LeanDojo: Theorem Proving with Retrieval-Augmented Language Models

Kaiyu Yang

Postdoc @ Computing + Mathematical Sciences

Caltech

Teaser: LLMs as Copilots for Theorem Proving

Formal 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⟩
```

Formal Theorem Proving

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

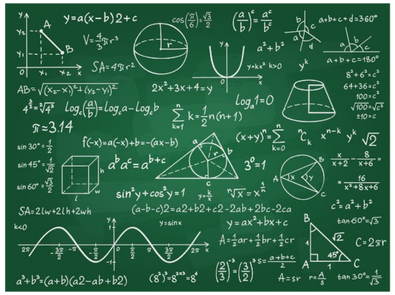
Theorem

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

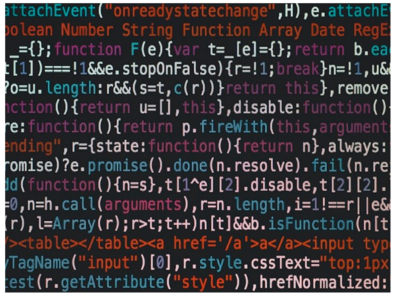
Proof

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

Formalize



Mathematics



Software

- Theorems/proofs represented formally as programs

Formal Theorem Proving

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

Theorem

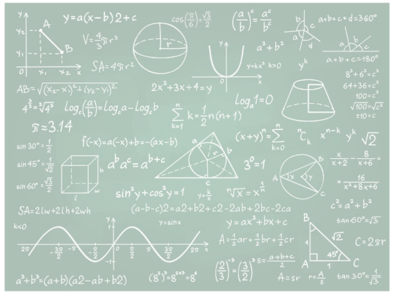
```
theorem set_inter_comm (s t : Set α) : s ∩ t = t ∩ s
```

Proof

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

✓

Formalize



Mathematics

Software

- Theorems/proofs represented formally as programs
- Proofs can be checked easily. No room for hallucination

Formal Theorem Proving

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

Theorem

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

Proof

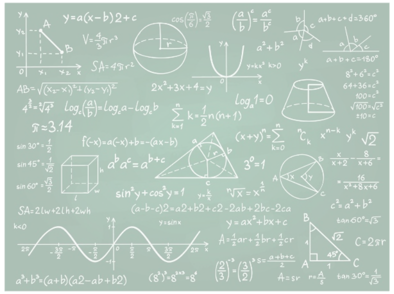
```
ext x
simp [Set.mem_inter_iff]
constructor
· rintro ⟨xs, xt⟩
  exact ⟨xt, xs⟩
· rintro ⟨xt, xs⟩
  exact ⟨xs, xt⟩
```

✓

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

✗

Formalize



Mathematics

Software

- Theorems/proofs represented formally as programs
- Proofs can be checked easily. No room for hallucination

How to Prove Theorems (with Machine Learning)?

Proof Assistants (Interactive Theorem Provers)

Theorem

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

Proof

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

Proof Assistants (Interactive Theorem Provers)

Humans  + Proof assistants

IDEs for writing formal proofs

Theorem

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

Proof

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

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

⊢ ∀ (a b c : 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**

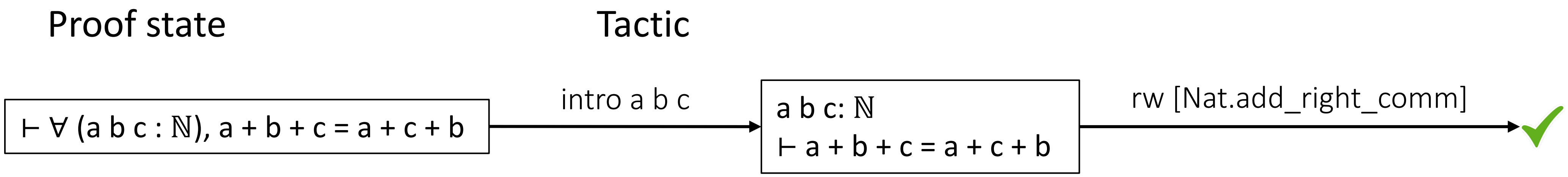
$\vdash \forall (a\ b\ c : \mathbb{N}),\ a + b + c = a + c + b$

→ intro a b c →

$a\ b\ c : \mathbb{N}$
$\vdash a + b + c = a + c + b$

Generating Proof Steps (Tactics)

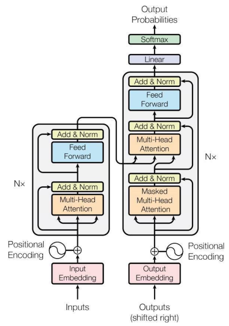
```
theorem add_abc : ∀ a b c : ℕ, a + b + c = a + c + b := by
  intro a b c
  rw [Nat.add_right_comm]
```



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

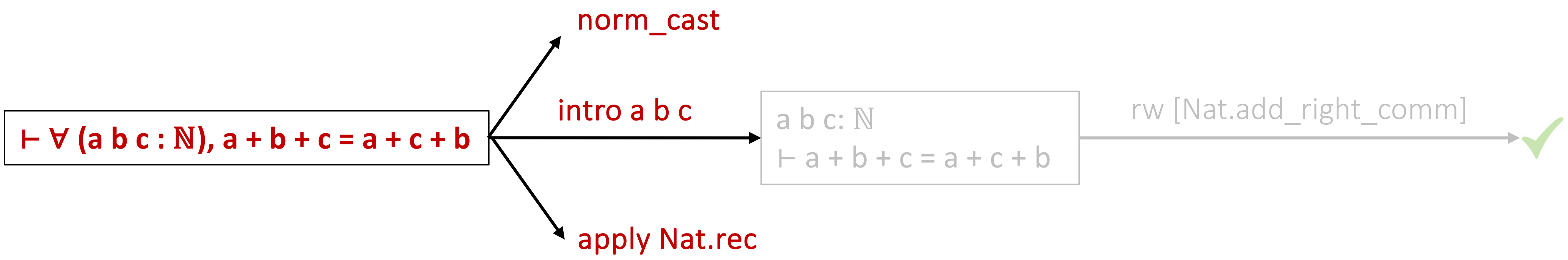
a b c: ℕ
⊢ a + b + c = a + c + b

rw [Nat.add_right_comm]

✓

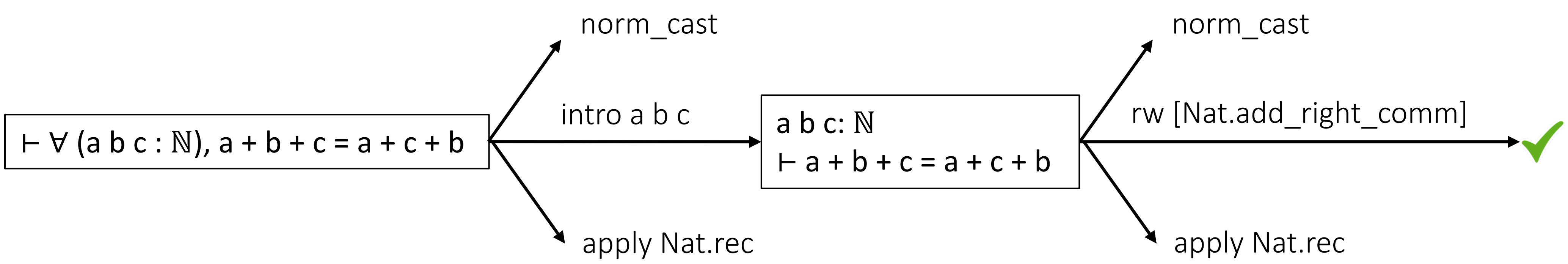
Generating Proof Steps (Tactics)

