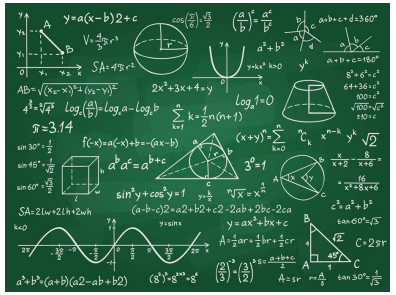
LeanDojo: Theorem Proving with Retrieval-Augmented Language Models

Kaiyu Yang

Postdoc @ Computing + Mathematical Sciences

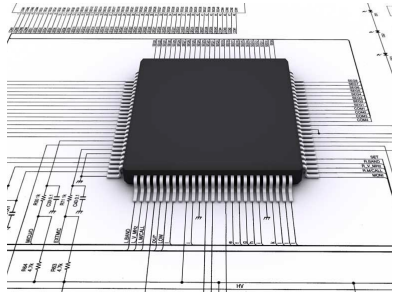
Caltech

Automated Reasoning and Formal Proofs



Formal mathematics

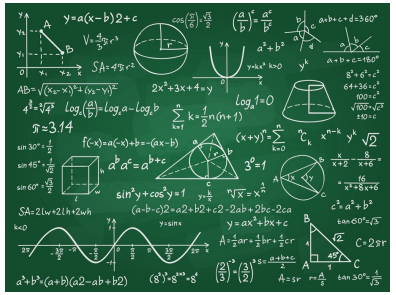
Software verification



Hardware verification

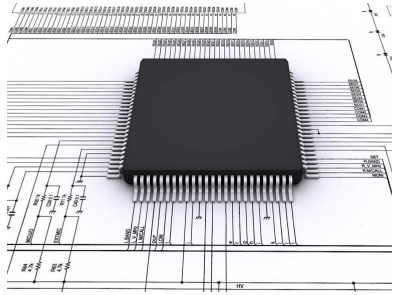
Cyber-physical systems

Automated Reasoning and Formal Proofs



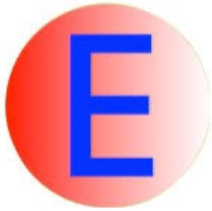
Formal mathematics

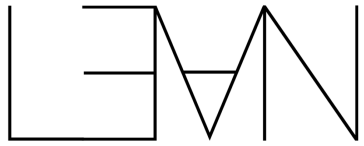
Software verification



Hardware verification

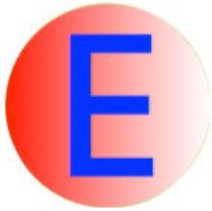
Cyber-physical systems





Automated Reasoning and Formal Proofs

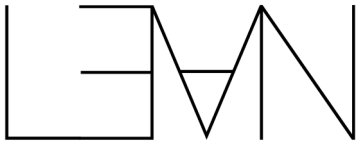
- **Automated theorem proving**
  - SMT solvers, model checkers, ATP systems in first-order logic, etc.
  - Minimal efforts from humans
  - Limited expressiveness
  - Difficult to scale




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

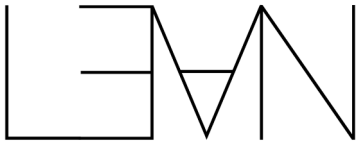
[Hales et al., "A Formal Proof of the Kepler Conjecture", 2017]
[Klein et al., "seL4: Formal Verification of an OS Kernel", 2009]
[Leroy, "Formal Verification of a Realistic Compiler", 2008]

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**

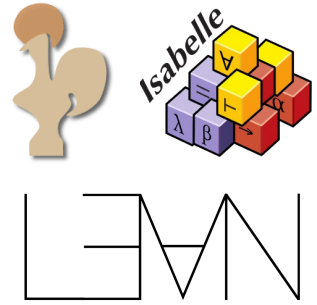
[Hales et al., "A Formal Proof of the Kepler Conjecture", 2017]
[Klein et al., "seL4: Formal Verification of an OS Kernel", 2009]
[Leroy, "Formal Verification of a Realistic Compiler", 2008]


Theorem Proving in Proof Assistants

Human

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



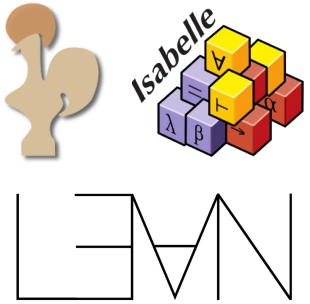
Proof assistant

Theorem Proving in Proof Assistants

Human

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

```
n : ℕ
⊢ gcd n n = n
```

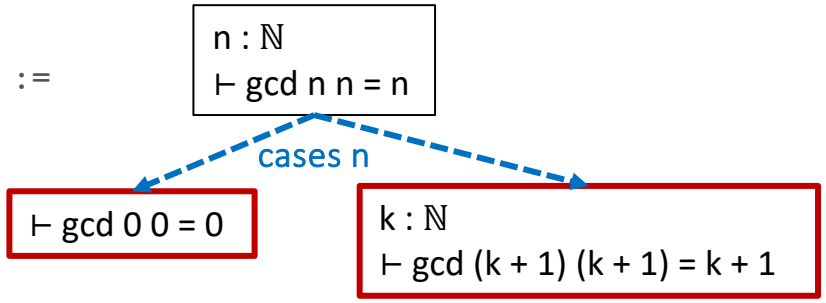


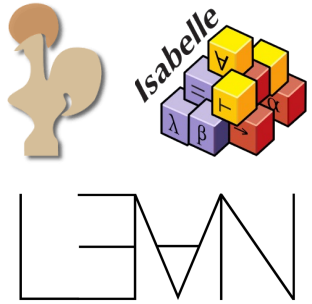
Proof assistant

Theorem Proving in Proof Assistants

Human

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



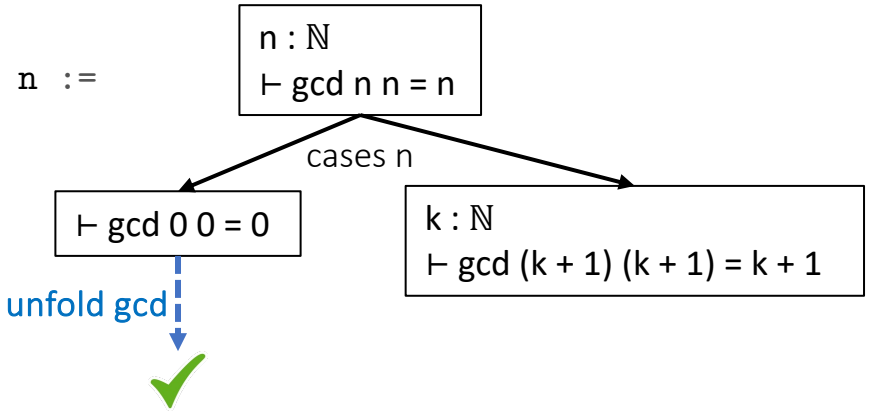


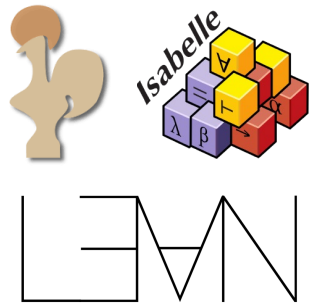
Proof assistant

Theorem Proving in Proof Assistants

Human

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



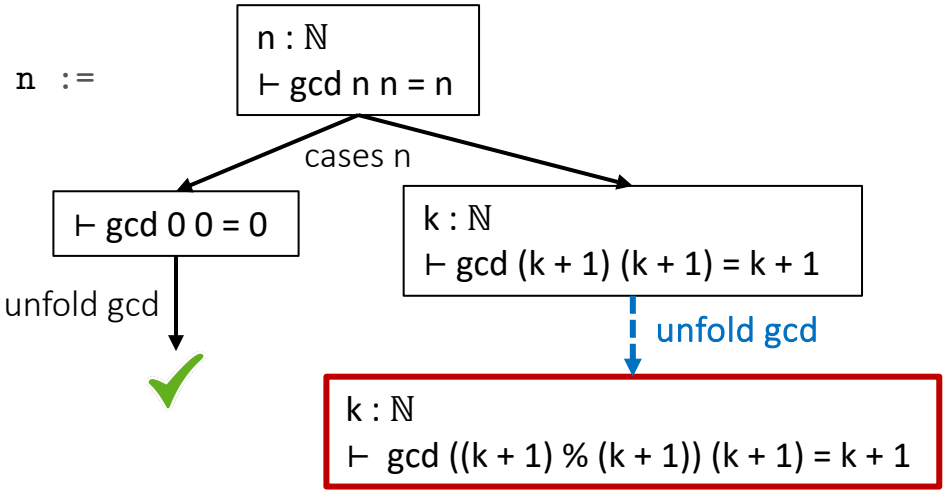


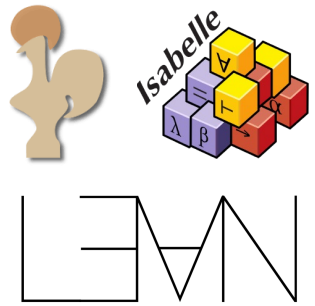
Proof assistant

Theorem Proving in Proof Assistants

Human

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



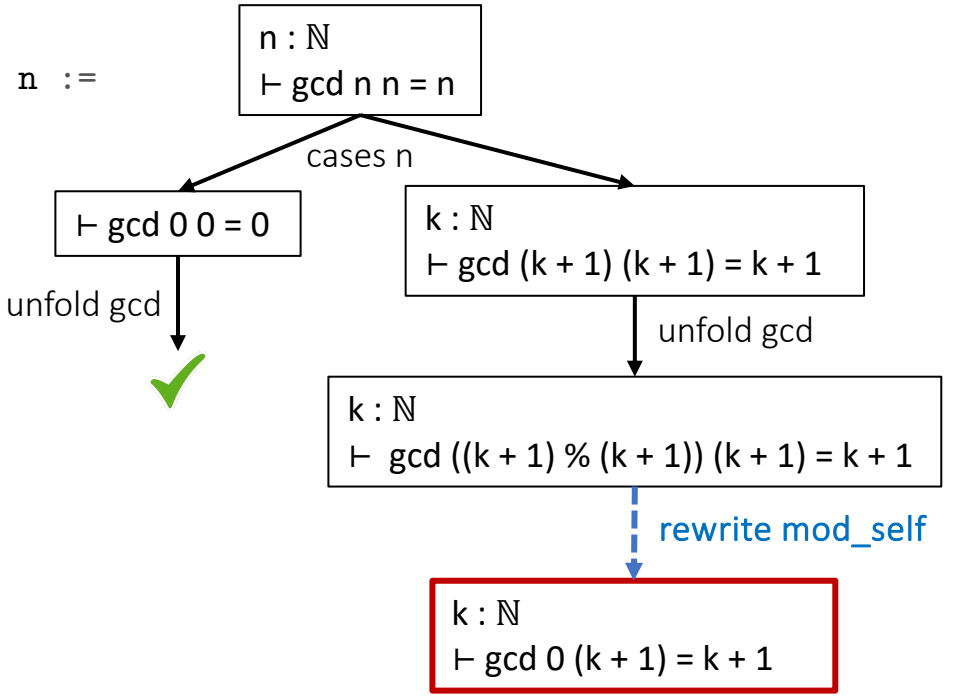


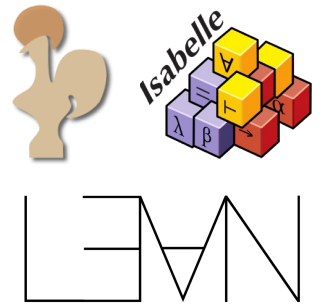
Proof assistant

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,
```



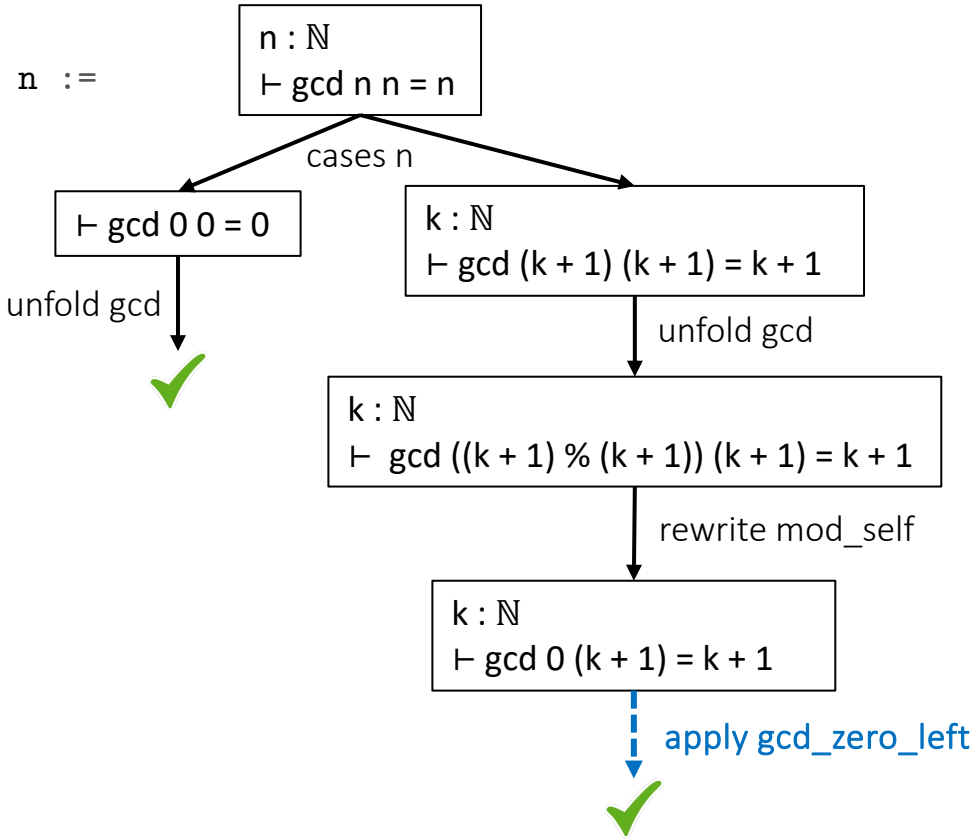


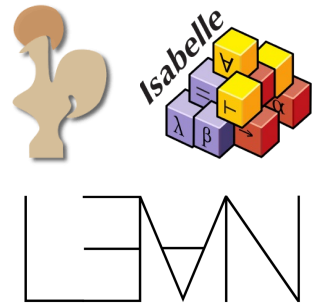
Proof assistant

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
```



