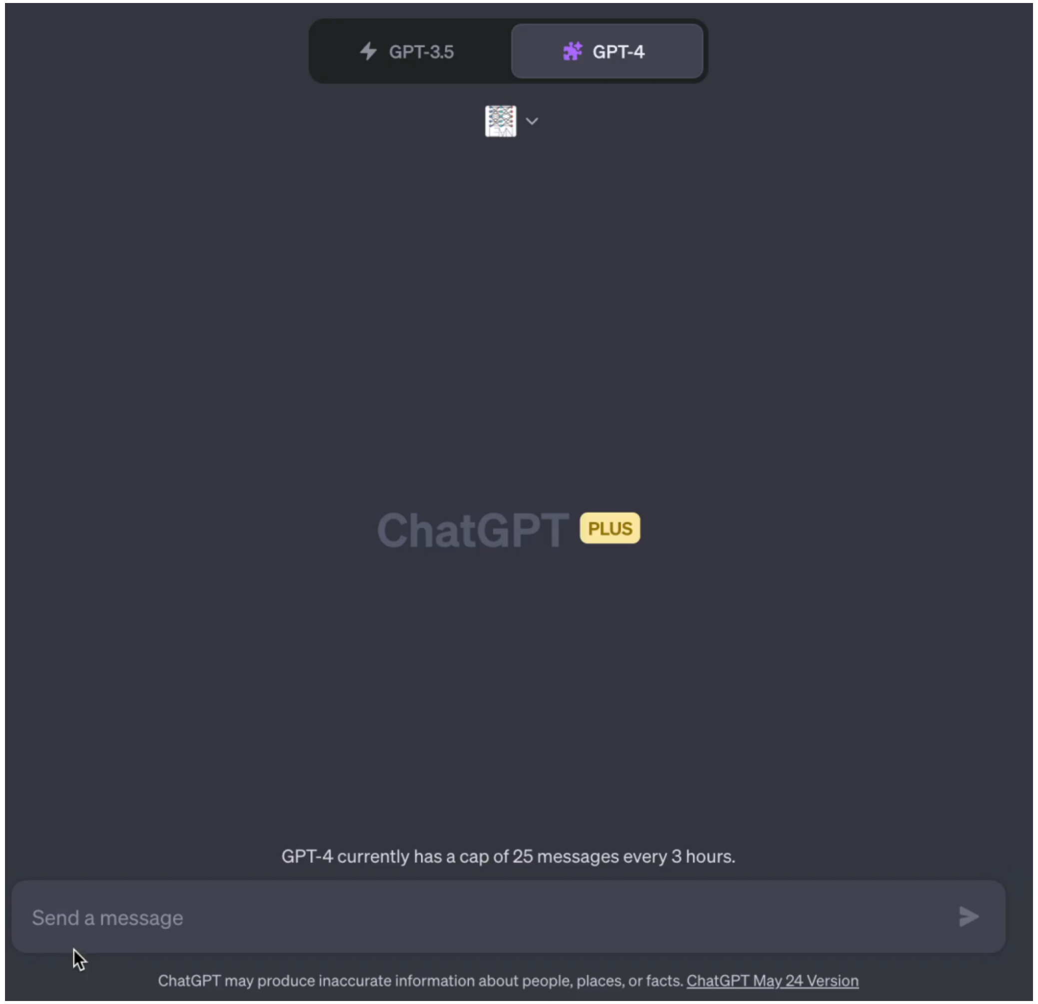
- Run on CPUs reasonably fast
- Integrated into VSCode
- Care about a specific domain, not aggregated performance on mathlib