```
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]
```

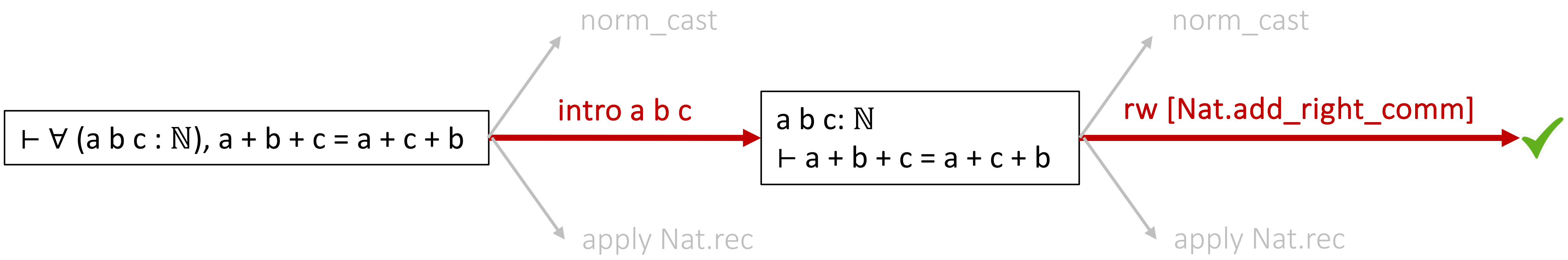


Searching for Proofs

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

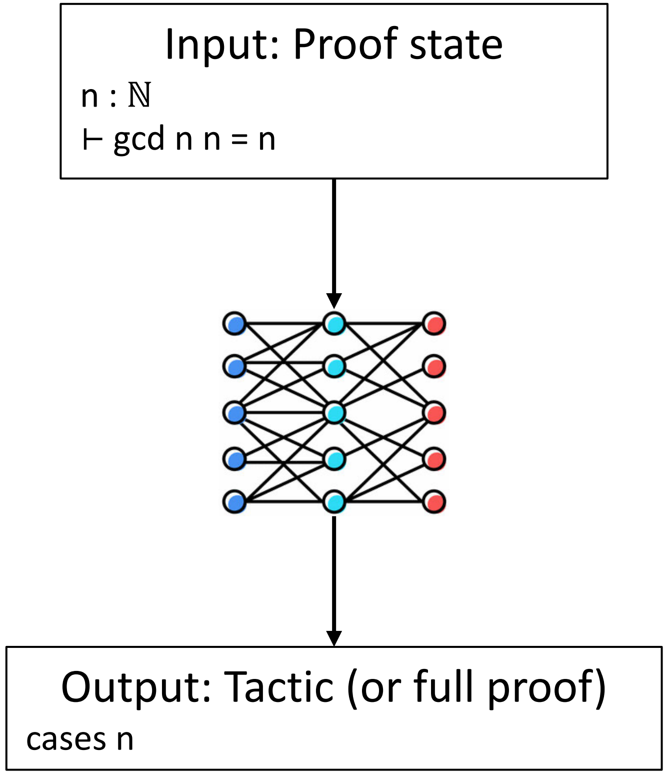
Classical proof search algorithms

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



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



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

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]

LeanDojo

Lean

LeanDojo

Data extraction

LeanDojo Benchmark

- 98,734 theorems and proofs
- 217,776 tactics
- 129,243 premises

Lean

LeanDojo

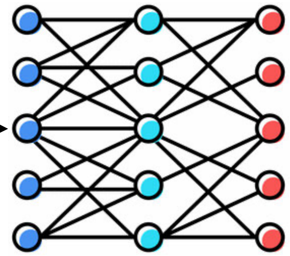
Lean

Data extraction

LeanDojo Benchmark

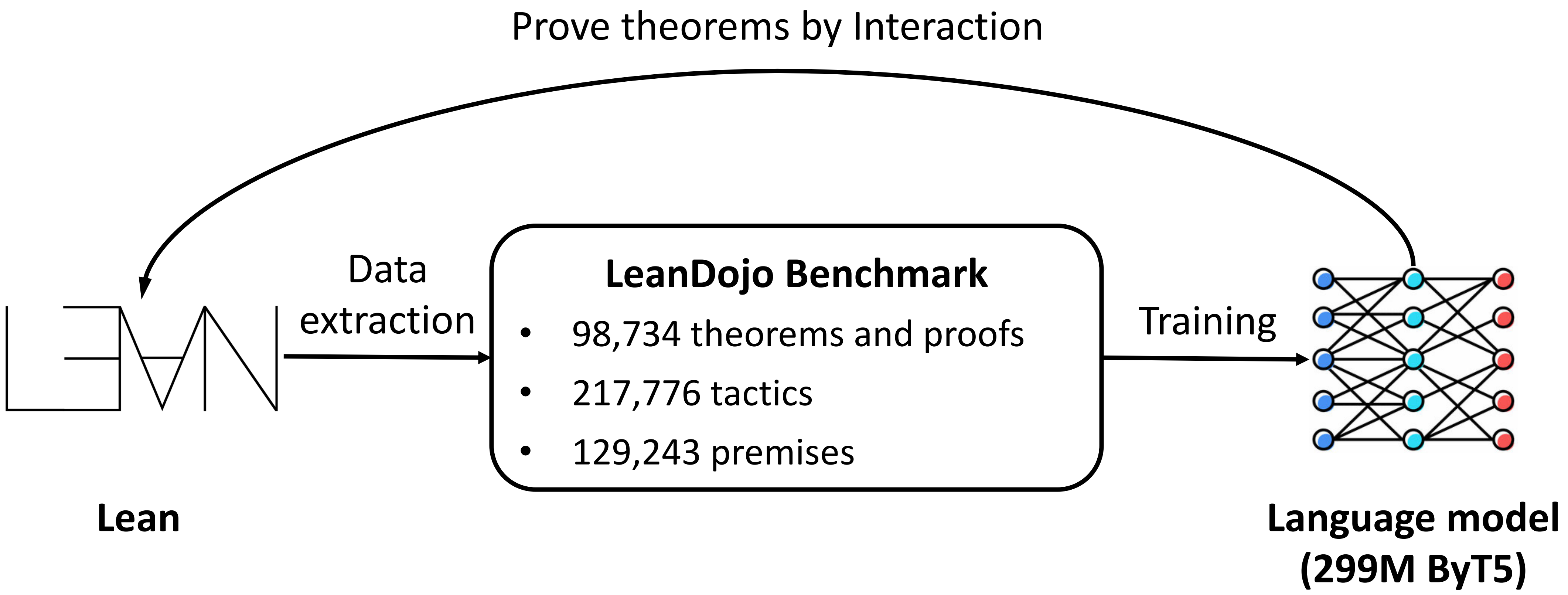
- 98,734 theorems and proofs
- 217,776 tactics
- 129,243 premises

Training

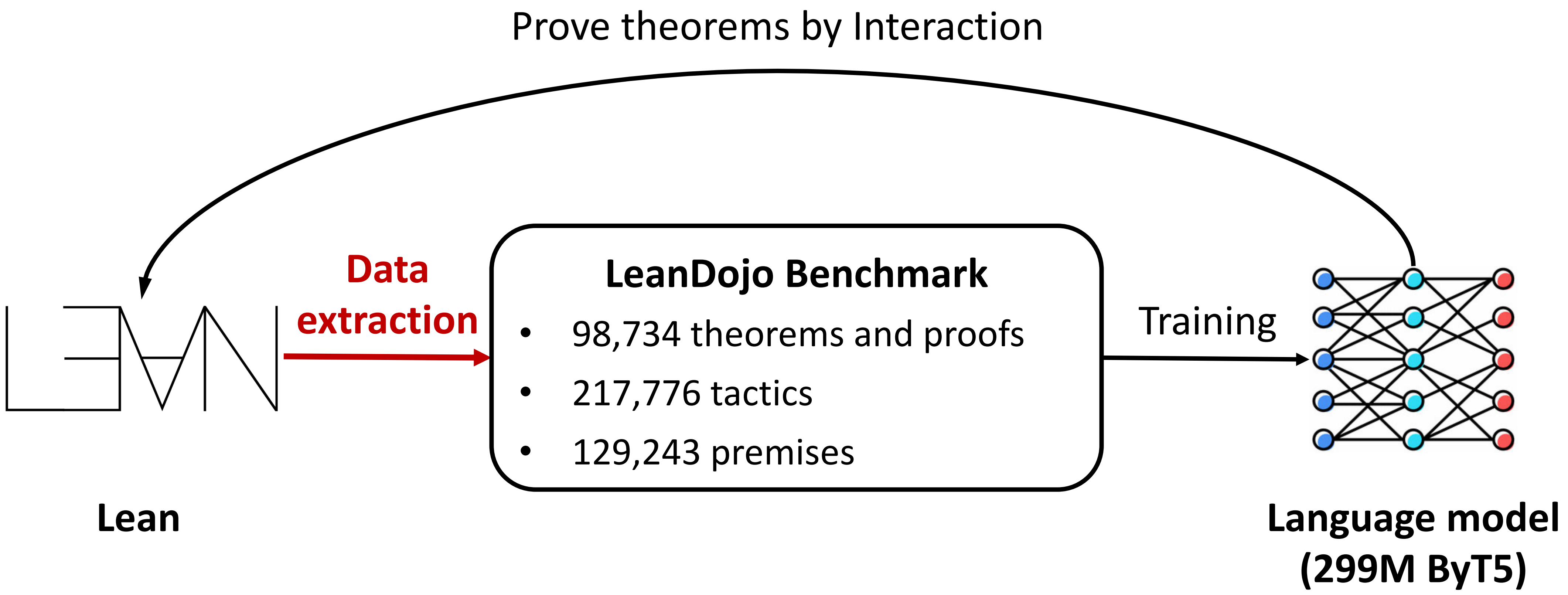


Language model (299M ByT5)

LeanDojo


Prove theorems by Interaction
Data extraction
LeanDojo Benchmark