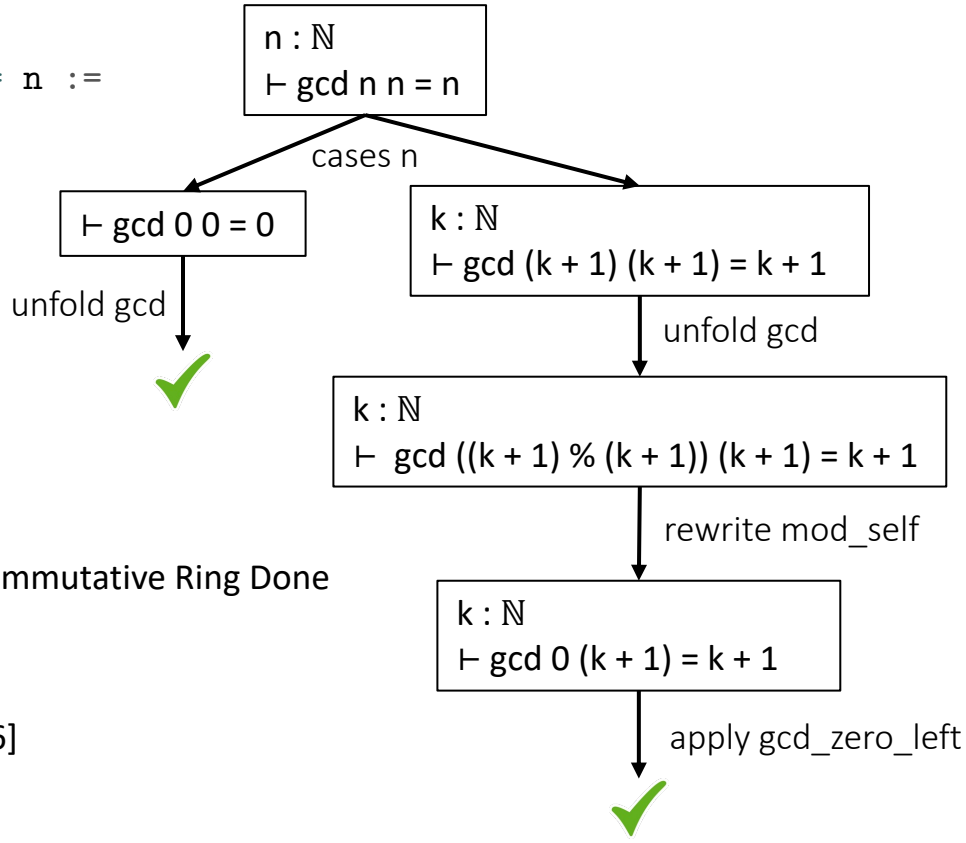


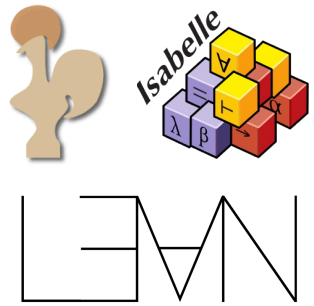
Proof assistant

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

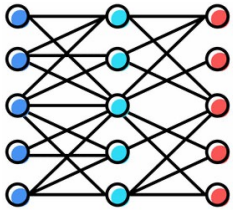
  [Blanchette, et al., "Hammering towards QED", 2016]
- Proof search within the proof assistant

  Coq's auto tactic

  [Limperg and From, "Aesop: White-Box Best-First Proof Search for Lean", 2023]