Tools for Interfacing with GPT-4



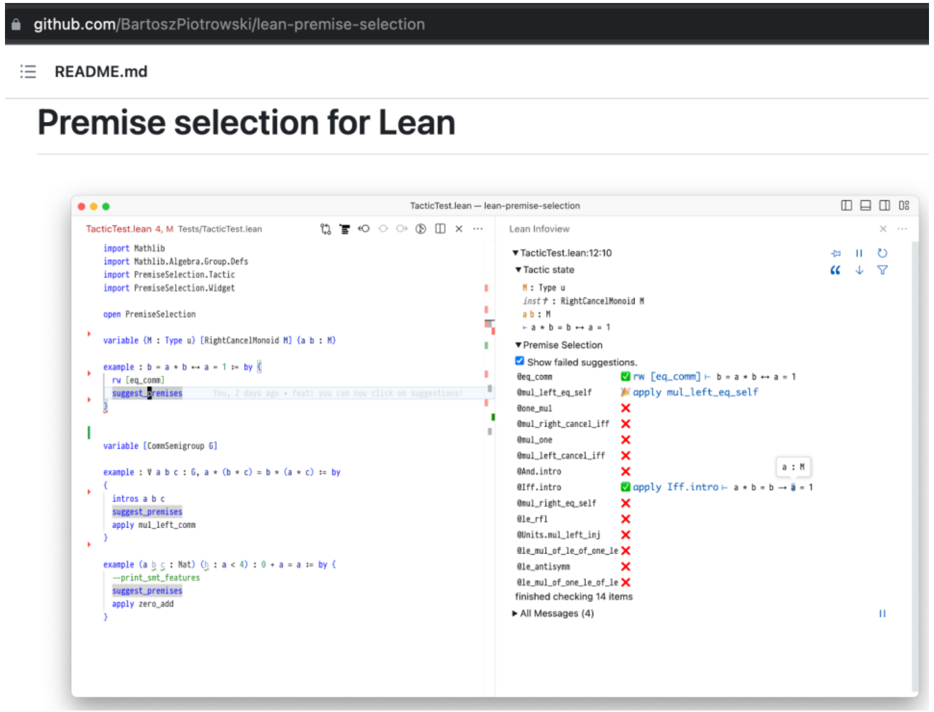
[Morrison et al., “Sagredo: automated dialogue between GPT and Lean”]
https://www.youtube.com/watch?v=CEwRMT0GpKo

ChatGPT Plugin for Theorem Proving in Lean



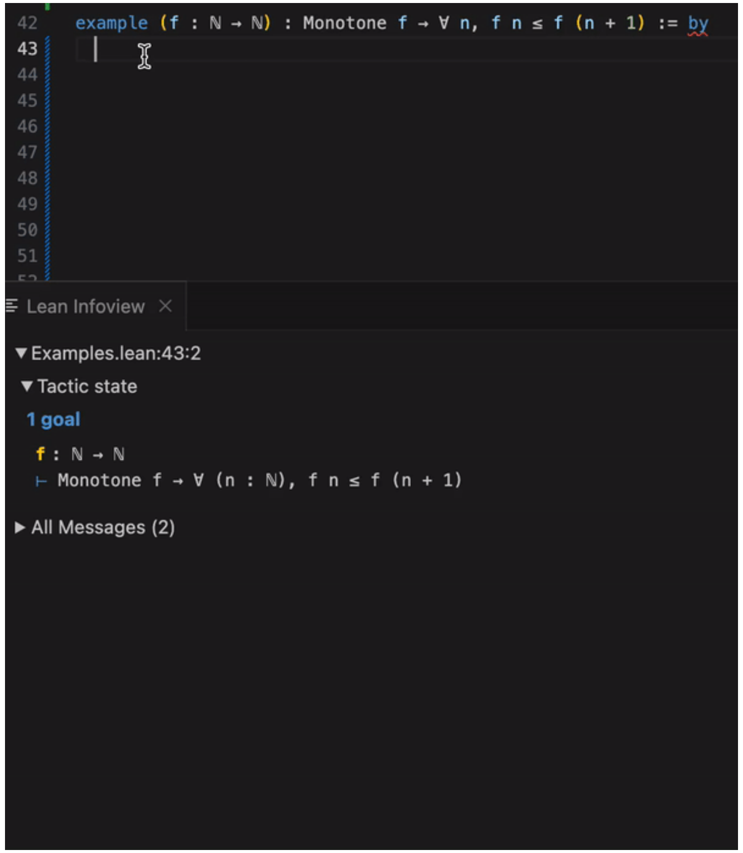
Tools for Premise Selection

- Built-in tactics such as library_search, apply?, exact?