• 98,734 theorems and proofs
• 217,776 tactics
• 129,243 premises
Training
Lean
Language model (299M ByT5)

LeanDojo

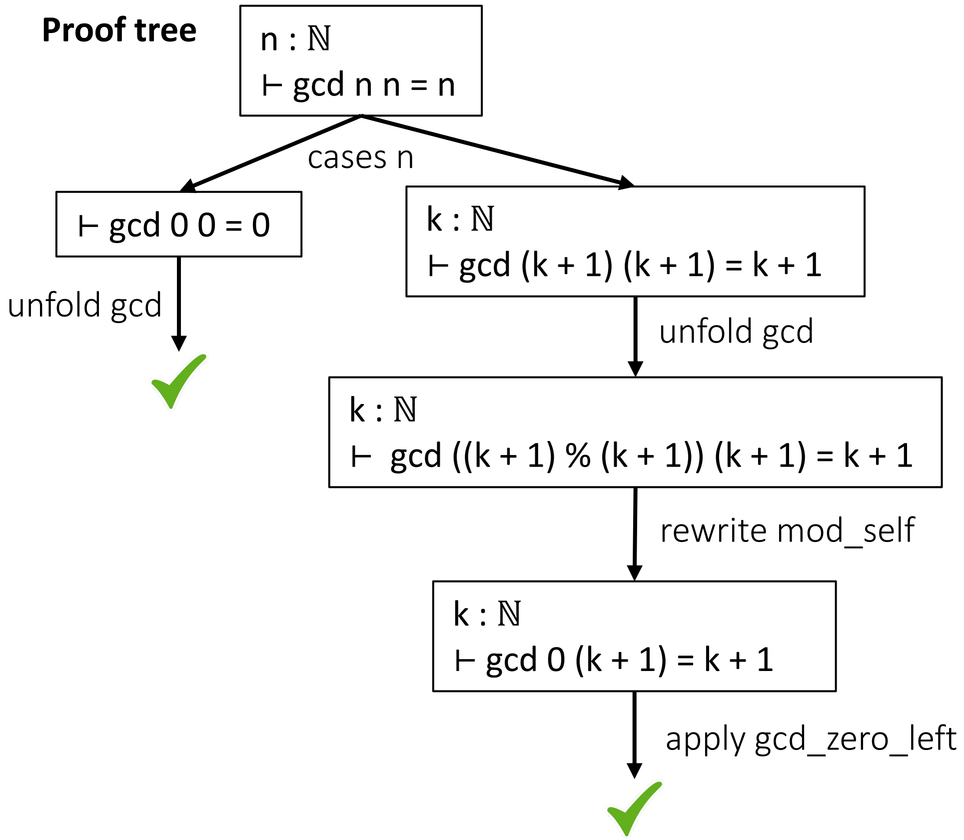

Prove theorems by Interaction
Data extraction
LeanDojo Benchmark
• 98,734 theorems and proofs
• 217,776 tactics
• 129,243 premises
Training
Lean
Language model
(299M ByT5)

Extracting States and Tactics

- Need **(state, tactic)** pairs for training
  - Tactics could be obtained by parsing the Lean source code into ASTs
  - Proof states are not available in the code

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

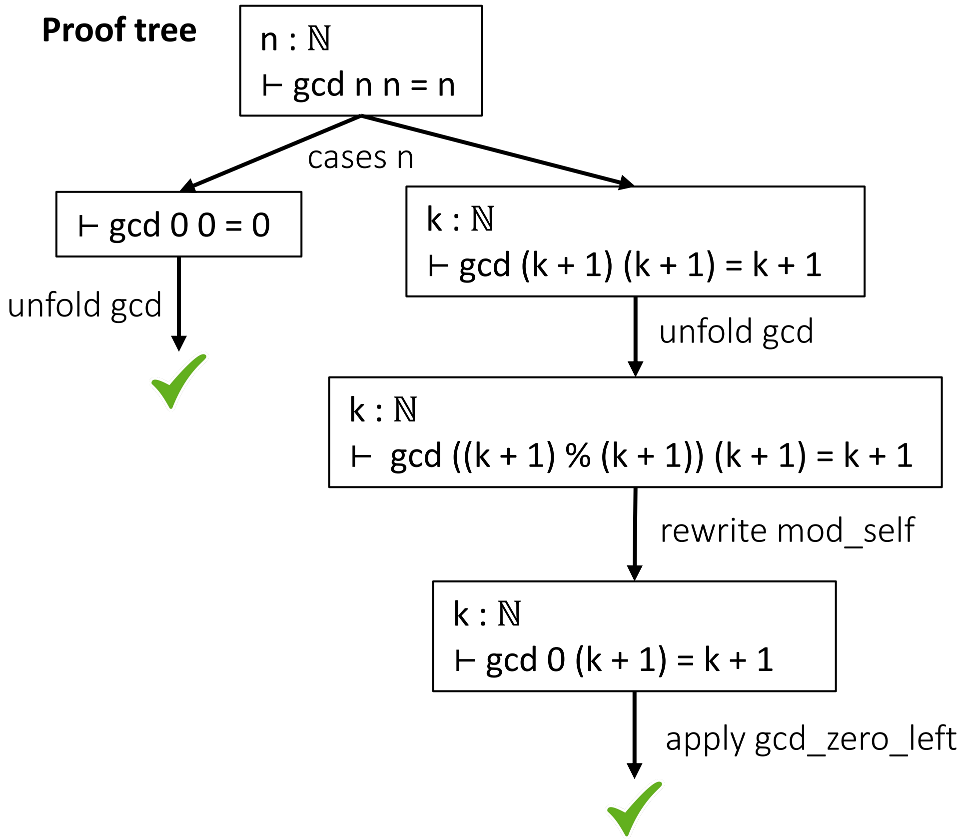
Proof tree



Extracting Premises

- Tactics rely on **premises**
  - Lemmas
  - Definitions

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

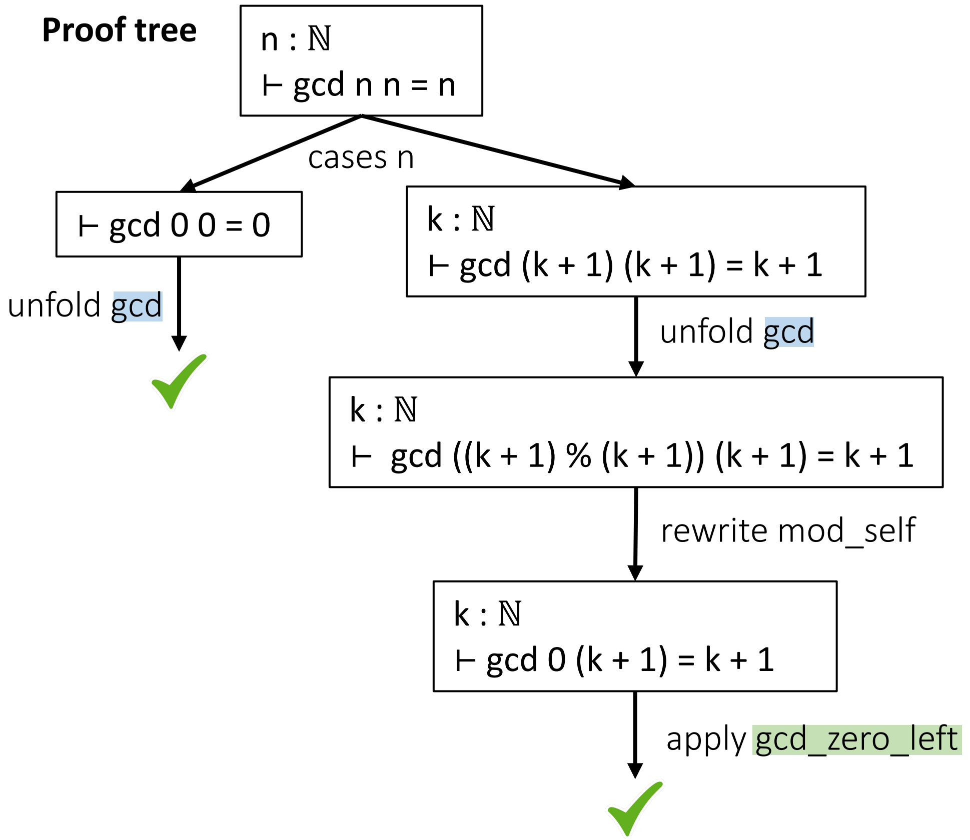


Extracting Premises

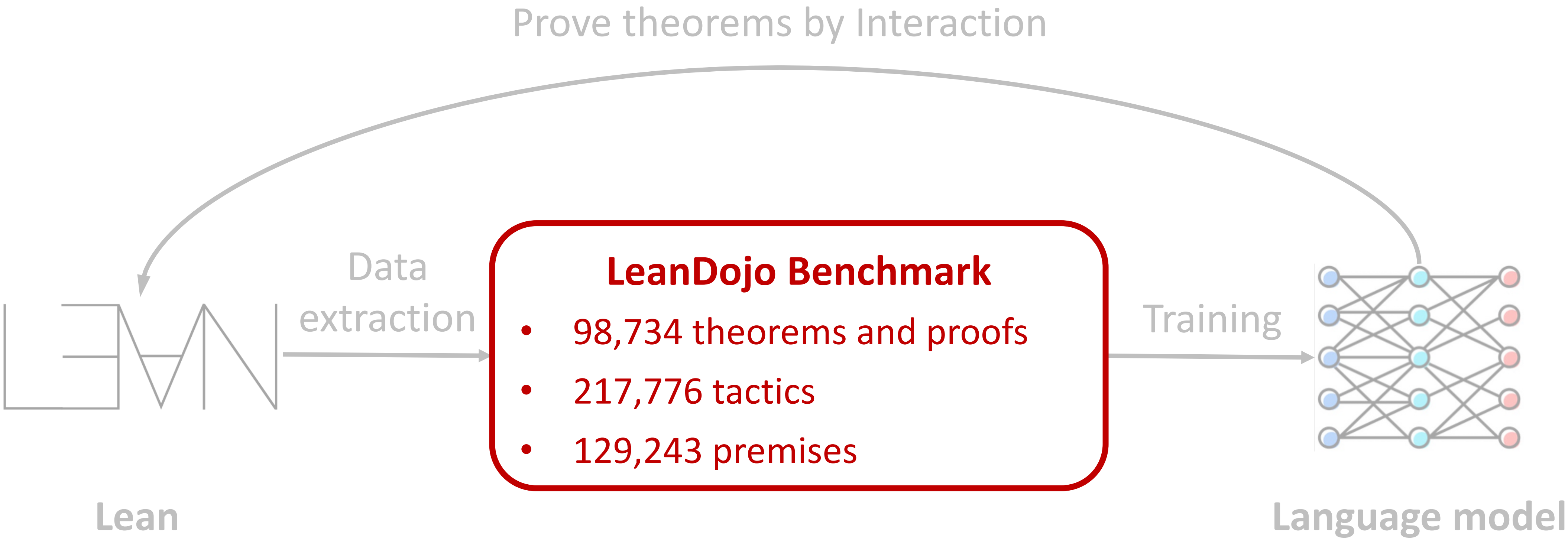
- Tactics rely on **premises**
  - Lemmas
  - Definitions

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
```