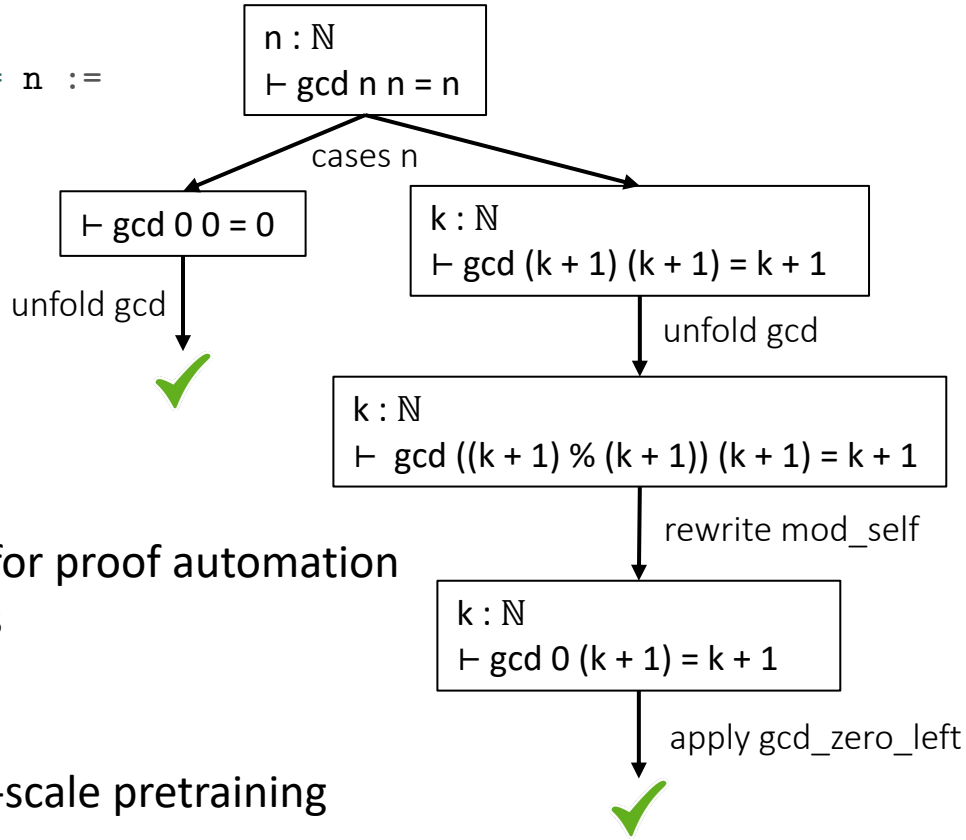
Theorem Proving in Proof Assistants

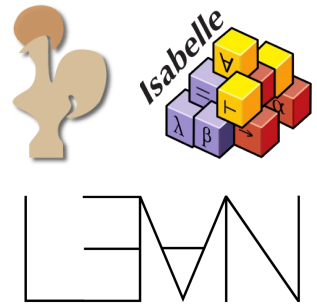
Human



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
```



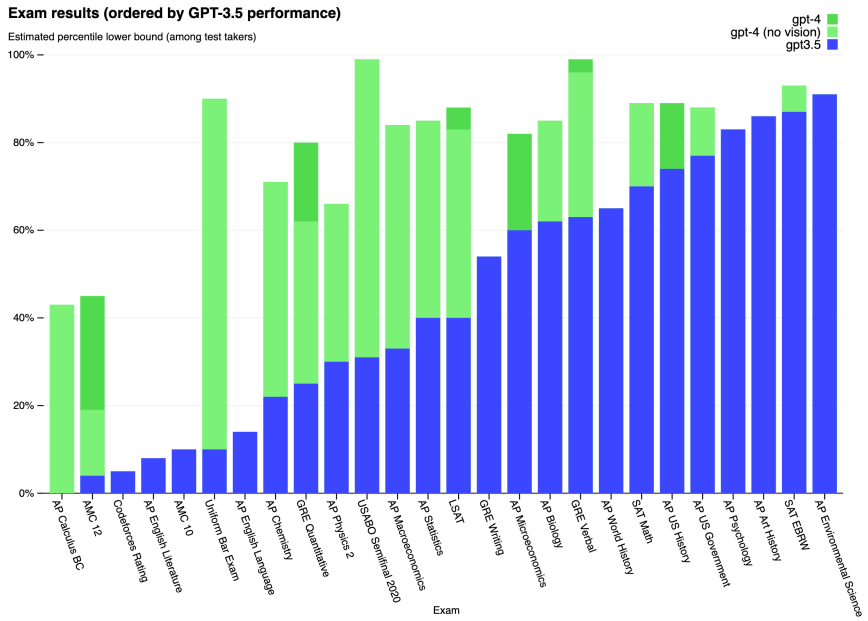


Proof assistant

- 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)

Large Language Models (LLMs) for Math

- GPT-4: 89th percentile in SAT Math among human test takers



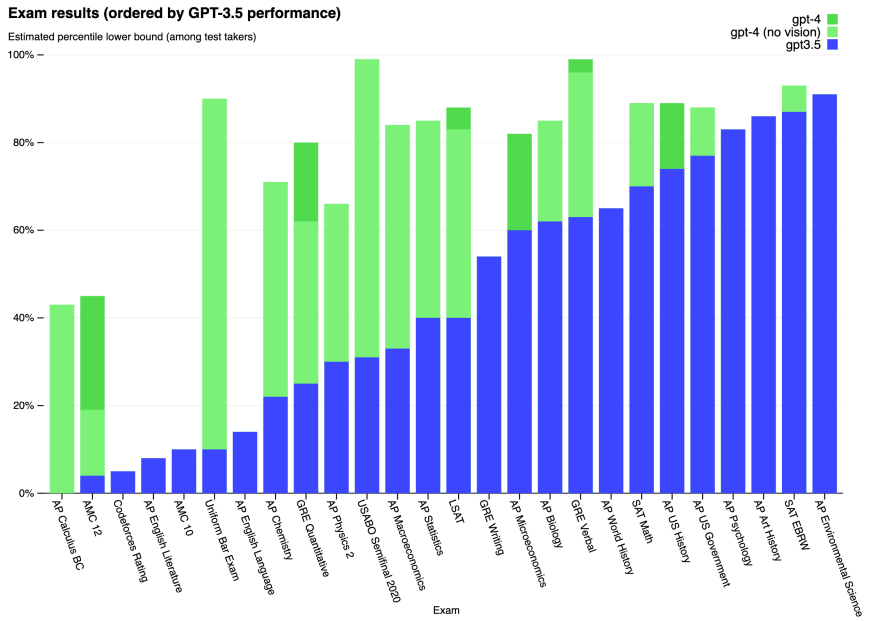
[OpenAI, "GPT-4 Technical Report", 2023]

Large Language Models (LLMs) for Math

- GPT-4: 89th percentile in SAT Math among human test takers
- Minerva: Google's LLM specialized in math

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

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

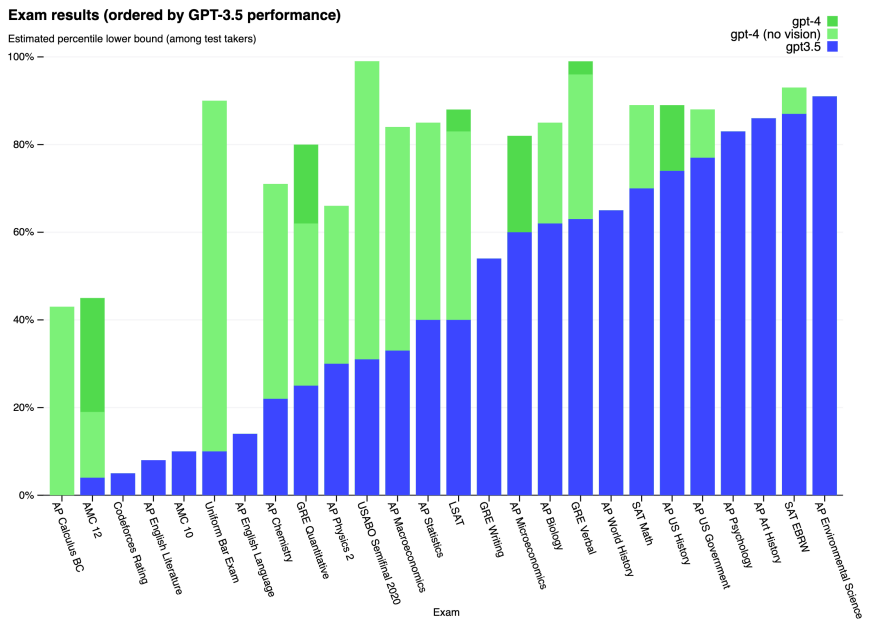


[OpenAI, "GPT-4 Technical Report", 2023]

[Lewkowycz et al., "Solving Quantitative Reasoning Problems with Language Models", 2022]

Large Language Models (LLMs) for Math

- GPT-4: 89th percentile in SAT Math among human test takers
- Minerva: Google's LLM specialized in math



[OpenAI, "GPT-4 Technical Report", 2023]

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

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

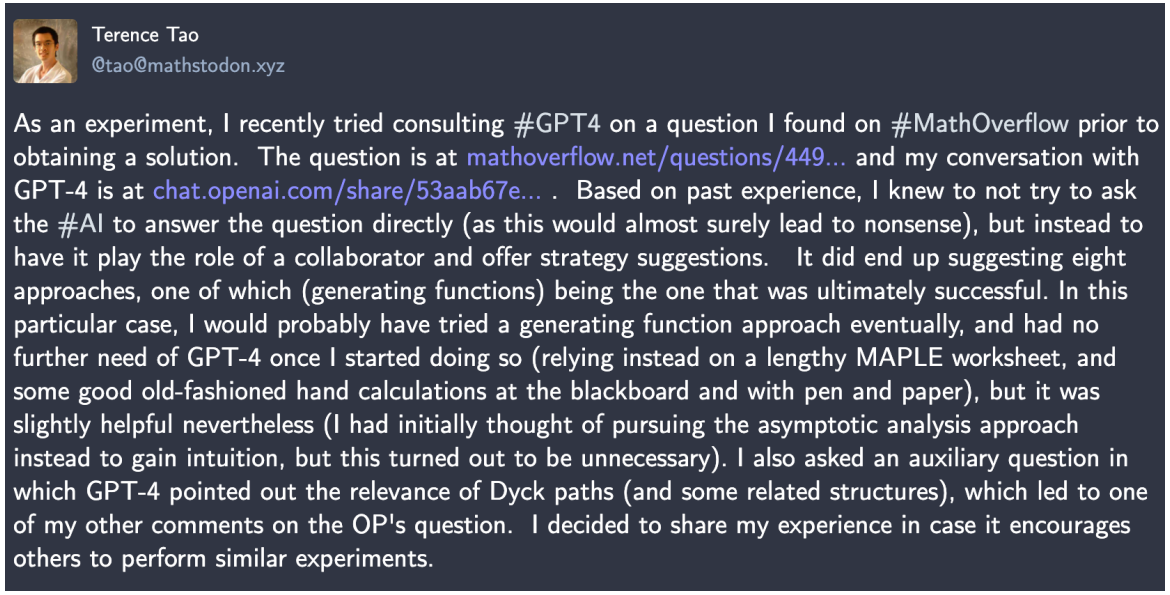
Model output:

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

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

[Lewkowycz et al., "Solving Quantitative Reasoning Problems with Language Models", 2022]

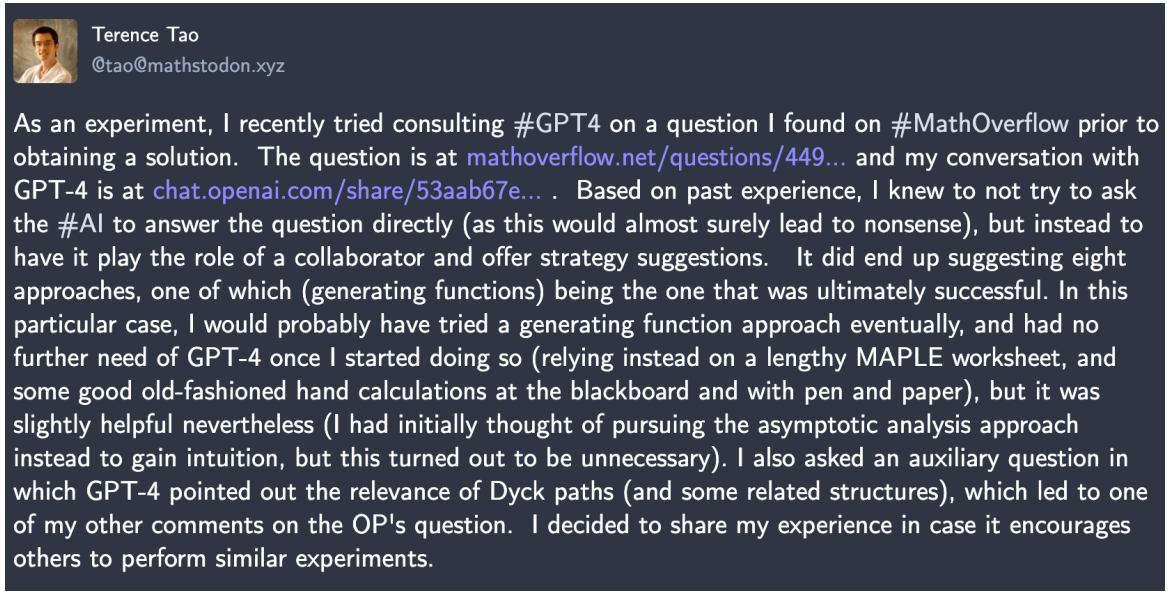
Large Language Models (LLMs) for Math



Terence Tao
@tao@mathstodon.xyz

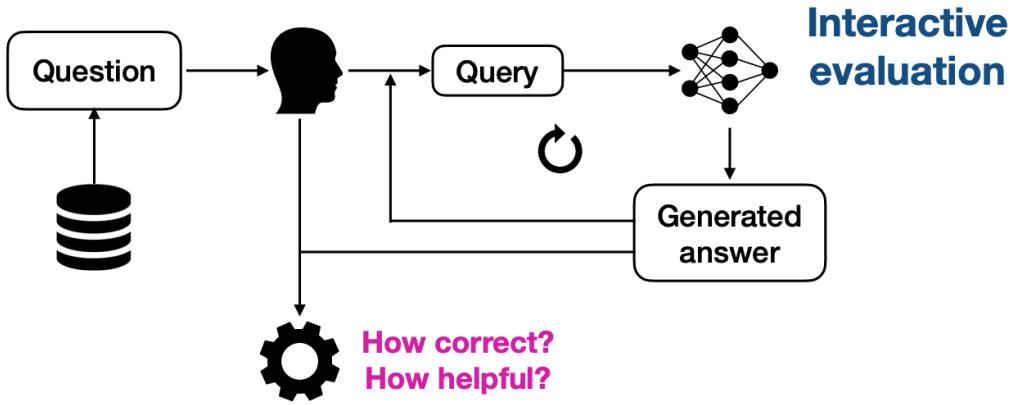
As an experiment, I recently tried consulting #GPT4 on a question I found on #MathOverflow prior to obtaining a solution. The question is at mathoverflow.net/questions/449... and my conversation with GPT-4 is at chat.openai.com/share/53aab67e... . Based on past experience, I knew to not try to ask the #AI to answer the question directly (as this would almost surely lead to nonsense), but instead to have it play the role of a collaborator and offer strategy suggestions. It did end up suggesting eight approaches, one of which (generating functions) being the one that was ultimately successful. In this particular case, I would probably have tried a generating function approach eventually, and had no further need of GPT-4 once I started doing so (relying instead on a lengthy MAPLE worksheet, and some good old-fashioned hand calculations at the blackboard and with pen and paper), but it was slightly helpful nevertheless (I had initially thought of pursuing the asymptotic analysis approach instead to gain intuition, but this turned out to be unnecessary). I also asked an auxiliary question in which GPT-4 pointed out the relevance of Dyck paths (and some related structures), which led to one of my other comments on the OP's question. I decided to share my experience in case it encourages others to perform similar experiments.

Large Language Models (LLMs) for Math



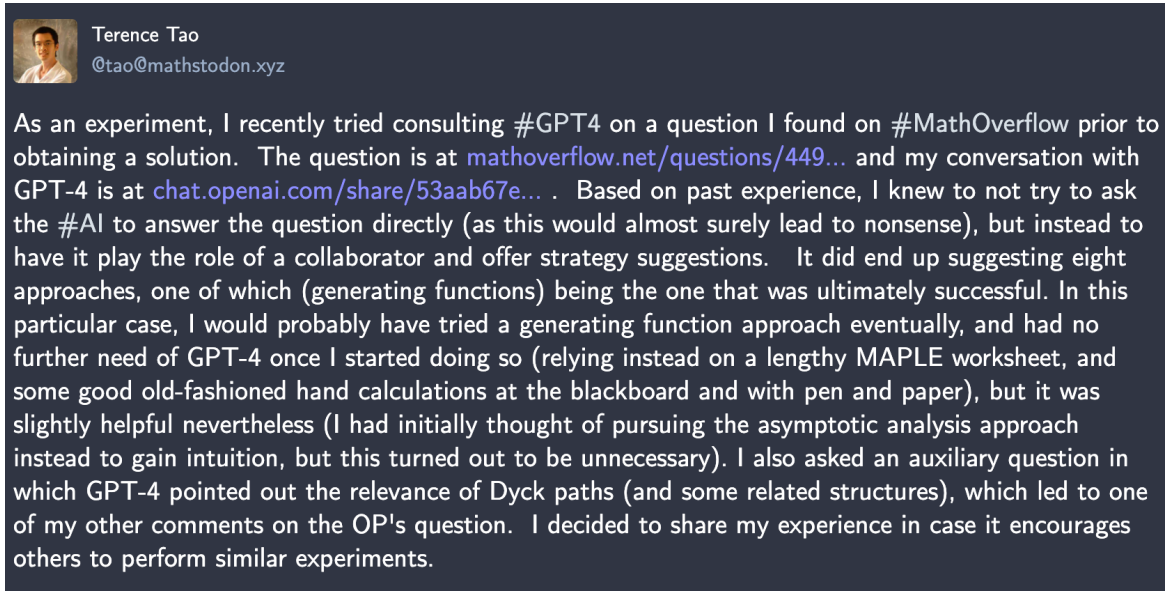
Terence Tao
@tao@mathstodon.xyz

As an experiment, I recently tried consulting #GPT4 on a question I found on #MathOverflow prior to obtaining a solution. The question is at mathoverflow.net/questions/449... and my conversation with GPT-4 is at chat.openai.com/share/53aab67e... . Based on past experience, I knew to not try to ask the #AI to answer the question directly (as this would almost surely lead to nonsense), but instead to have it play the role of a collaborator and offer strategy suggestions. It did end up suggesting eight approaches, one of which (generating functions) being the one that was ultimately successful. In this particular case, I would probably have tried a generating function approach eventually, and had no further need of GPT-4 once I started doing so (relying instead on a lengthy MAPLE worksheet, and some good old-fashioned hand calculations at the blackboard and with pen and paper), but it was slightly helpful nevertheless (I had initially thought of pursuing the asymptotic analysis approach instead to gain intuition, but this turned out to be unnecessary). I also asked an auxiliary question in which GPT-4 pointed out the relevance of Dyck paths (and some related structures), which led to one of my other comments on the OP's question. I decided to share my experience in case it encourages others to perform similar experiments.



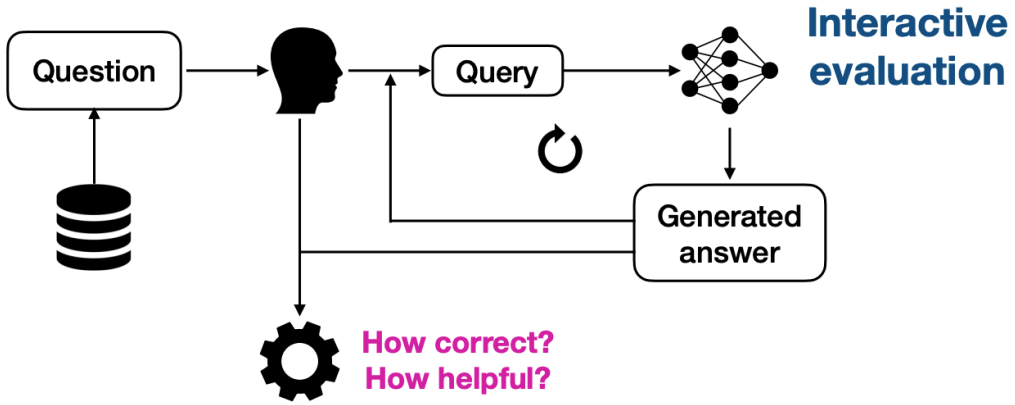
[Collins et al., "Evaluating Language Models for Mathematics through Interactions", 2023]

Large Language Models (LLMs) for Math



Terence Tao
@tao@mathstodon.xyz

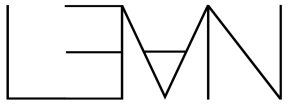
As an experiment, I recently tried consulting #GPT4 on a question I found on #MathOverflow prior to obtaining a solution. The question is at mathoverflow.net/questions/449... and my conversation with GPT-4 is at chat.openai.com/share/53aab67e... . Based on past experience, I knew to not try to ask the #AI to answer the question directly (as this would almost surely lead to nonsense), but instead to have it play the role of a collaborator and offer strategy suggestions. It did end up suggesting eight approaches, one of which (generating functions) being the one that was ultimately successful. In this particular case, I would probably have tried a generating function approach eventually, and had no further need of GPT-4 once I started doing so (relying instead on a lengthy MAPLE worksheet, and some good old-fashioned hand calculations at the blackboard and with pen and paper), but it was slightly helpful nevertheless (I had initially thought of pursuing the asymptotic analysis approach instead to gain intuition, but this turned out to be unnecessary). I also asked an auxiliary question in which GPT-4 pointed out the relevance of Dyck paths (and some related structures), which led to one of my other comments on the OP's question. I decided to share my experience in case it encourages others to perform similar experiments.



LLMs can be useful for theorem proving in Lean

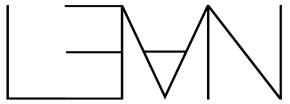
[Collins et al., "Evaluating Language Models for Mathematics through Interactions", 2023]

Breaking the Barriers in LLMs for Theorem Proving



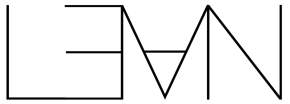
Jiang et al., LISA, 2021
Jiang et al., Thor, 2022
First et al., Baldur, 2023
Polu and Sutskever, GPT-f, 2020
Han et al., PACT, 2022
Polu et al., 2023
Lample et al., HTPS 2022
Wang et al., DT-Solver, 2023

Breaking the Barriers in LLMs for Theorem Proving



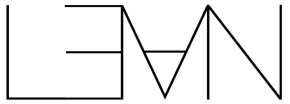
	Dataset available
Jiang et al., LISA, 2021	✓
Jiang et al., Thor, 2022	✓
First et al., Baldur, 2023	✗
Polu and Sutskever, GPT-f, 2020	✗
Han et al., PACT, 2022	✗
Polu et al., 2023	✗
Lample et al., HTPS 2022	✗
Wang et al., DT-Solver, 2023	✓

Breaking the Barriers in LLMs for Theorem Proving



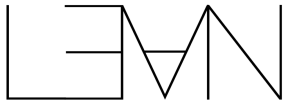
	Dataset available	Model available	Code available
Jiang et al., LISA, 2021	✓	✗	✗
Jiang et al., Thor, 2022	✓	✗	✗
First et al., Baldur, 2023	✗	✗	✗
Polu and Sutskever, GPT-f, 2020	✗	✗	✗
Han et al., PACT, 2022	✗	✗	✗
Polu et al., 2023	✗	✗	✗
Lample et al., HTPS 2022	✗	✗	✗
Wang et al., DT-Solver, 2023	✓	✗	✗

Breaking the Barriers in LLMs for Theorem Proving



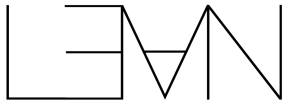
	Dataset available	Model available	Code available	Interaction tool available
Jiang et al., LISA, 2021	✓	✗	✗	✓
Jiang et al., Thor, 2022	✓	✗	✗	✓
First et al., Baldur, 2023	✓	✗	✗	✓
Polu and Sutskever, GPT-f, 2020	✗	✗	✗	✗
Han et al., PACT, 2022	✓	✗	✗	✓
Polu et al., 2023	✓	✗	✗	✓
Lample et al., HTPS 2022	✓	✗	✗	✗
Wang et al., DT-Solver, 2023	✓	✗	✗	✗

Breaking the Barriers in LLMs for Theorem Proving



	Dataset available	Model available	Code available	Interaction tool available	Model size (# params)	Compute (hours)
Jiang et al., LISA, 2021	✓	✗	✗	✓	163M	-
Jiang et al., Thor, 2022	✓	✗	✗	✓	700M	1K on TPU
First et al., Baldur, 2023	✗	✗	✗	✓	62,000M	-
Polu and Sutskever, GPT-f, 2020	✗	✗	✗	✗	774M	40K on GPU
Han et al., PACT, 2022	✗	✗	✗	✓	837M	1.5K on GPU
Polu et al., 2023	✗	✗	✗	✓	774M	48K on GPU
Lample et al., HTPS 2022	✗	✗	✗	✗	600M	34K on GPU
Wang et al., DT-Solver, 2023	✓	✗	✗	✗	774M	1K on GPU

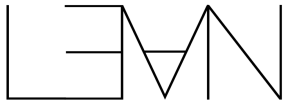
Breaking the Barriers in LLMs for Theorem Proving



	Dataset available	Model available	Code available	Interaction tool available	Model size (# params)	Compute (hours)
Jiang et al., LISA, 2021	✓	✗	✗	✓	163M	-
Jiang et al., Thor, 2022	✓	✗	✗	✓	700M	1K on TPU
First et al., Baldur, 2023	✓	✗	✗	✓	62,000M	-
Polu and Sutskever, GPT-f, 2020	✗	✗	✗	✗	774M	40K on GPU
Han et al., PACT, 2022	✓	✗	✗	✓	837M	1.5K on GPU
Polu et al., 2023	✓	✗	✗	✓	774M	48K on GPU
Lample et al., HTPS 2022	✓	✗	✗	✗	600M	34K on GPU
Wang et al., DT-Solver, 2023	✓	✗	✗	✗	774M	1K on GPU
LeanDojo (ours)	✓	✓	✓	✓	**517M**	**120 on GPU**

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

Breaking the Barriers in LLMs for Theorem Proving

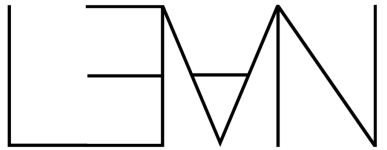


	Dataset available	Model available	Code available	Interaction tool available	Model size (# params)	Compute (hours)
Jiang et al., LISA, 2021	✓	✗	✗	✓	163M	-
Jiang et al., Thor, 2022	✓	✗	✗	✓	700M	1K on TPU
First et al., Baldur, 2023	✓*	✗	✗	✓	62,000M	-
Polu and Sutskever, GPT-f, 2020	✗	✗	✗	✗	774M	40K on GPU
Han et al., PACT, 2022	✓*	✗	✗	✓	837M	1.5K on GPU
Polu et al., 2023	✓*	✗	✗	✓	774M	48K on GPU
Lample et al., HTPS 2022	✓*	✗	✗	✗	600M	34K on GPU
Wang et al., DT-Solver, 2023	✓	✗	✗	✗	774M	1K on GPU
LeanDojo (ours)	✓	✓	✓	✓	**517M**	**120 on GPU**

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

Give researchers access to state-of-the-art LLM-based provers with modest computational costs

The Lean 4 Proof Assistant



- Type-theoretical foundation [Carneiro, "The Type Theory of Lean", 2019]
  - Similar to Coq, Calculus of Inductive Constructions (CIC)
  - Minor differences, e.g., proof irrelevance, more liberal use of classical axioms

- Ecosystem
  - Strong in formalized mathematics, e.g., algebra, algebraic geometry, etc.
  - Mathlib: Lean's centralized mathematical library
  - Lacking in formal verification, e.g., floating-point arithmetic

Lean as a General-Purpose Programming Language

- Lean 4 resembles Haskell with dependent types [Christiansen, "Functional Programming in Lean 4"]
- Environments, local contexts, and goals implemented within Lean as a hierarchy of state monads

Environment
- Definitions, lemmas, etc.

Local context
- Local hypotheses & goals

Current goals

Lean as a General-Purpose Programming Language

- Lean 4 resembles Haskell with dependent types [Christiansen, "Functional Programming in Lean 4"]
- Environments, local contexts, and goals implemented within Lean as a hierarchy of state monads

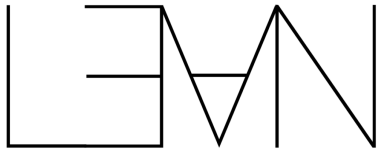
Environment
- Definitions, lemmas, etc.

Local context
- Local hypotheses & goals

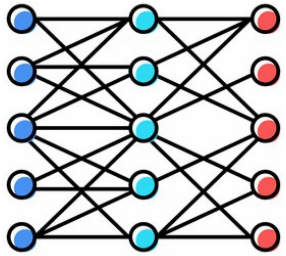
Current goals

- Strong support for metaprogramming
  - Write tactics directly in Lean, no Ltac or Ocaml
  - Access to Lean's frontend (parser and elaborator) [de Moura et al., "Elaboration in Dependent Type Theory", 2015]
- Convenient for machine learning

LeanDojo

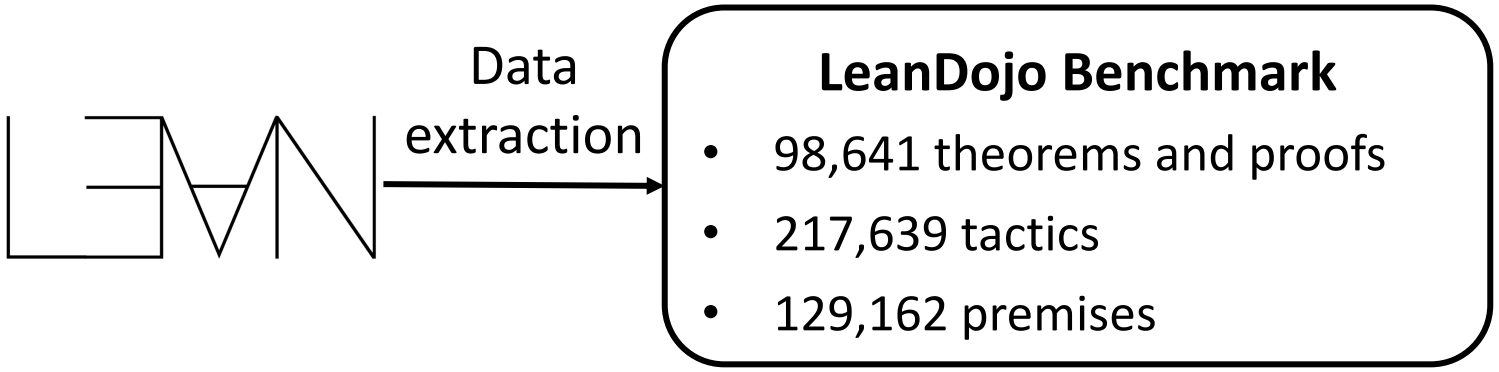


Lean

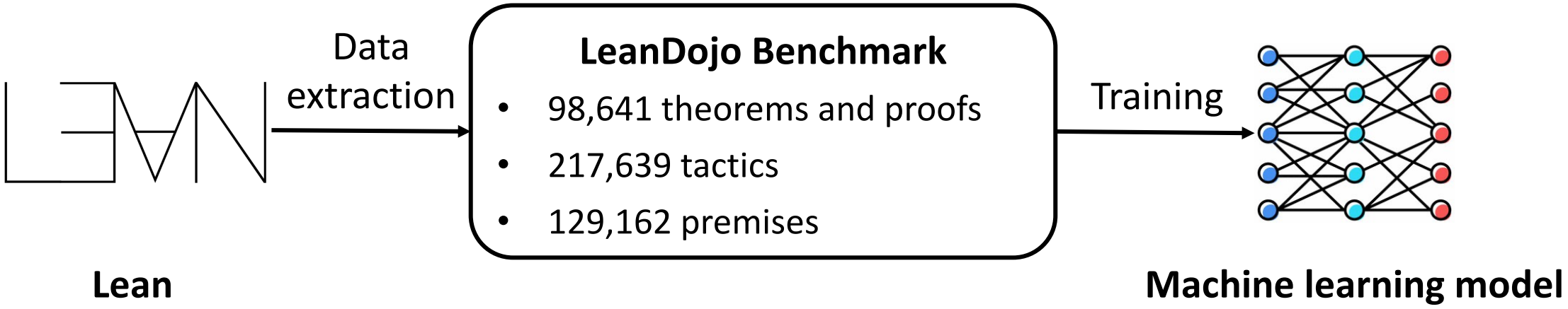


Machine learning model

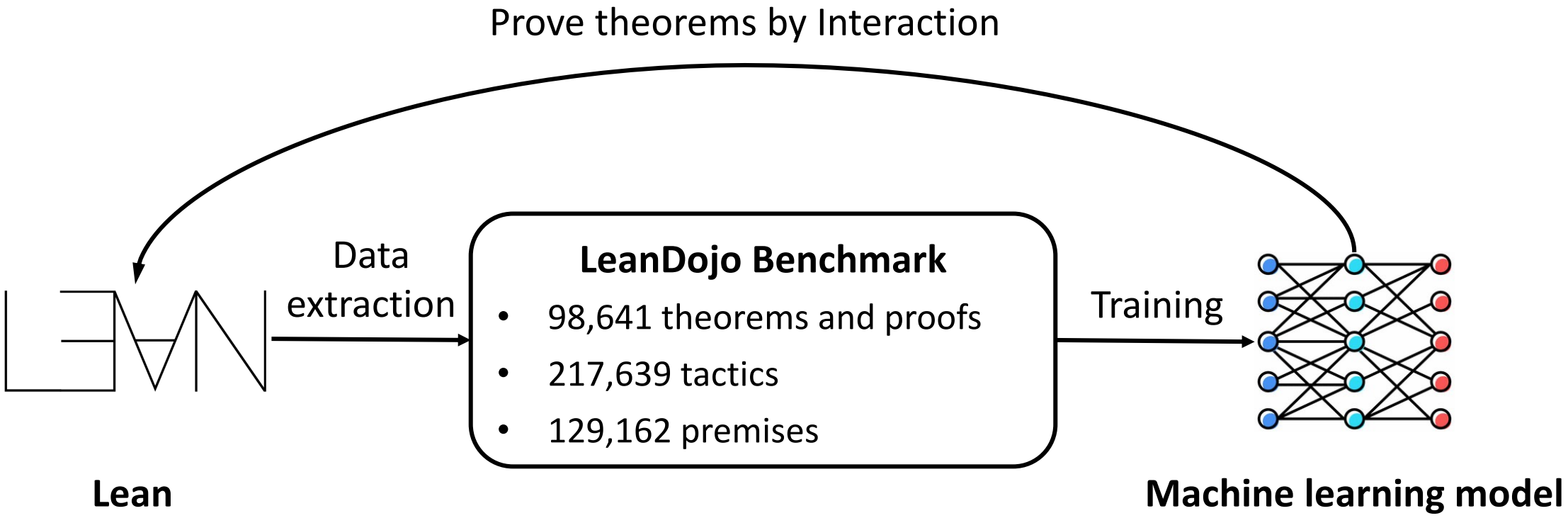
LeanDojo


Data
extraction
LeanDojo Benchmark
• 98,641 theorems and proofs
• 217,639 tactics
• 129,162 premises
Lean
Machine learning model

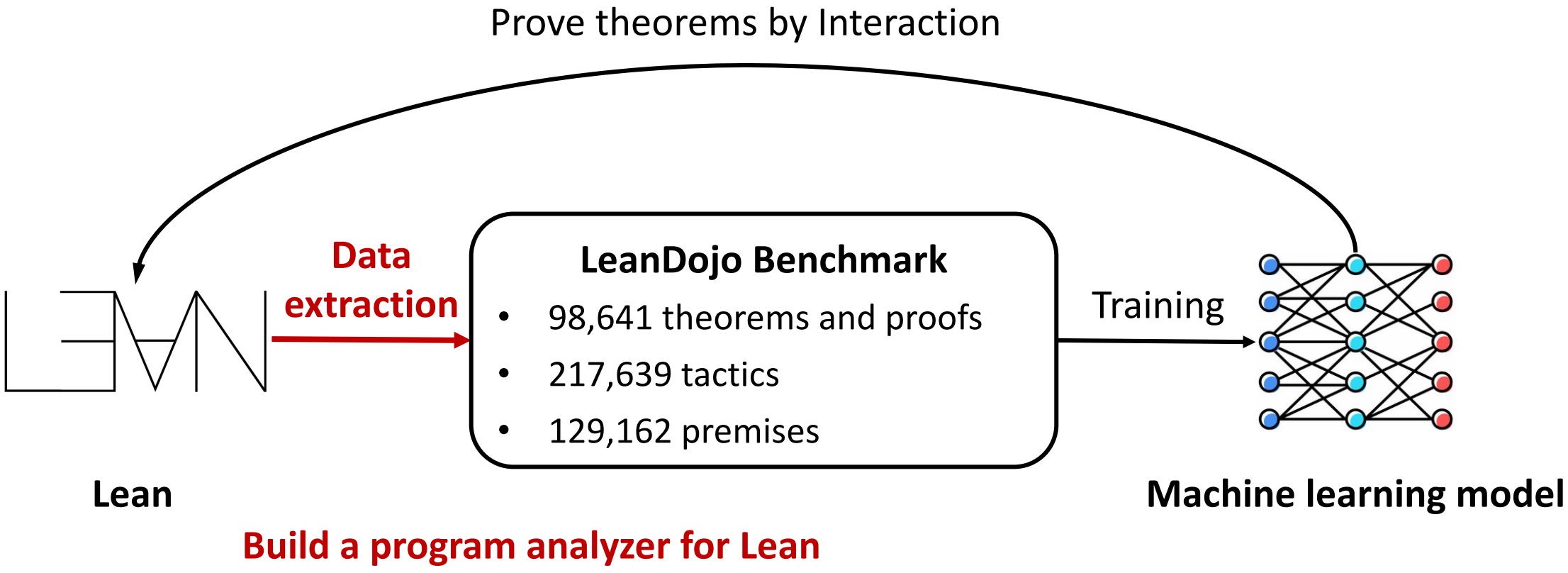
LeanDojo


Data
extraction
LeanDojo Benchmark
• 98,641 theorems and proofs
• 217,639 tactics
• 129,162 premises
Training
Lean
Machine learning model

LeanDojo


Prove theorems by Interaction
Data extraction
LeanDojo Benchmark
• 98,641 theorems and proofs
• 217,639 tactics
• 129,162 premises
Training
Lean
Machine learning model

LeanDojo


Prove theorems by Interaction
Data extraction
LeanDojo Benchmark
• 98,641 theorems and proofs
• 217,639 tactics
• 129,162 premises
Training
Lean
Machine learning model
Build a program analyzer for Lean

Data Extraction from Lean

- **ASTs, tactics**
  - From Lean's parser

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
  cases n,
  { unfold gcd },
  unfold gcd,
  rewrite mod_self,
  apply gcd_zero_left
end
```