[Piotrowski et al. "Machine-Learned Premise Selection for Lean"]
https://github.com/BartoszPiotrowski/lean-premise-selection

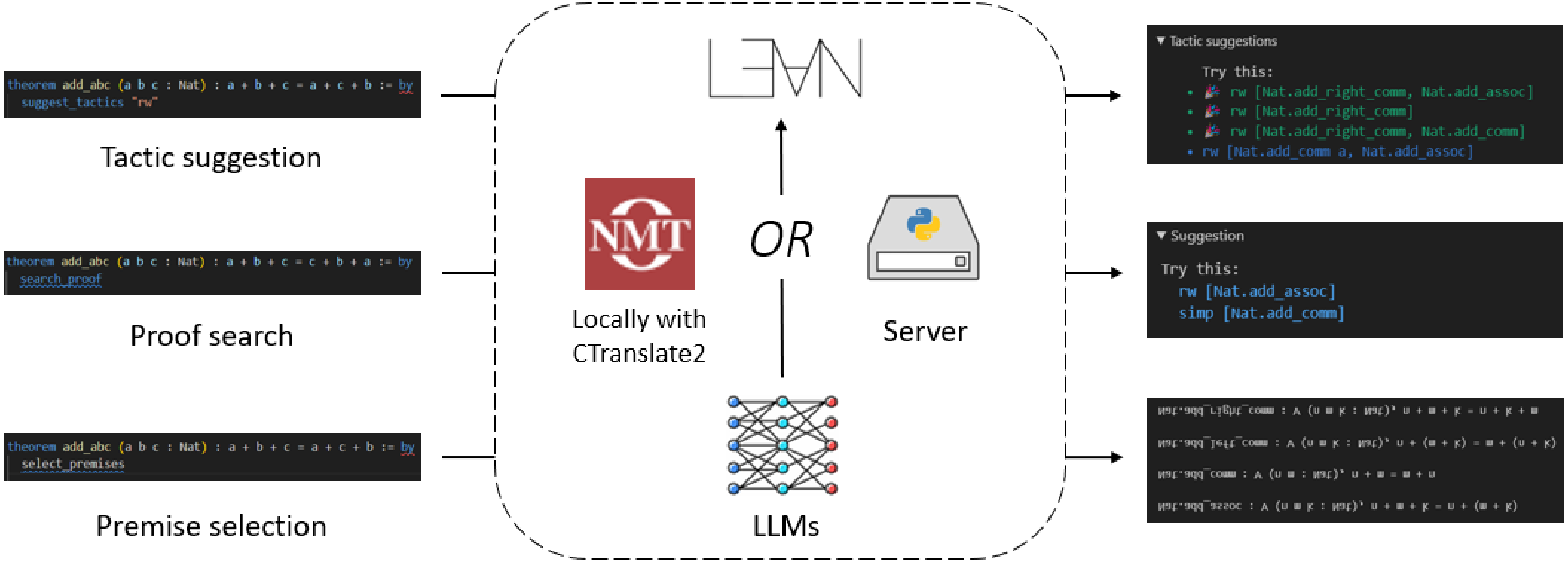
Tools for Tactic Suggestion



[Welleck and Saha, “llmstep: LLM proofstep suggestions in Lean”]
https://github.com/wellecks/llmstep

Lean Copilot Toolkit

Easy to install just like any Lean package
Run locally on most laptops w/o GPUs
Respond in seconds



[Song et al., "Towards Large Language Models as Copilots for Theorem Proving in Lean", NeurIPS MATH-AI, 2023]

Tactic Suggestion

Easy to install just like any Lean package
Run locally on most laptops w/o GPUs
Respond in seconds

```
import LeanCopilot

theorem add_abc (a b c : Nat) : a + b + c = a + c + b := by
  suggest_tactics
```

▼ Tactic state

No goals

▼ Suggestions

```
Try these:

• apply  Nat.add_right_comm
• rw [Nat.add_assoc ]
  Remaining subgoals:
  ⊢ a + (b + c) = a + c + b
• rw [Nat.add_comm ]
  Remaining subgoals:
  ⊢ c + (a + b) = a + c + b
• simp [Nat.add_assoc ]
  Remaining subgoals:
  ⊢ a + (b + c) = a + (c + b)
```