LeanDojo



From mathlib: https://github.com/leanprover-community/mathlib

Easy to construct your own benchmarks

Challenging Data Split

- random: Splitting theorems into training/testing randomly
- LLMs can prove seemingly nontrivial theorems by memorizing similar proofs in training

Challenging Data Split

- random: Splitting theorems into training/testing randomly
- LLMs can prove seemingly nontrivial theorems by memorizing similar proofs in training

src/algebra/quaternion.lean

```
lemma conj_mul : (a * b).conj = b.conj * a.conj := begin
  ext; simp; ring_exp
end

lemma conj_conj_mul : (a.conj * b).conj = b.conj * a := begin
  rw [conj_mul, conj_conj]
end

lemma conj_mul_conj : (a * b.conj).conj = b * a.conj := begin
  rw [conj_mul, conj_conj]
end
```

Challenging Data Split

- random: Splitting theorems into training/validation/testing randomly
- LLMs can prove seemingly nontrivial theorems by memorizing similar proofs in training

src/algebra/quaternion.lean

```
lemma conj_mul : (a * b).conj = b.conj * a.conj := begin
  ext; simp; ring_exp
end
```

Train

```
lemma conj_conj_mul : (a.conj * b).conj = b.conj * a := begin
  rw [conj_mul, conj_conj]
end
```

Test

```
lemma conj_mul_conj : (a * b.conj).conj = b * a.conj := begin
  rw [conj_mul, conj_conj]
end
```

Challenging Data Split

- random: Splitting theorems into training/validation/testing randomly
- LLMs can prove seemingly nontrivial theorems by memorizing similar proofs in training

src/algebra/quaternion.lean

```
lemma conj_mul : (a * b).conj = b.conj * a.conj := begin
  ext; simp; ring_exp
end
```