Data Extraction from Lean

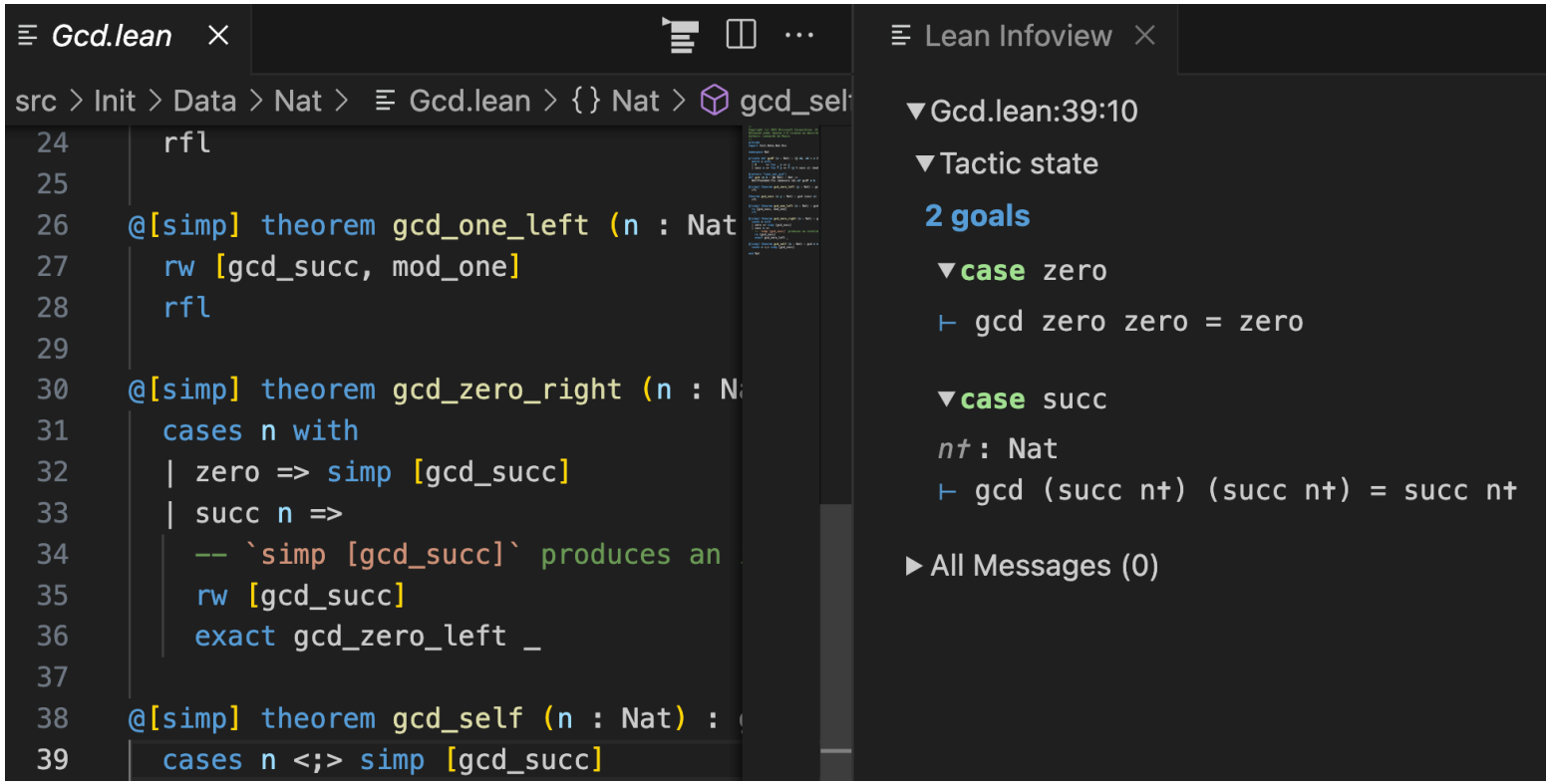
- ASTs, tactics
  - From Lean's parser
- **Proof goals**
  - From Lean's InfoTree