[Song et al., "Towards Large Language Models as Copilots for Theorem Proving in Lean", NeurIPS MATH-AI, 2023]

Proof Search

Easy to install just like any Lean package
Run locally on most laptops w/o GPUs
Respond in seconds

```
import LeanCopilot

theorem add_abc (a b c : Nat) : a + b + c = c + b + a := by
  search_proof
```

▼ Tactic state

No goals

▼ Suggestions

```
Try this:
  simp [Nat.add_comm,
    Nat.add_left_comm]
```

[Song et al., "Towards Large Language Models as Copilots for Theorem Proving in Lean", NeurIPS MATH-AI, 2023]

Premise Selection

Easy to install just like any Lean package
Run locally on most laptops w/o GPUs
Respond in seconds

```
import LeanCopilot

theorem add_abc (a b c : Nat) : a + b + c = a + c + b := by
  select_premises
```

```
Nat.add_assoc : ∀ (n m k : Nat), n + m + k = n + (m + k)

Nat.add_comm : ∀ (n m : Nat), n + m = m + n

Nat.add_left_comm : ∀ (n m k : Nat), n + (m + k) = m + (n + k)

Nat.add_right_comm : ∀ (n m k : Nat), n + m + k = n + k + m
```

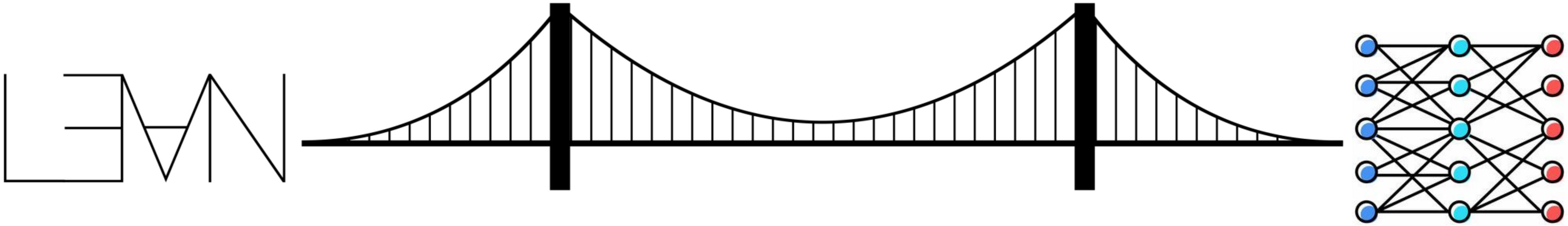
With rich annotations!

- In-scope premises: provide type information and doc strings
- Out-of-scope premises: provide complete definition + instruction on usage

[Song et al., "Towards Large Language Models as Copilots for Theorem Proving in Lean”, NeurIPS MATH-AI, 2023]

Bridging Machine Learning and Theorem Proving

Machine learning researchers work on theorem proving



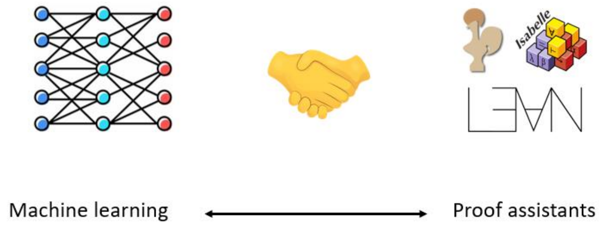
Lean

Machine learning model

Learning-based proof automation tools for Lean users

Neuro-Symbolic Theorem Proving with Lean

- LeanDojo: Theorem Proving with Retrieval-Augmented Language Models
  - LeanDojo: Data Extraction & Interaction Tool for Theorem Proving in Lean
  - ReProver: Retrieval-Augmented Language Model as Theorem Prover



- Towards Large Language Models as Copilots for Theorem Proving in Lean
  - Lean Copilot: Native Machine Learning Toolkit in Lean
  - LLM-Powered Tools for Tactic Suggestion, Proof Search & Premise Selection

Neuro-Symbolic Theorem Proving with Lean

Happy to take questions!

Peiyang Song
California Institute of Technology
psong@caltech.edu