Train

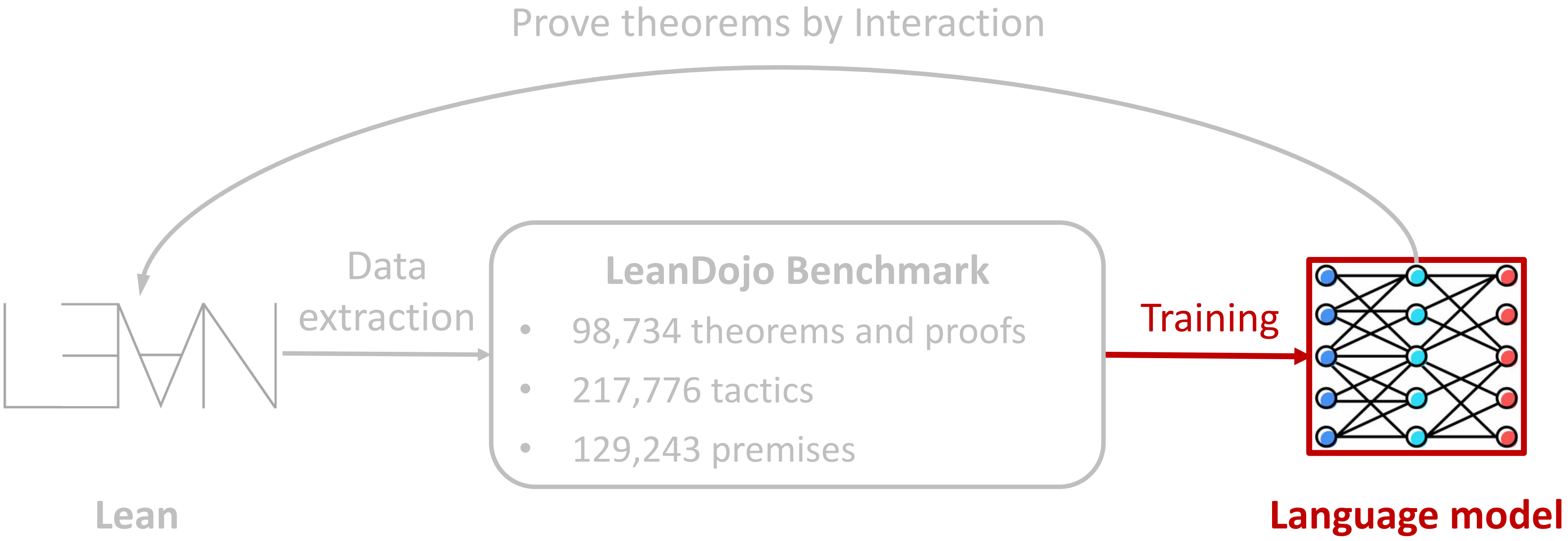
```
lemma conj_conj_mul : (a.conj * b).conj = b.conj * a := begin
  rw [conj_mul, conj_conj]
end
```

Test

```
lemma conj_mul_conj : (a * b.conj).conj = b * a.conj := begin
  rw [conj_mul, conj_conj]
end
```

- novel_premises: **Testing proofs must use >1 premise that is never used in training**

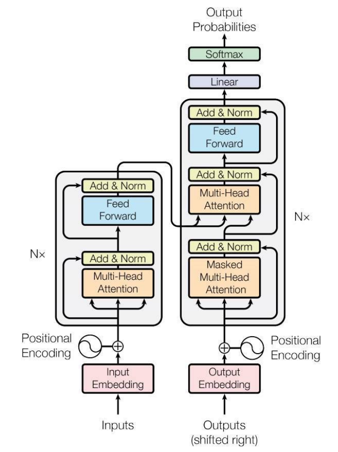
LeanDojo


Prove theorems by Interaction
Data extraction
LeanDojo Benchmark
98,734 theorems and proofs
217,776 tactics
129,243 premises
Training
Lean
Language model

Existing Models for Tactic Generation

State

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



[Vaswani et al., NeurIPS 2017]

→ rewrite mod_self

Tactic

ReProver: Retrieval-Augmented Prover

- Given a state, we retrieve premises from the set of **all accessible premises**

State

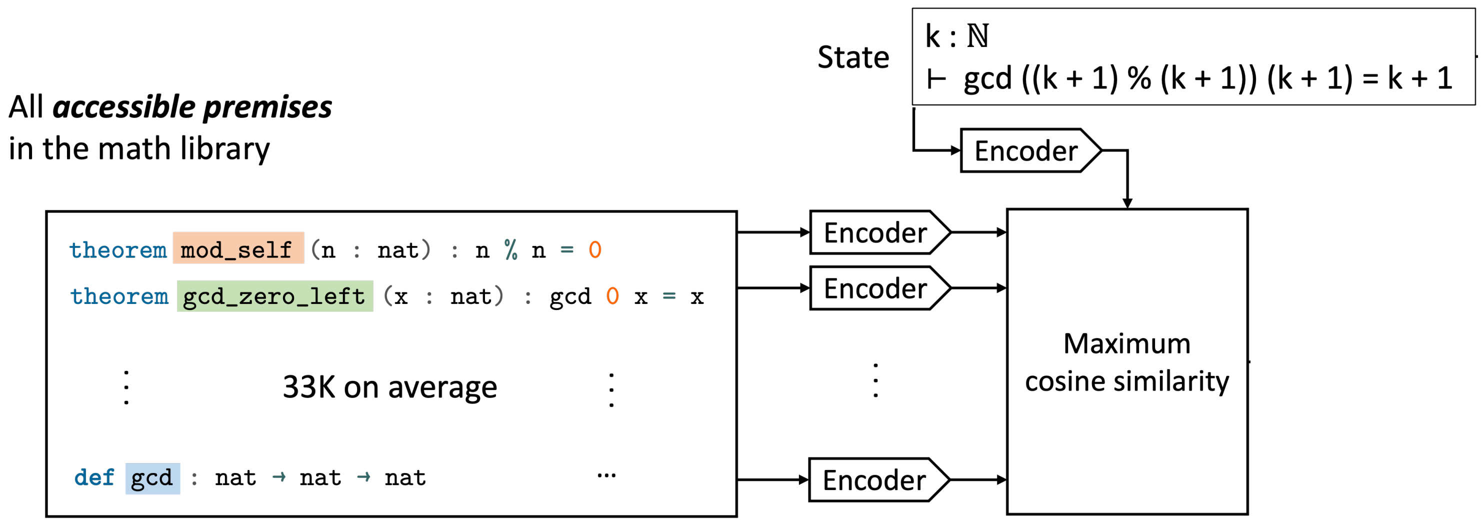
$k : \mathbb{N}$
$\vdash$ gcd ((k + 1) % (k + 1)) (k + 1) = k + 1