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
  cases n,
  { unfold gcd },
  unfold gcd,
  rewrite mod_self,
  apply gcd_zero_left
end
```

Data Extraction from Lean

- ASTs, tactics
  - From Lean's parser
- **Proof goals**
  - From Lean's InfoTree



Data Extraction from Lean

- ASTs, tactics
  - From Lean's parser
- **Proof goals**
  - From Lean's InfoTree

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
  cases n,
  { unfold gcd },
  unfold gcd,
  rewrite mod_self,
  apply gcd_zero_left
end
```

Data Extraction from Lean

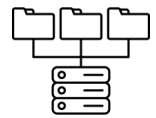
- ASTs, tactics
  - From Lean's parser
- Proof goals
  - From Lean's InfoTree
- **Premises**
  - Definitions, lemmas, etc.
  - Where they are used/defined

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
  cases n,
  { unfold gcd },
  unfold gcd,
  rewrite mod_self,
  apply gcd_zero_left
end
```

Data Extraction from Lean

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



Math library

data/nat/lemmas.lean

```
theorem mod_self (n : nat) : n % n = 0 :=
begin
  rw [mod_eq_sub_mod (le_refl _), nat.sub_self, zero_mod]
end
```

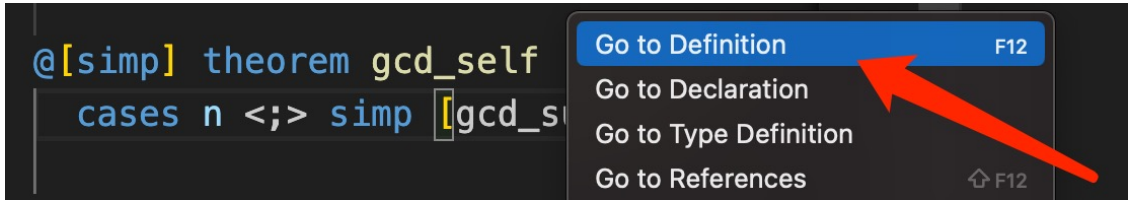
data/nat/gcd.lean

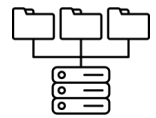
```
def gcd : nat → nat → nat                    -- gcd z y
| 0       y := y                             -- Case 1: z == 0
| (x + 1) y := gcd (y % (x + 1)) (x + 1)     -- Case 2: z > 0

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

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

Import

Data Extraction from Lean

- 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





Math library

data/nat/lemmas.lean

```
theorem mod_self (n : nat) : n % n = 0 :=
begin
  rw [mod_eq_sub_mod (le_refl _), nat.sub_self, zero_mod]
end
```

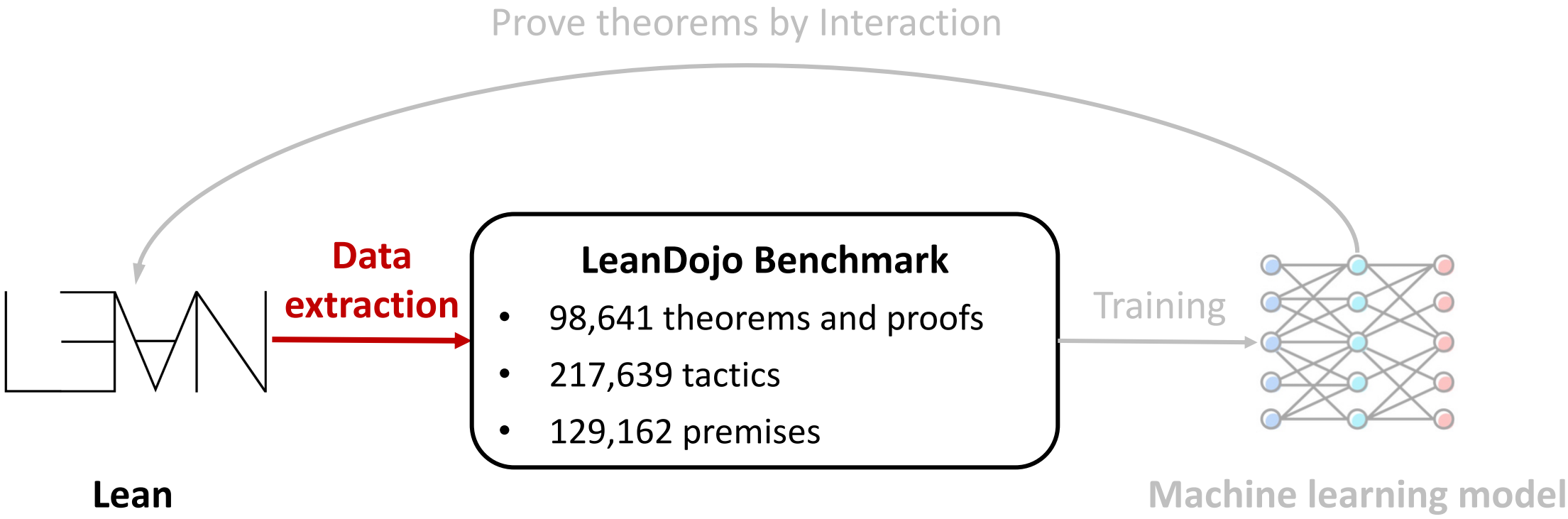
data/nat/gcd.lean

```
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
  cases n,
  { unfold gcd },
  unfold gcd,
  rewrite mod_self,
  apply gcd_zero_left
end
```

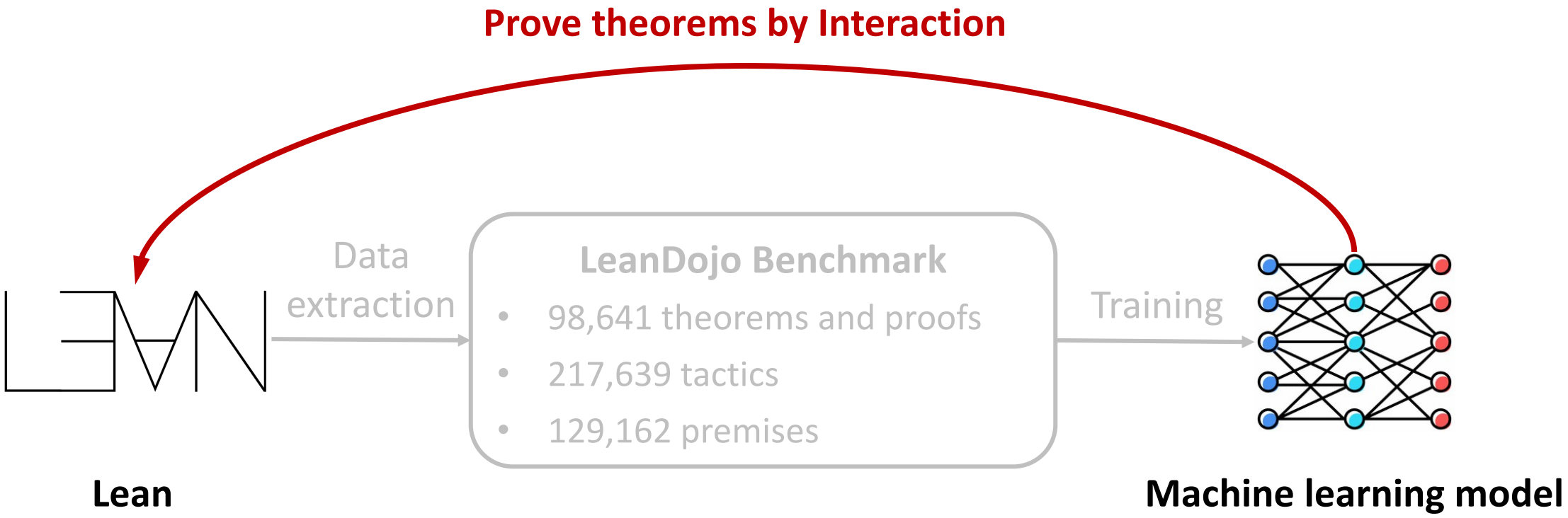
Import

Data Extraction Demo

LeanDojo

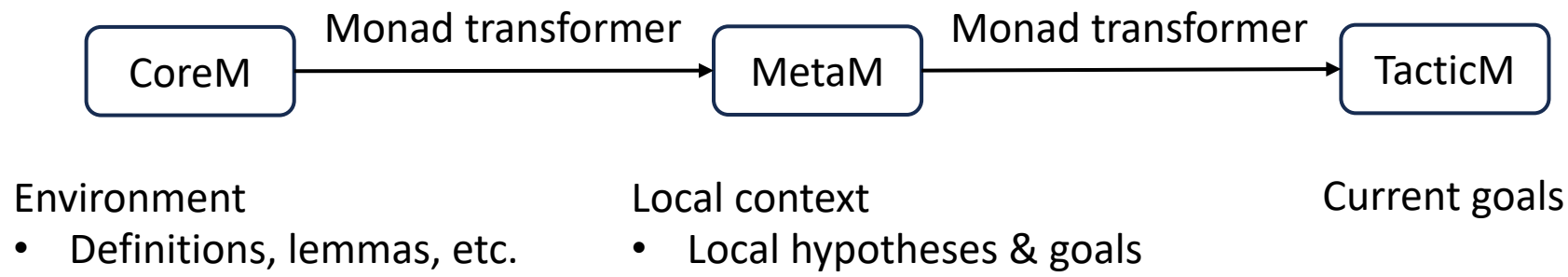

Prove theorems by Interaction
Data extraction
LeanDojo Benchmark
• 98,641 theorems and proofs
• 217,639 tactics
• 129,162 premises
Training
Lean
Machine learning model

LeanDojo


Prove theorems by Interaction
Data extraction
LeanDojo Benchmark
• 98,641 theorems and proofs
• 217,639 tactics
• 129,162 premises
Training
Lean
Machine learning model

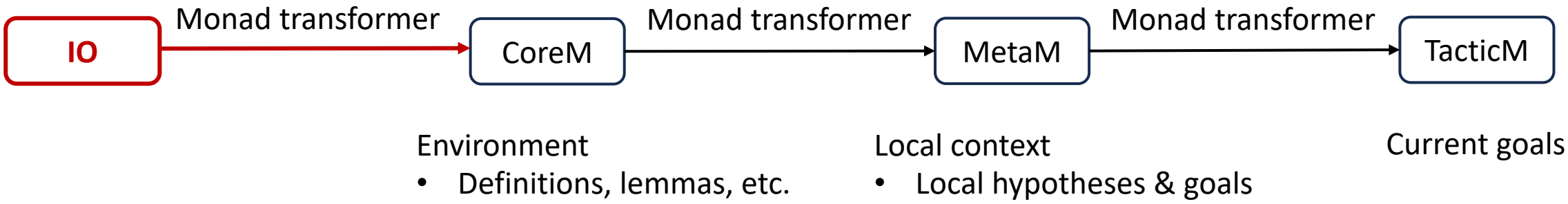
Tactic Monad in Lean

- TacticM can perform IO to interact with the outside world



Tactic Monad in Lean

- TacticM can perform IO to interact with the outside world



Interacting with Lean Programmatically

- Replace the human-written proof with a single repl tactic

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
  cases n,
  { unfold gcd },
  unfold gcd,
  rewrite mod_self,
  apply gcd_zero_left
end
```

Interacting with Lean Programmatically

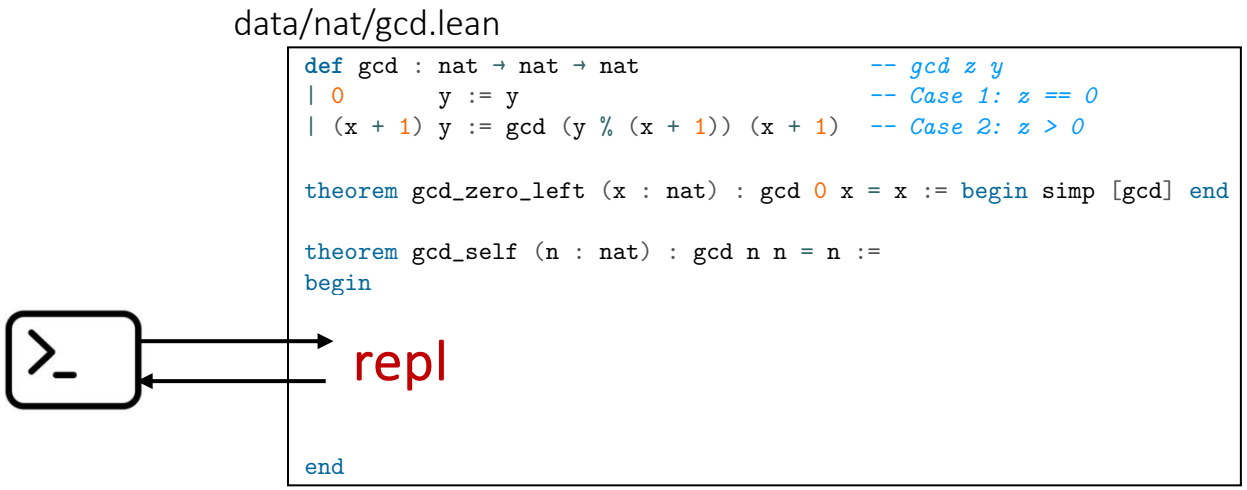
- Replace the human-written proof with a single repl tactic

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
```

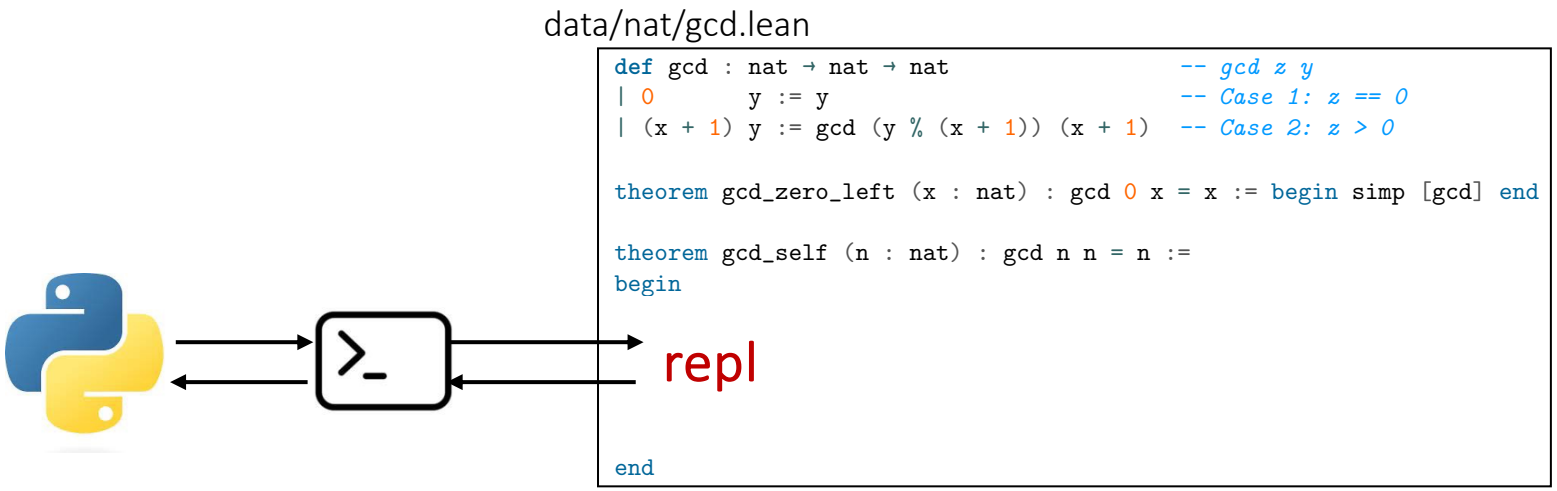
Interacting with Lean Programmatically

- Replace the human-written proof with a single repl tactic
- repl performs IO to provide a command line interface for interacting with Lean