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                 …
```

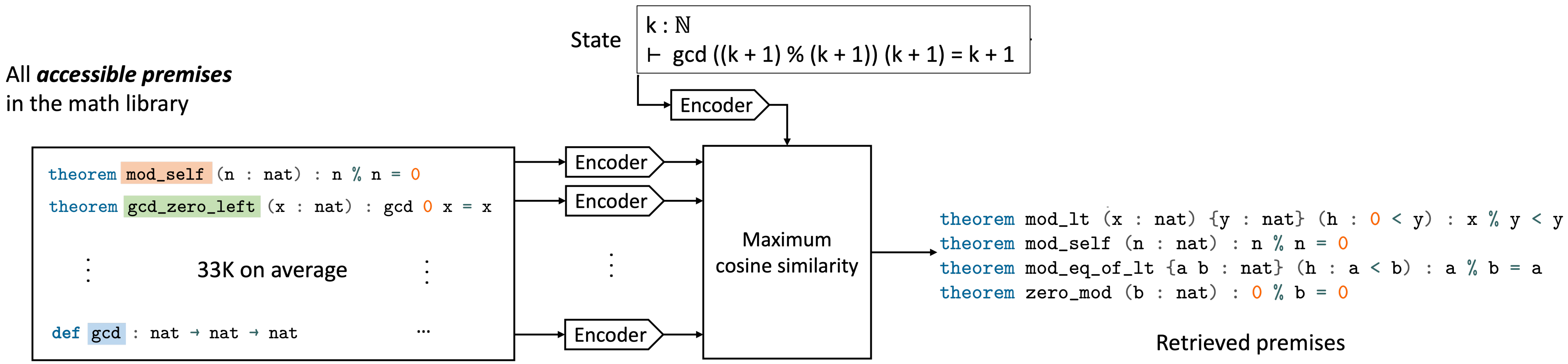
ReProver: Retrieval-Augmented Prover

- Given a state, we retrieve premises from the set of **all accessible premises**



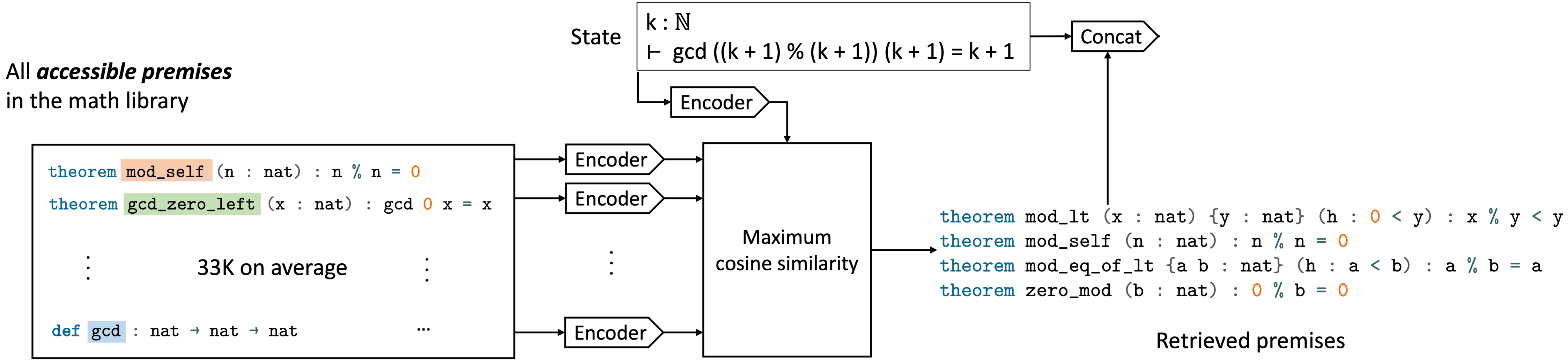
ReProver: Retrieval-Augmented Prover

- Given a state, we retrieve premises from the set of **all accessible premises**



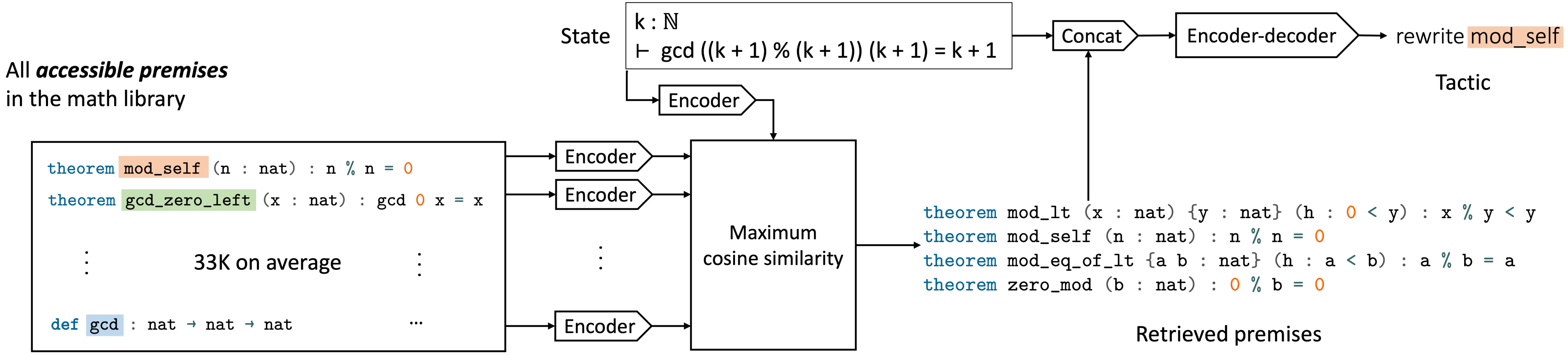
ReProver: Retrieval-Augmented Prover

- 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

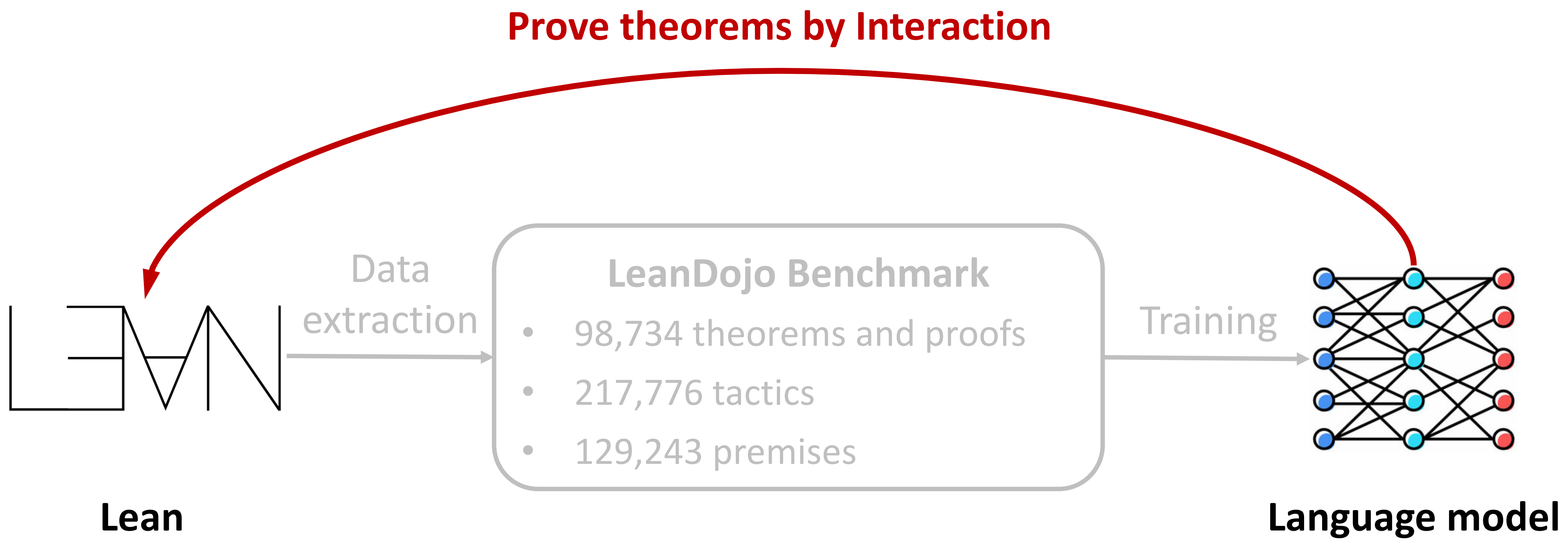


ReProver: Retrieval-Augmented Prover

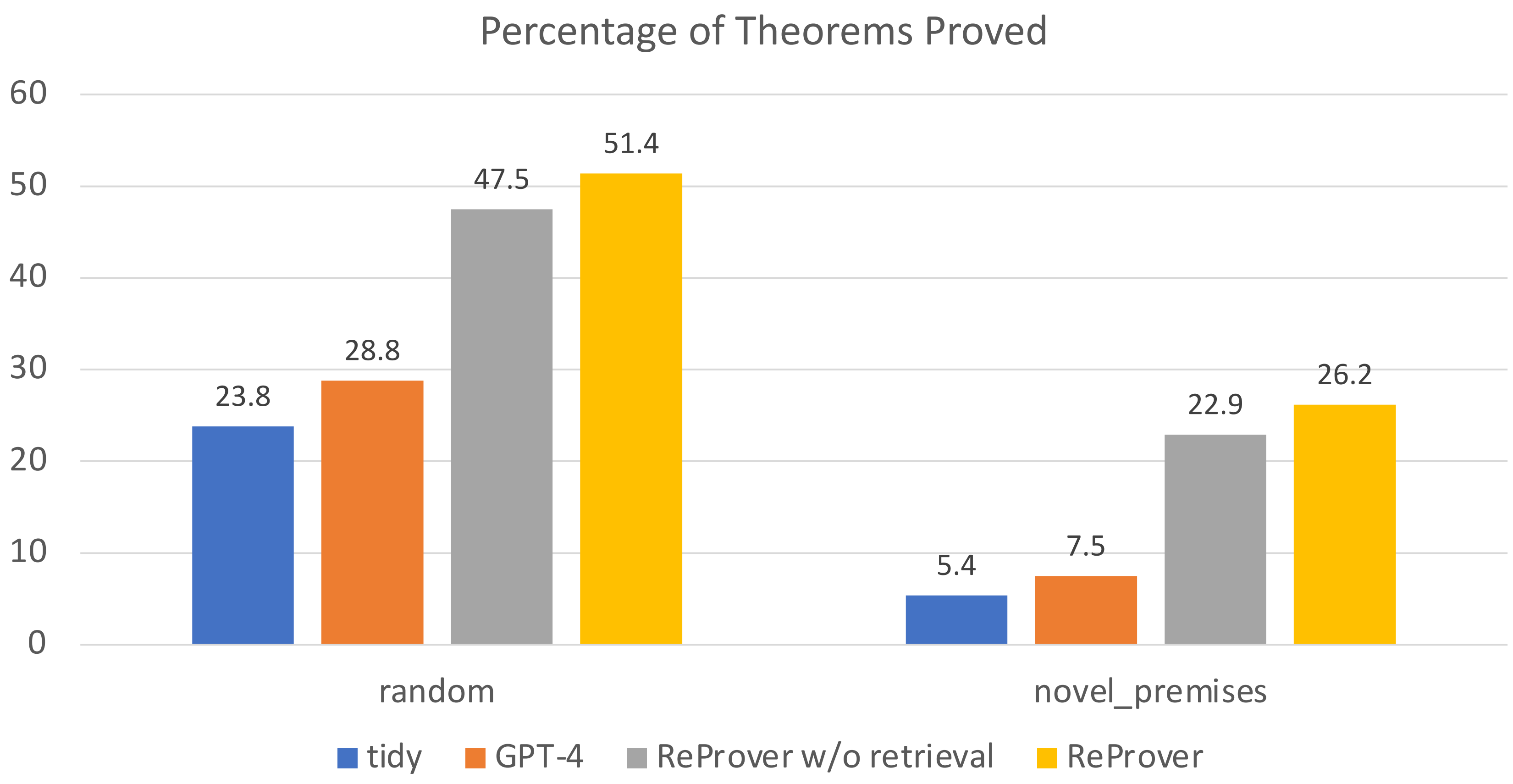
- Given a state, we retrieve premises from the set of **all accessible premises**
- Retrieved premises are concatenated with the state and used for tactic generation



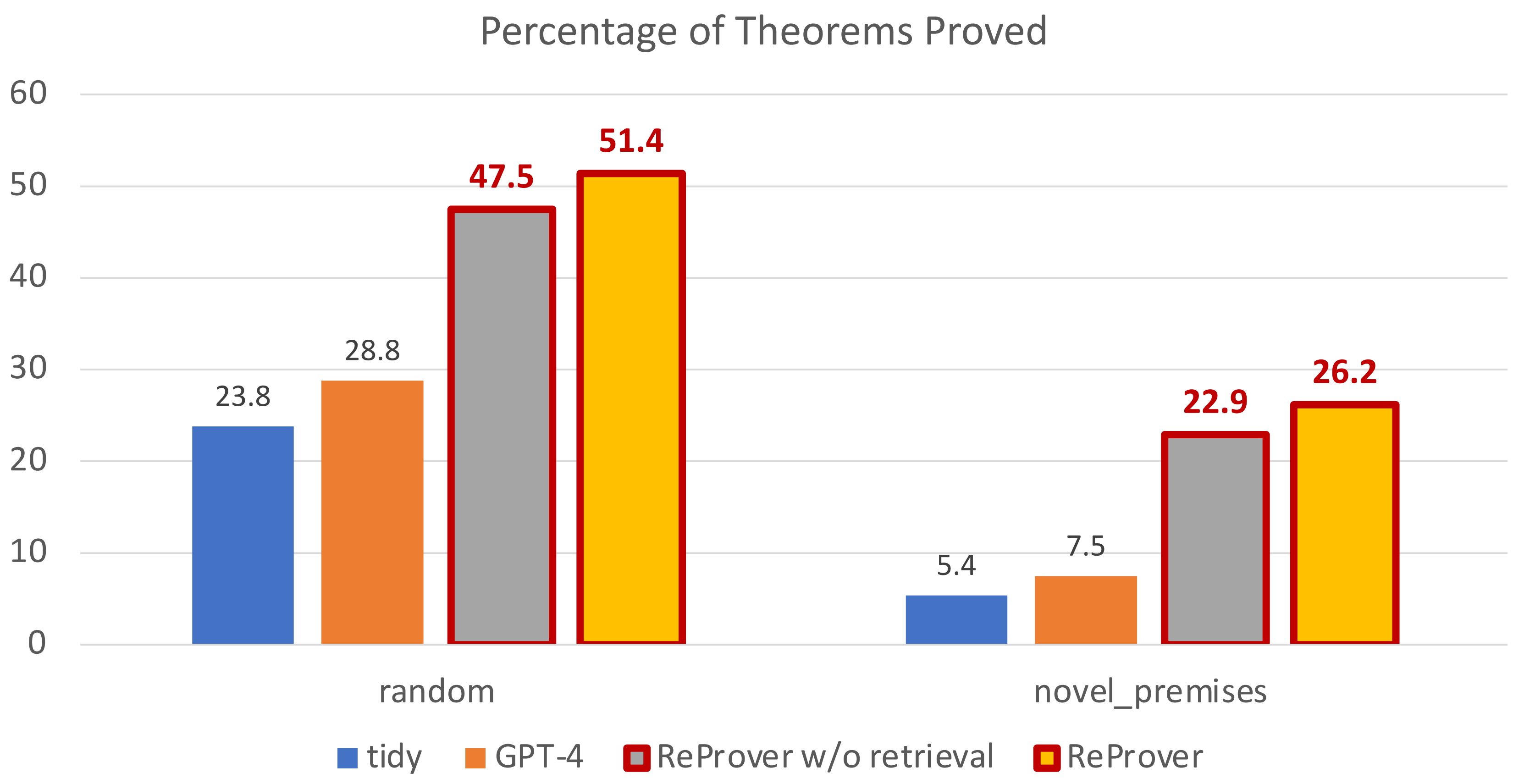
LeanDojo


Prove theorems by Interaction
Data extraction
LeanDojo Benchmark
• 98,734 theorems and proofs
• 217,776 tactics
• 129,243 premises
Training
Lean
Language model

Premise Retrieval Improves Theorem Proving



Premise Retrieval Improves Theorem Proving



Results on MiniF2F

- MiniF2F: Benchmark for formal math Olympiads [Zheng et al., MiniF2F, ICLR 2022]
- ReProver is competitive to state of the art
  - 26.5% vs. 25.9% (w/o RL) [Polu et al., "Formal Mathematics Statement Curriculum Learning", ICLR 2023]
  - Smaller model: 517M vs. 774M
  - Much smaller compute: 120 hours vs. 48K hours (We don’t pretrain on math-specific data)
  - Open-source!

Discovering New Proofs ON ProofNet

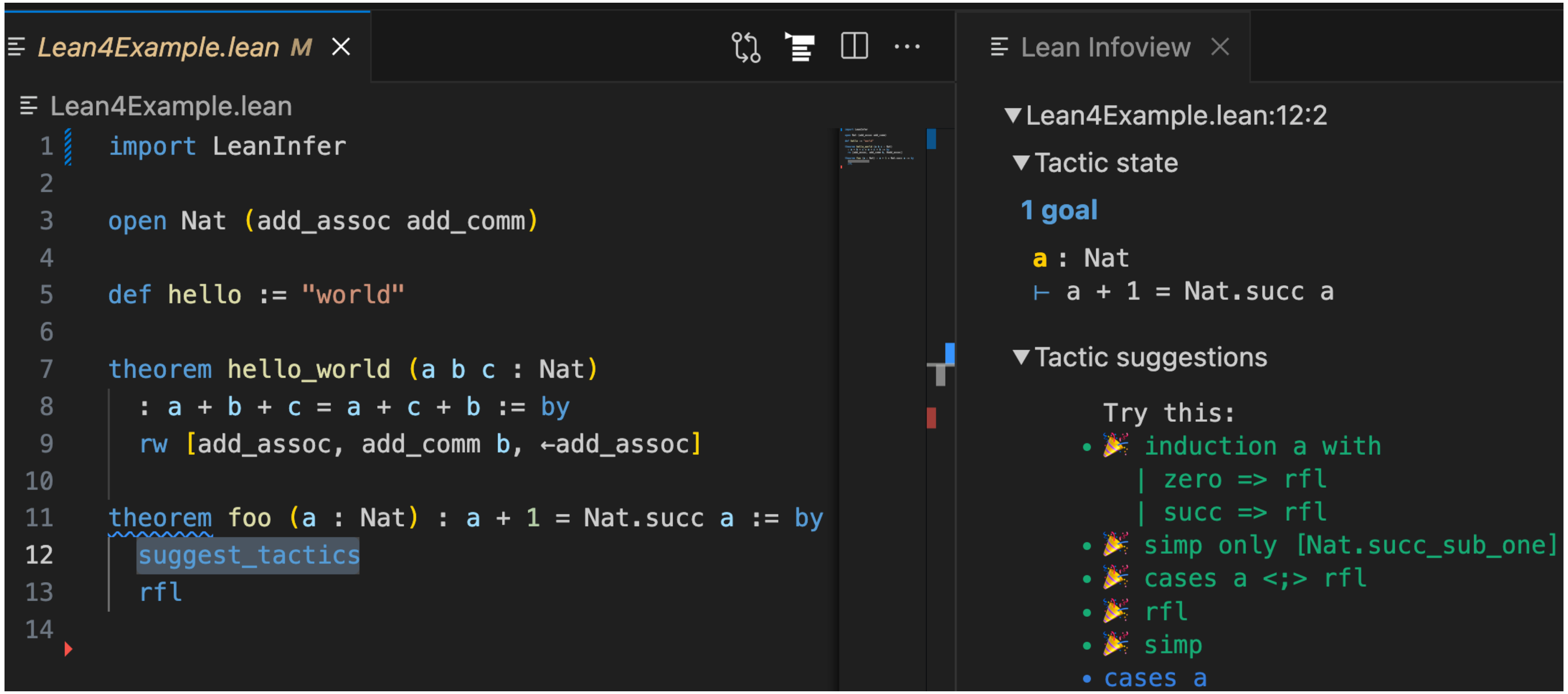
[Azerbayev et al., ProofNet, 2023]

We evaluate the model on 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
```