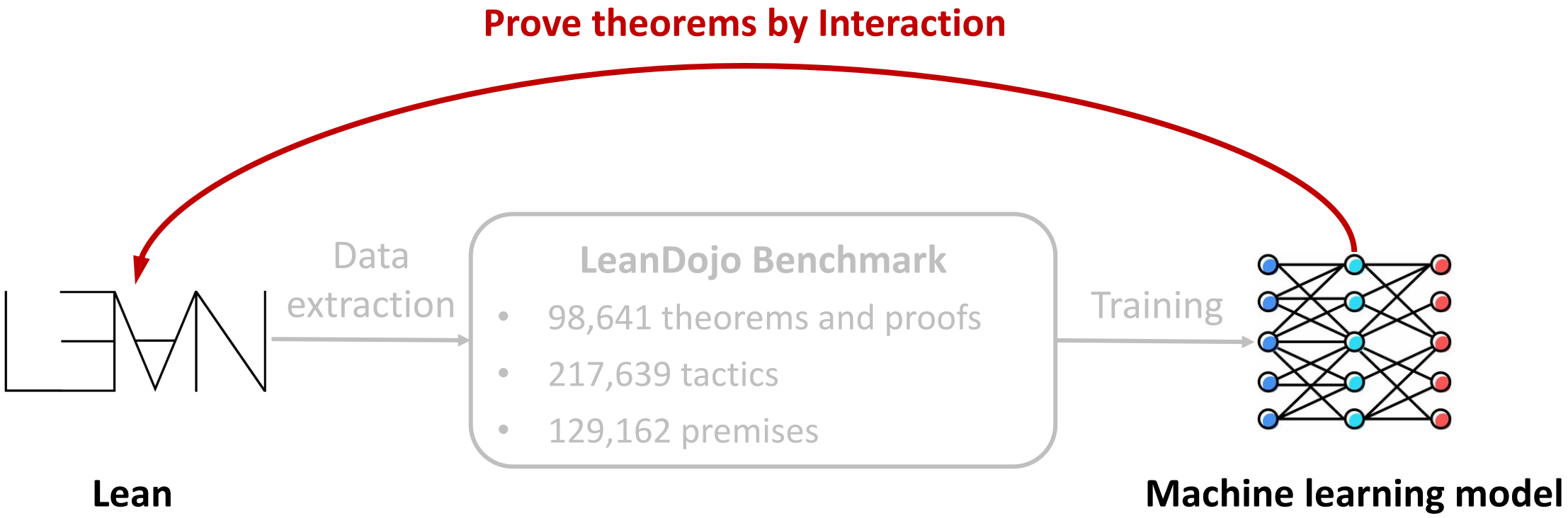
Interacting with Lean Programmatically

- 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

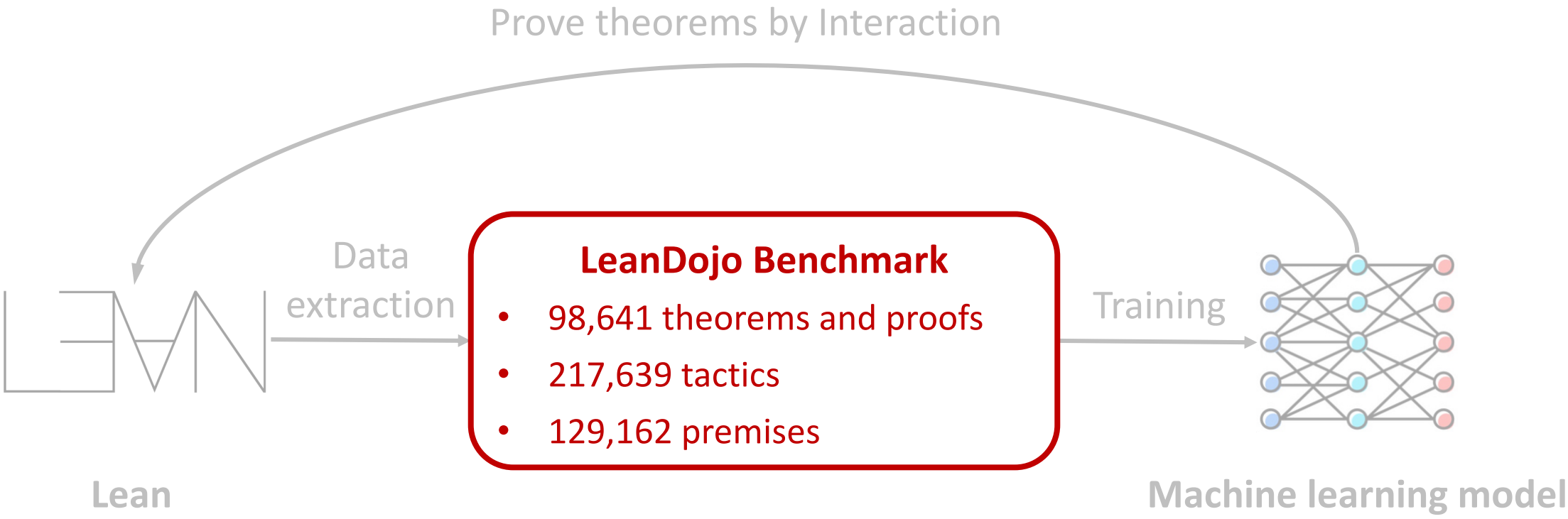


Interaction Demo

LeanDojo

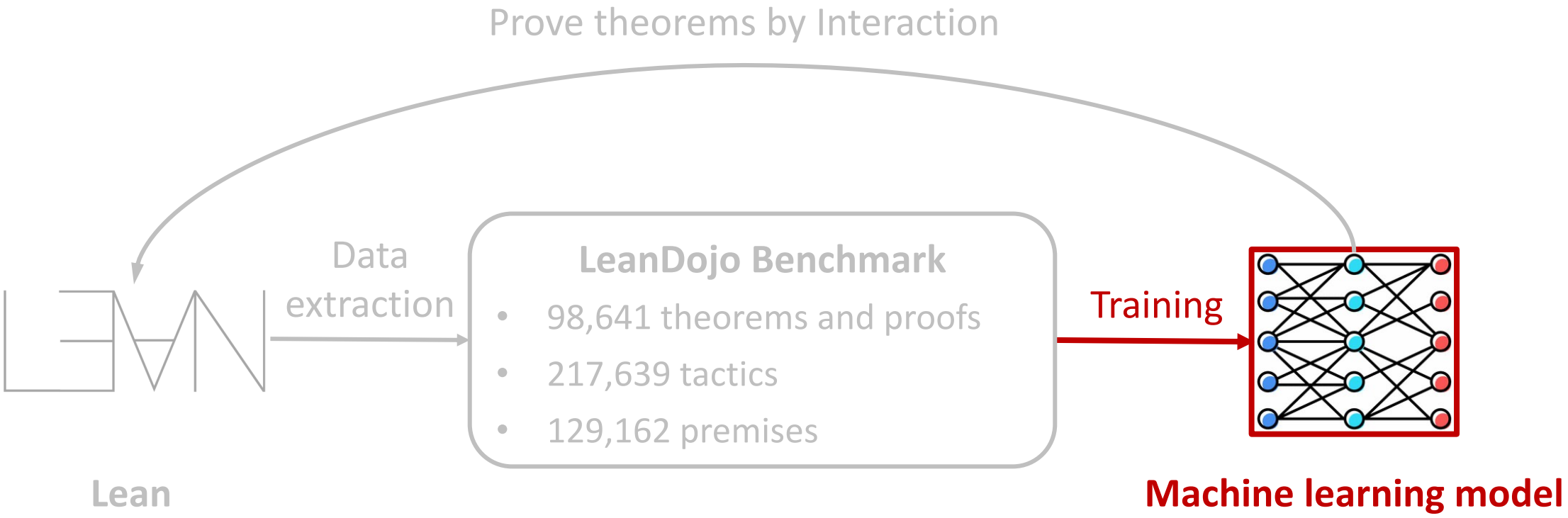

Prove theorems by Interaction
Data extraction
LeanDojo Benchmark
• 98,641 theorems and proofs
• 217,639 tactics
• 129,162 premises
Training
Lean
Machine learning model

LeanDojo



Easy to construct your own benchmarks

LeanDojo

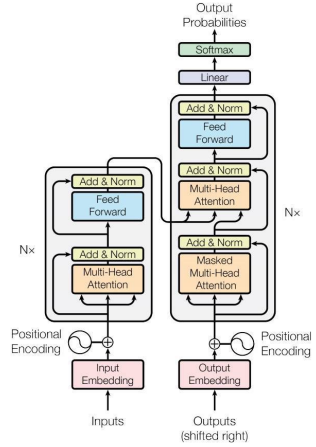

Prove theorems by Interaction
Data extraction
LeanDojo Benchmark
• 98,641 theorems and proofs
• 217,639 tactics
• 129,162 premises
Training
Lean
Machine learning model

Tactic Generation Augmented by Premise Retrieval

- Existing methods can use premises only by memorizing their names

State

```
k : ℕ
⊢ gcd ((k + 1) % (k + 1)) (k + 1) = k + 1
```



→ rewrite mod_self

Tactic

[Vaswani et al., NeurIPS 2017]

Tactic Generation Augmented by Premise Retrieval

- Existing methods can use premises only by memorizing their names
- Given a state, we retrieve premises from the set of **all accessible premises**

State

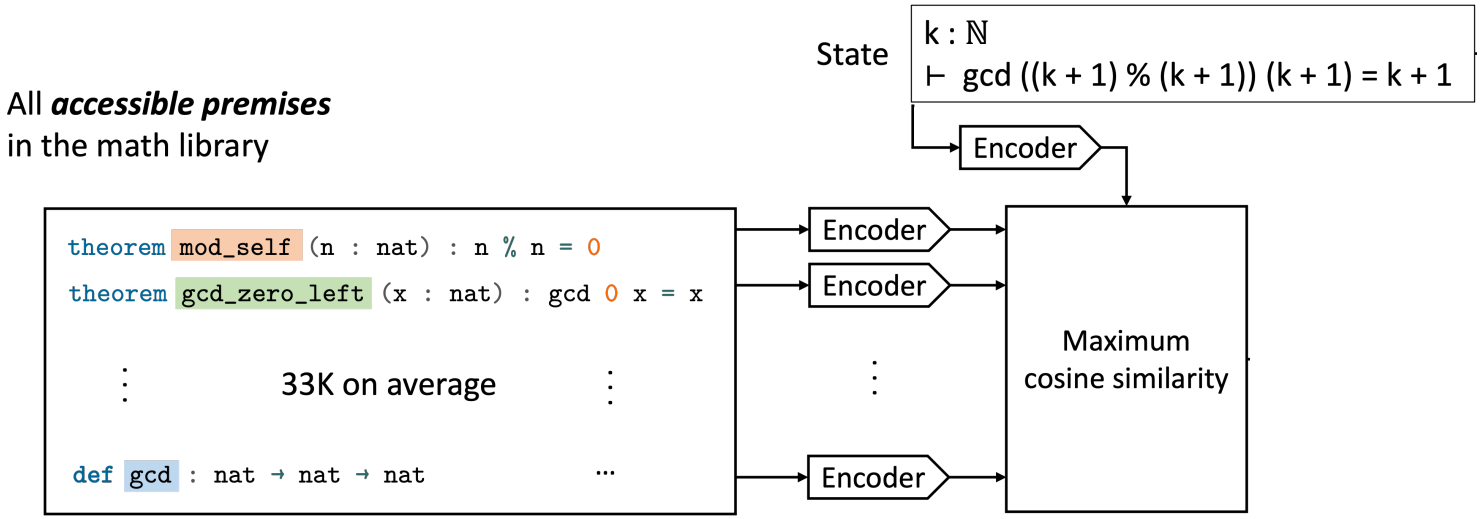
```
k : ℕ
⊢ gcd ((k + 1) % (k + 1)) (k + 1) = k + 1
```