LLMs as Copilots for Theorem Proving



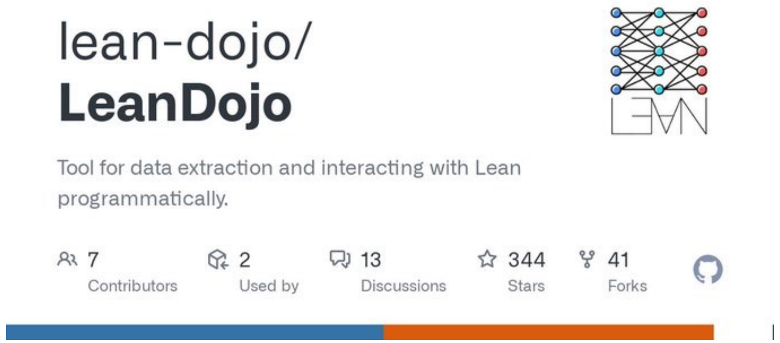
[Song et al., "Towards Large Language Models as Copilots for Theorem Proving in Lean", NeurIPS MATH-AI Workshop 2023]

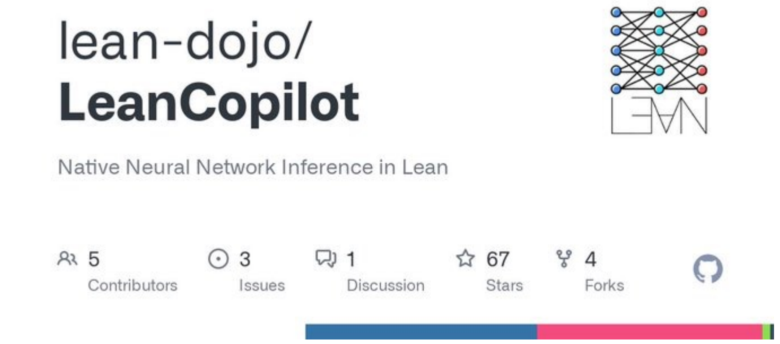
Posters

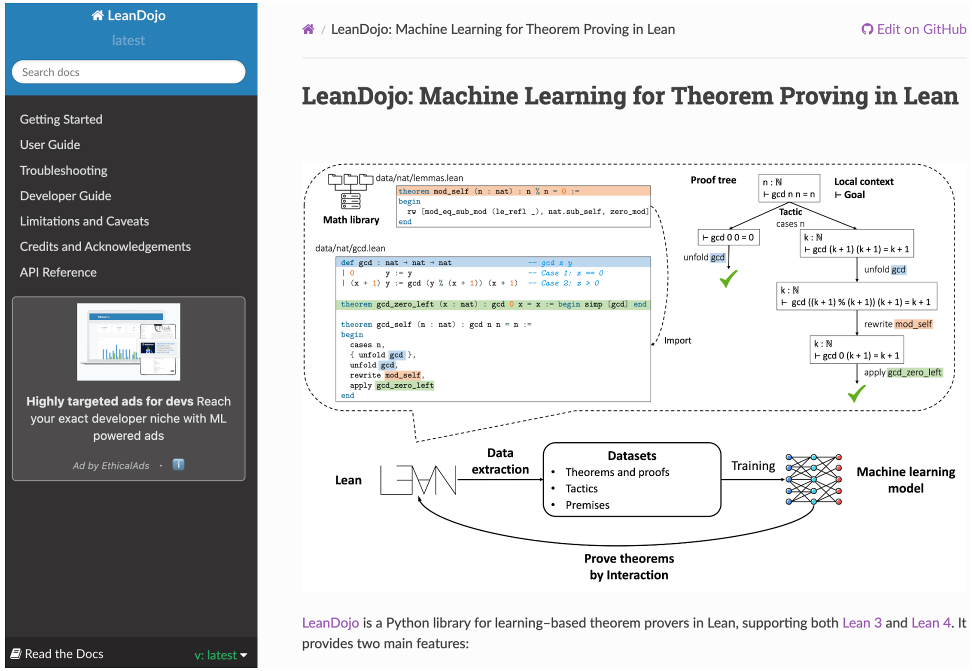
LeanDojo: 10:45 AM TODAY!

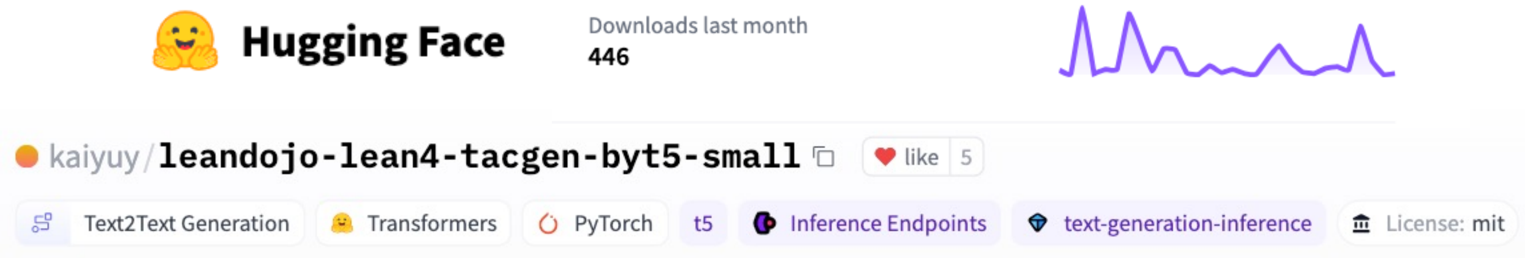
Lean Copilot: 3 PM Friday @ MATH-AI Workshop

Open Code, Models, Data









Team

Aidan Swope

Alex Gu

Rahul Chalamala

Peiyang Song

Shixing Yu

Saad Godil

Ryan Prenger

Anima Anandkumar

LeanDojo

- Project webpage: leandojo.org
- **Poster: Today 10:45 AM–12:45 PM**