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                          ⋯
```

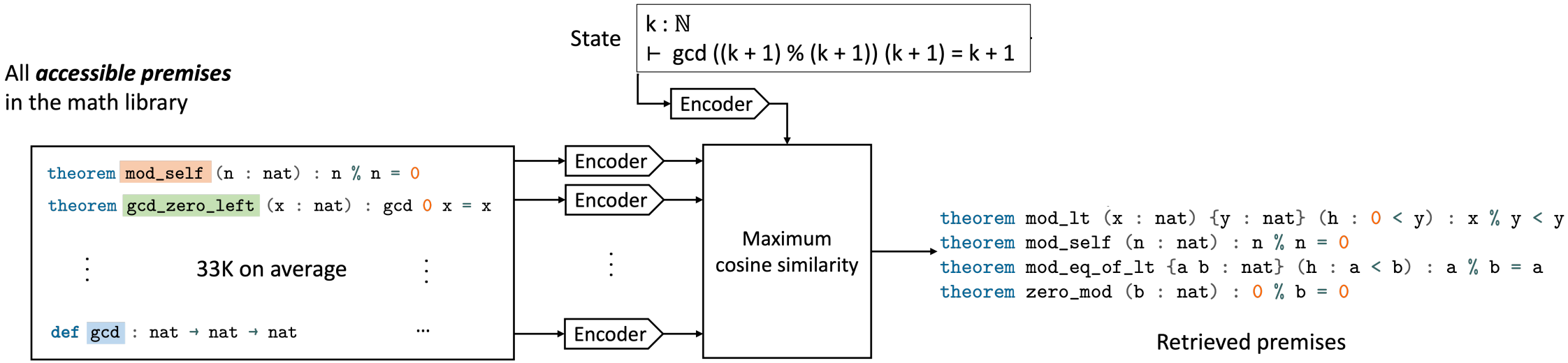
Tactic Generation Augmented by Premise Retrieval

- Existing methods can use premises only by memorizing their names
- Given a state, we retrieve premises from the set of **all accessible premises**



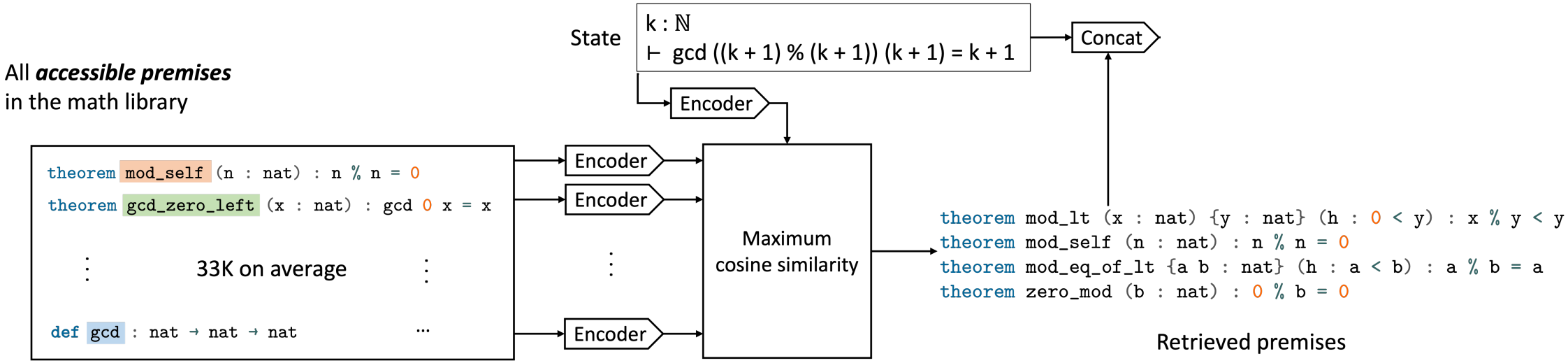
Tactic Generation Augmented by Premise Retrieval

- Existing methods can use premises only by memorizing their names
- Given a state, we retrieve premises from the set of **all accessible premises**



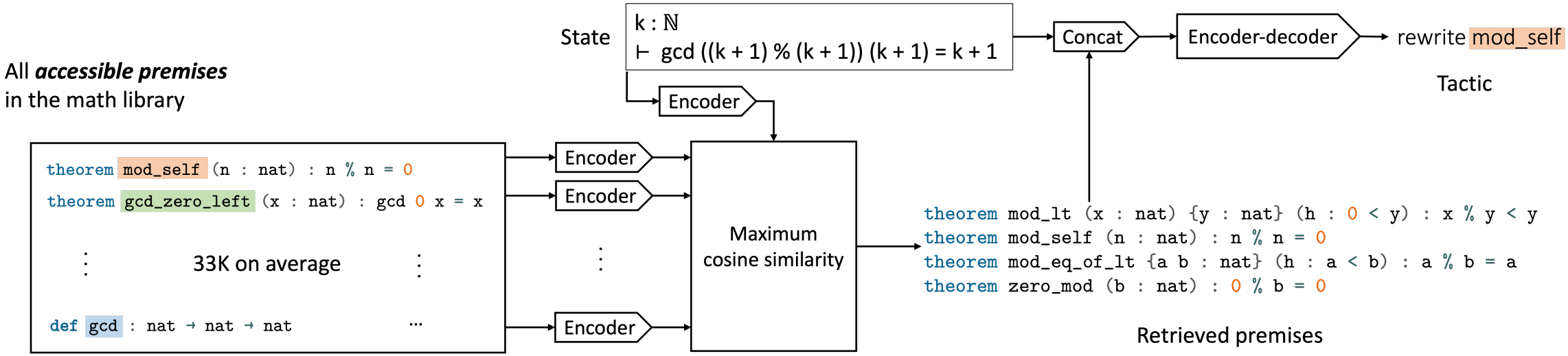
Tactic Generation Augmented by Premise Retrieval

- Existing methods can use premises only by memorizing their names
- Given a state, we retrieve premises from the set of **all accessible premises**
- Retrieved premises are concatenated with the state and used for tactic generation

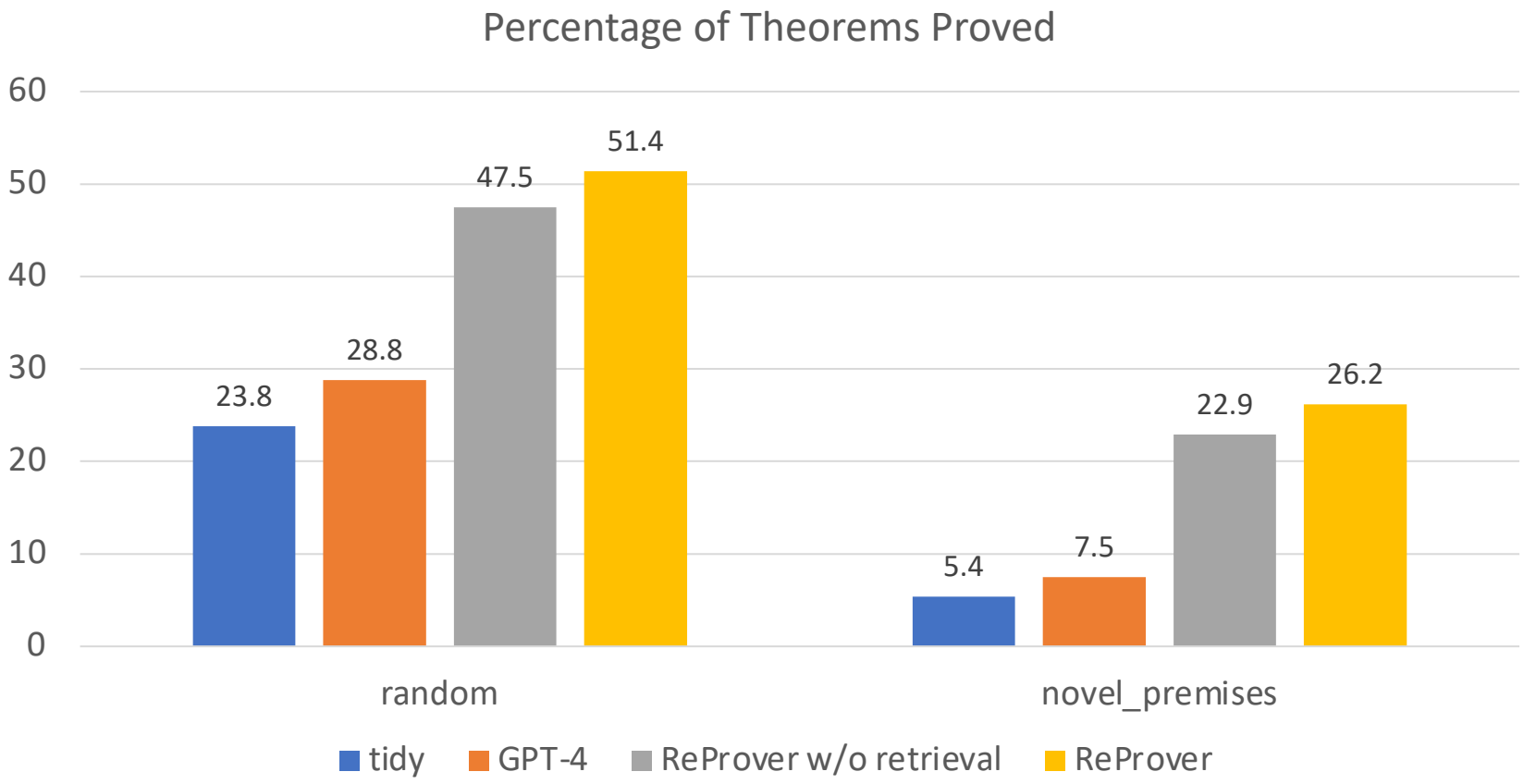


Tactic Generation Augmented by Premise Retrieval

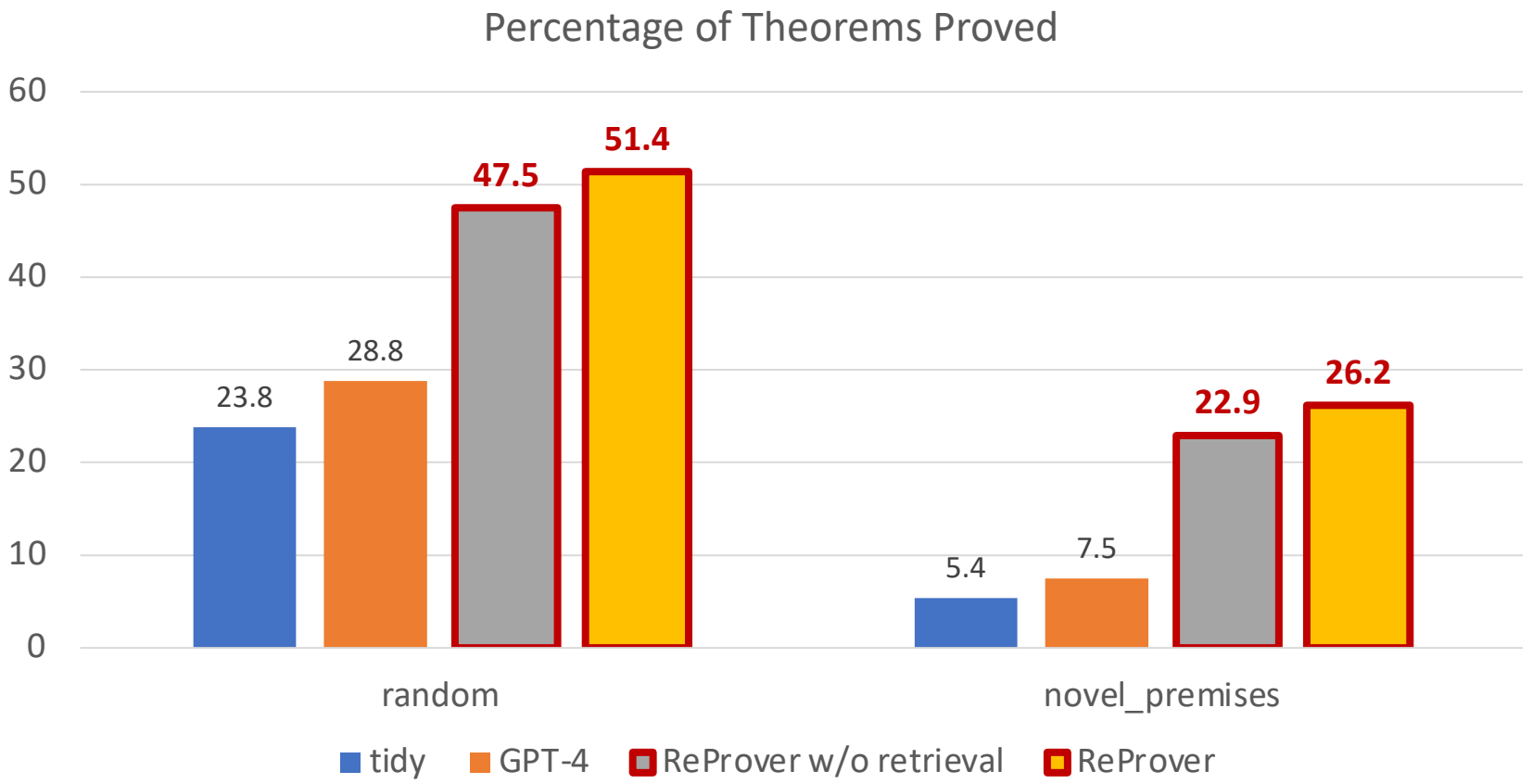
- Existing methods can use premises only by memorizing their names
- Given a state, we retrieve premises from the set of **all accessible premises**
- Retrieved premises are concatenated with the state and used for tactic generation



Premise Retrieval Improves Theorem Proving



Premise Retrieval Improves Theorem Proving



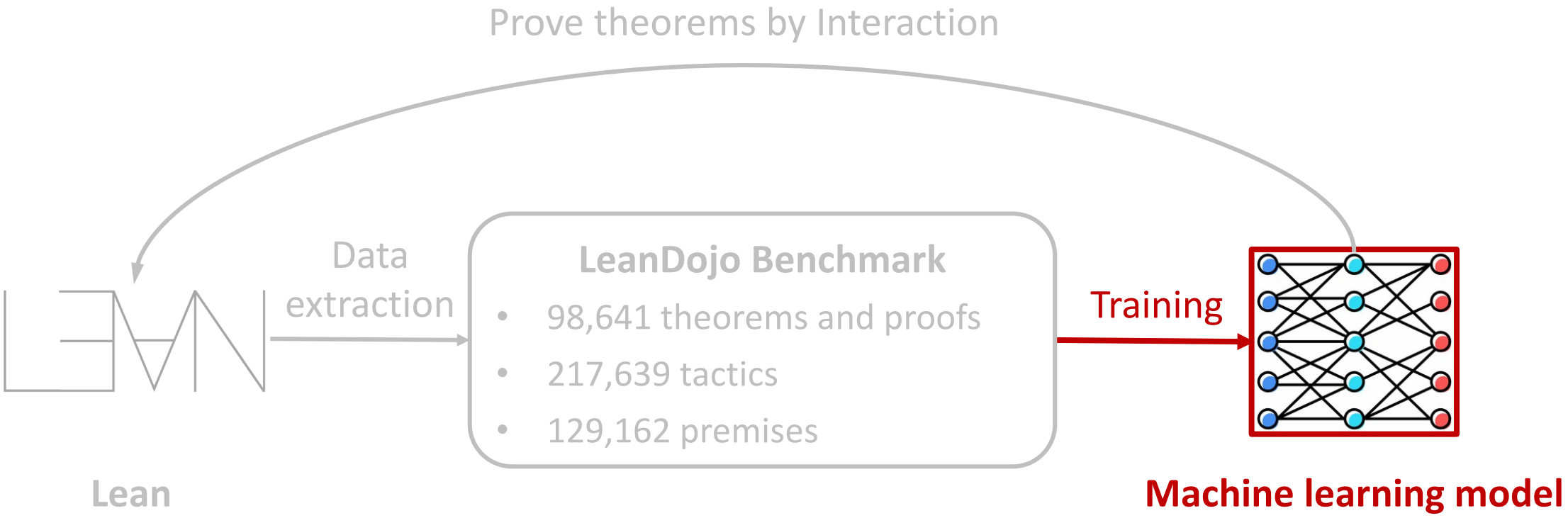
Discovering New Proofs on MiniF2F and ProofNet

- We evaluate the model on MiniF2F and ProofNet (in zero shot) to discover new Lean proofs

```
theorem exercise_2_3_2 {G : Type*} [group G] (a b : G) :
    g : G, b * a = g * a * b * g ⁻¹ :=
begin
  exact  b, by simp ,
end

theorem exercise_11_2_13 (a b :  ) :
  (of_int a : gaussian_int)   of_int b → a   b :=
begin
  contrapose,
  simp,
end

theorem exercise_1_1_17 {G : Type*} [group G] {x : G} {n :  }
  (hxn: order_of x = n) :
  x ⁻¹ = x ^ (n - 1 :  ) :=
begin
  rw zpow_sub_one,
  simp,
  rw [← hxn, pow_order_of_eq_one],
end
```

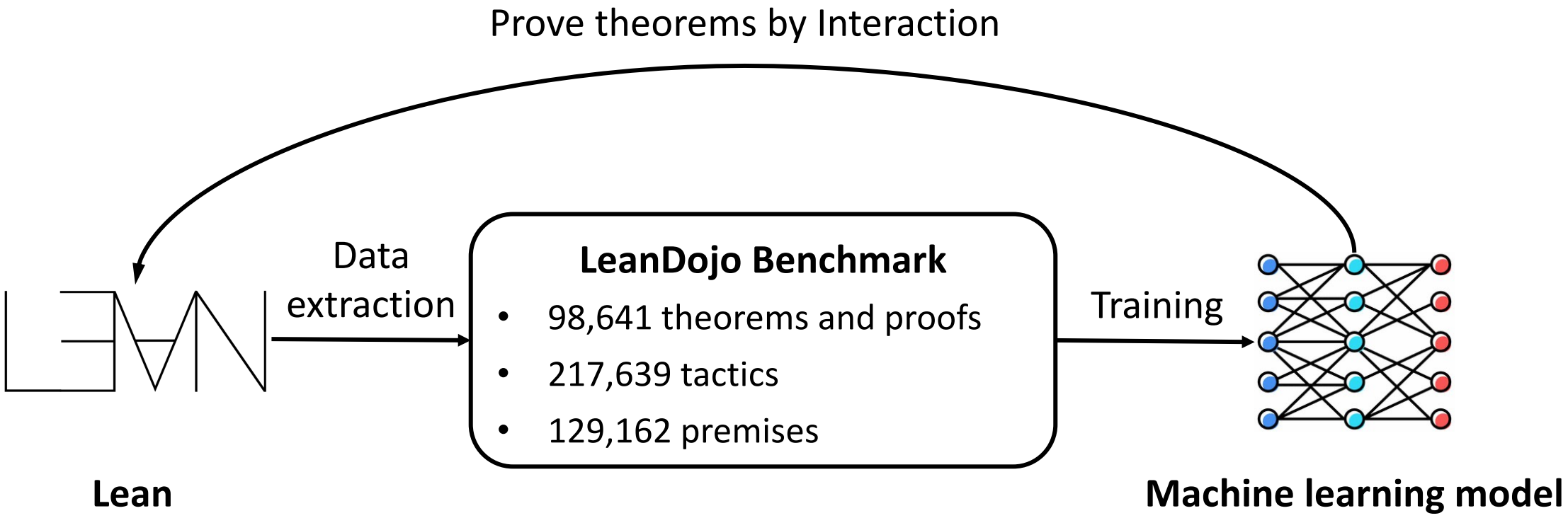
```
theorem exercise_3_1_22b {G : Type*} [group G] (I : Type*)
  (H : I → subgroup G) (hH :   i : I, subgroup.normal (H i)) :
  subgroup.normal (  (i : I), H i):=
begin
  rw infi,
  rw ←set.image_univ,
  rw Inf_image,
  simp [hH],
  haveI :=  i, (H i).normal,
  split,
  intros x hx g,
  rw subgroup.mem_infi at hx  ,
  intro i,
  apply (hH i).conj_mem _ (hx i),
end

theorem exercise_3_4_5a {G : Type*} [group G]
  (H : subgroup G) [is_solvable G] : is_solvable H :=
begin
  apply_instance,
end
```

LeanDojo

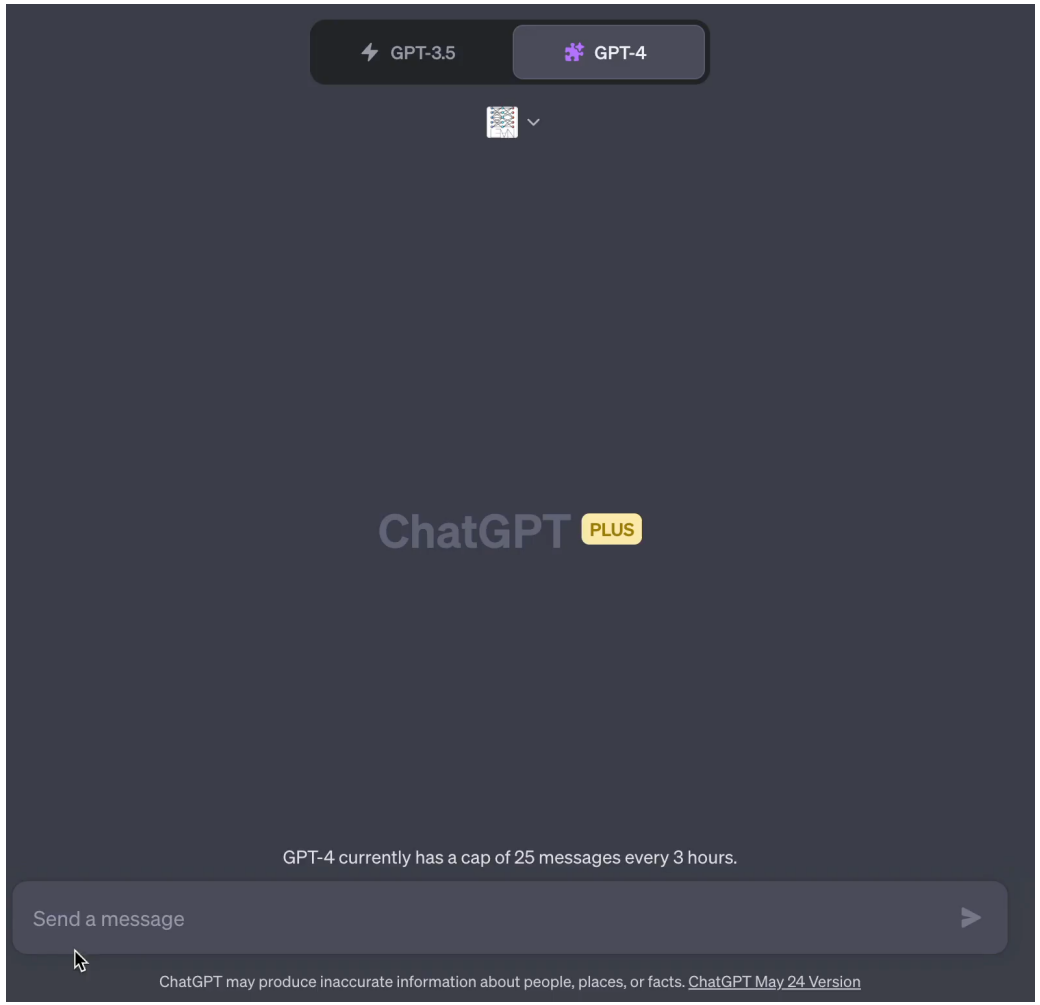

Prove theorems by Interaction
Data extraction
LeanDojo Benchmark
• 98,641 theorems and proofs
• 217,639 tactics
• 129,162 premises
Training
Lean
Machine learning model

LeanDojo

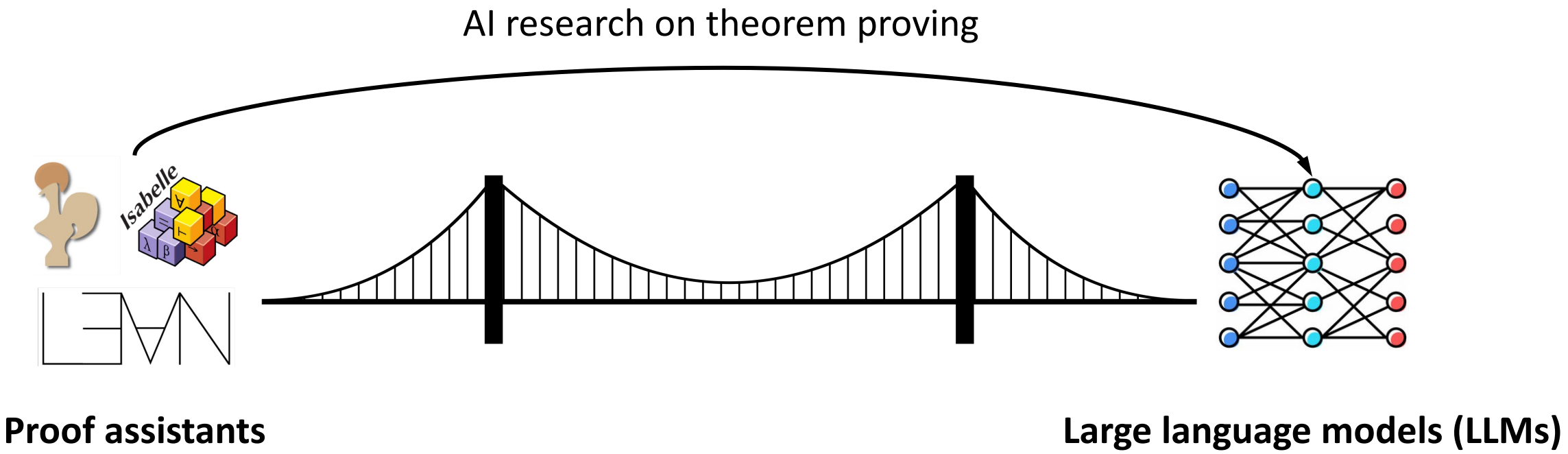
- https://leandojo.org
- Tutorial @ NeurIPS 2023



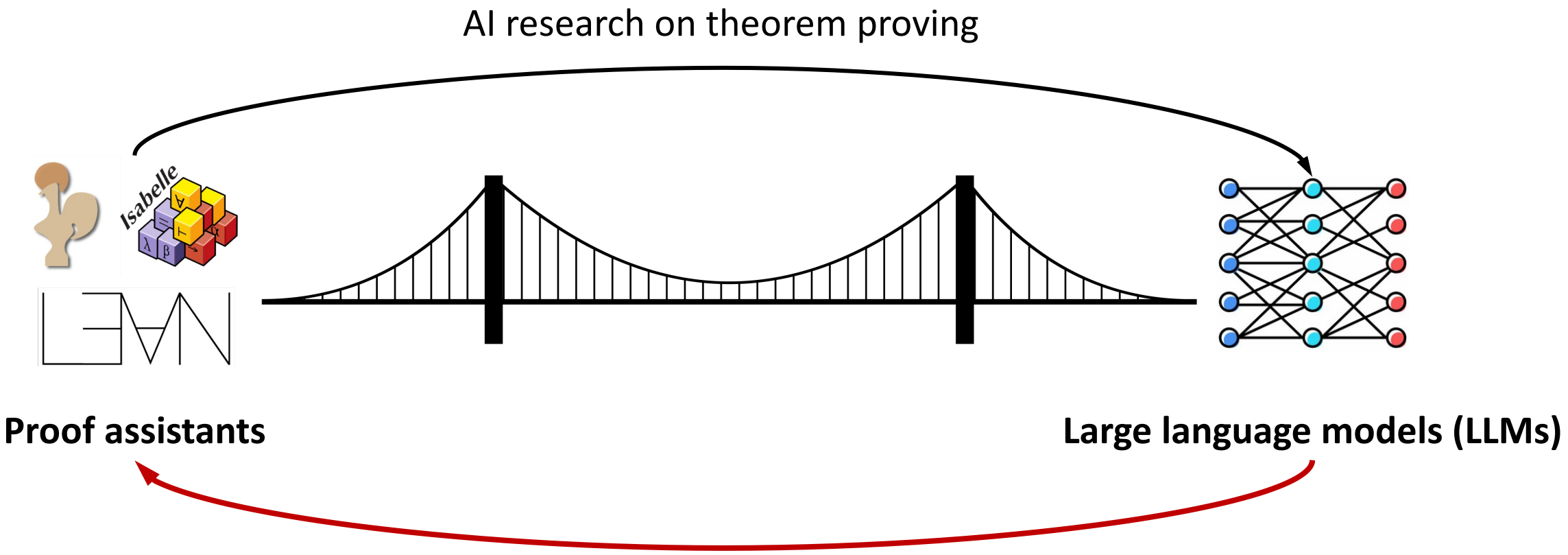
ChatGPT for Theorem Proving in Lean



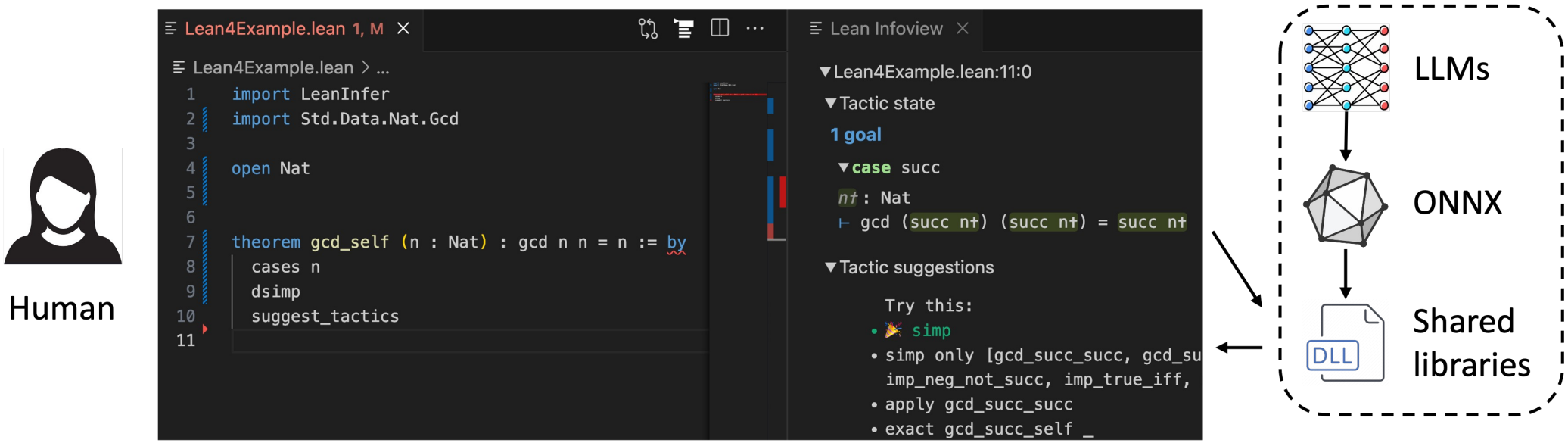
Practical LLM-Based Proof Automation Tools



Practical LLM-Based Proof Automation Tools



LLMs for Generating Tactic Suggestions in Lean



[Song et al., "Towards Large Language Models as Copilots for Theorem Proving in Lean", In submission, 2023]

https://github.com/lean-dojo/LeanInfer

Easy to install just like any Lean package

Run locally on most laptops w/o GPUs

Searching for Proofs with LLM-aesop

- Aesop: best-first search in Lean [Limperg and From, "Aesop: White-Box Best-First Proof Search for Lean", 2023]
  - Users can configure the rules used for expanding nodes in the search tree
  - The rules are fixed during search and independent of proof goals
- LLM-aesop: Aesop + tactic suggestions generated by LLMs
- Evaluate on 50 theorems selected in the “Mathematics in Lean” book
  - Autonomous setting: Apply Aesop or LLM-aesop directly to the theorem
  - Human-assistive setting: Humans enter a number of tactics before calling the tool
  - LLM-aesop outperforms Aesop in both settings

Method	Avg. # human-entered tactics (↓)	% Theorems proved autonomously (↑)
`aesop`	3.62	12%
`suggest_tactics`	2.72	34%
LLM-aesop	**1.02**	**64%**

Searching for Proofs with LLM-aesop

- Aesop: best-first search in Lean [Limperg and From, "Aesop: White-Box Best-First Proof Search for Lean", 2023]
  - Users can configure the rules used for expanding nodes in the search tree
  - The rules are fixed during search and independent of proof goals
- LLM-aesop: Aesop + tactic suggestions generated by LLMs

Team

Aidan Swope

Alex Gu

Rahul Chalamala

Peiyang Song

Shixing Yu

Saad Godil

Ryan Prenger

Anima Anandkumar

LeanDojo

- https://leandojo.org
- Tutorial @ NeurIPS 2023