

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  $\alpha$ ) : 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

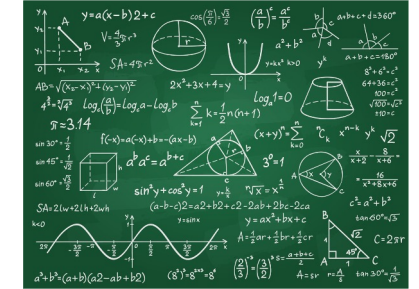
```
theorem set_inter_comm (s t : Set  $\alpha$ ) : s  $\cap$  t = t  $\cap$  s
```



Proof

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

Formalize



Mathematics

```
attachEvent("onreadystatechange",H),e.attachE
boolean Number String Function Array Date RegE
_={};function F(e){var t=_[e]={};return b.ea
t[1]==!&&e.stopOnFalse}{r=!1;break}n=1,u&
?o=u.length:r&&(s=t,c(r))}return this},remove
ction(){return u=[],this},disable:function(){
re:function(){return p.fireWith(this,argument
ending",r={state:function(){return n},always:
romise)?e.promise().done(n.resolve).fail(n.re
id(function(){#s},t[1^e][2],disable,t[2][2].
e=0,n=h.call(arguments),r=n.length,i=1==r|e&
(r),l=Array(r);r>t;t++n[t]&&&.isFunction(n[t
]><table></table><a href="/a">a</a><input typ
/TagName("input")[0],r.style.cssText="top:1px
est(r.getAttribute("style")),hrefNormalized:
```

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  $\alpha$ ) : s  $\cap$  t = t  $\cap$  s
```



Proof

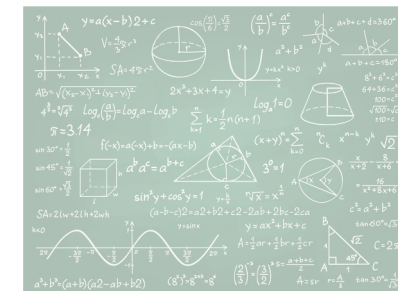
```
ext x  
simp [Set.mem_inter_iff]  
constructor  
· rintro ⟨xs, xt⟩  
  exact ⟨xt, xs⟩  
· rintro ⟨xt, xs⟩  
  exact ⟨xs, xt⟩
```



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



Formalize



Mathematics

```
attachEvent("onreadystatechange",H),e.attach  
Boolean Number String Function Array Date RegE  
_={};function F(e){var t=_[e]={};return b.ear  
t[1]==!&&e.stopOnFalse){r=!;break}n=1,u&  
?o=u.length:r&&(s=t,c(r))return this},remove  
ction(){return u=[],this},disable:function()  
re:function(){return p.fireWith(this,argument  
ending",r={state:function(){return n},always:  
romise)?e.promise().done(n.resolve).fail(n.re  
id(function(){#s},t[1^e][2].disable,t[2][2].  
=0,n=h.call(arguments),r=n.length,i!=r|e&  
(r),l=Array(r);r>t;t++)n[t]&&b.isFunction(n[t  
><table></table><a href="/a">a</a><input typ  
TagName("input")][0],r.style.cssText="top:1px  
est(r.getAttribute("style")),hrefNormalized:
```

Software

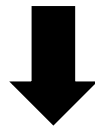
- Theorems/proofs represented formally as programs
- Proofs can be checked easily. No room for hallucination

How to Prove Theorems (with Machine Learning)?

Proof Assistants (Interactive Theorem Provers)

Theorem

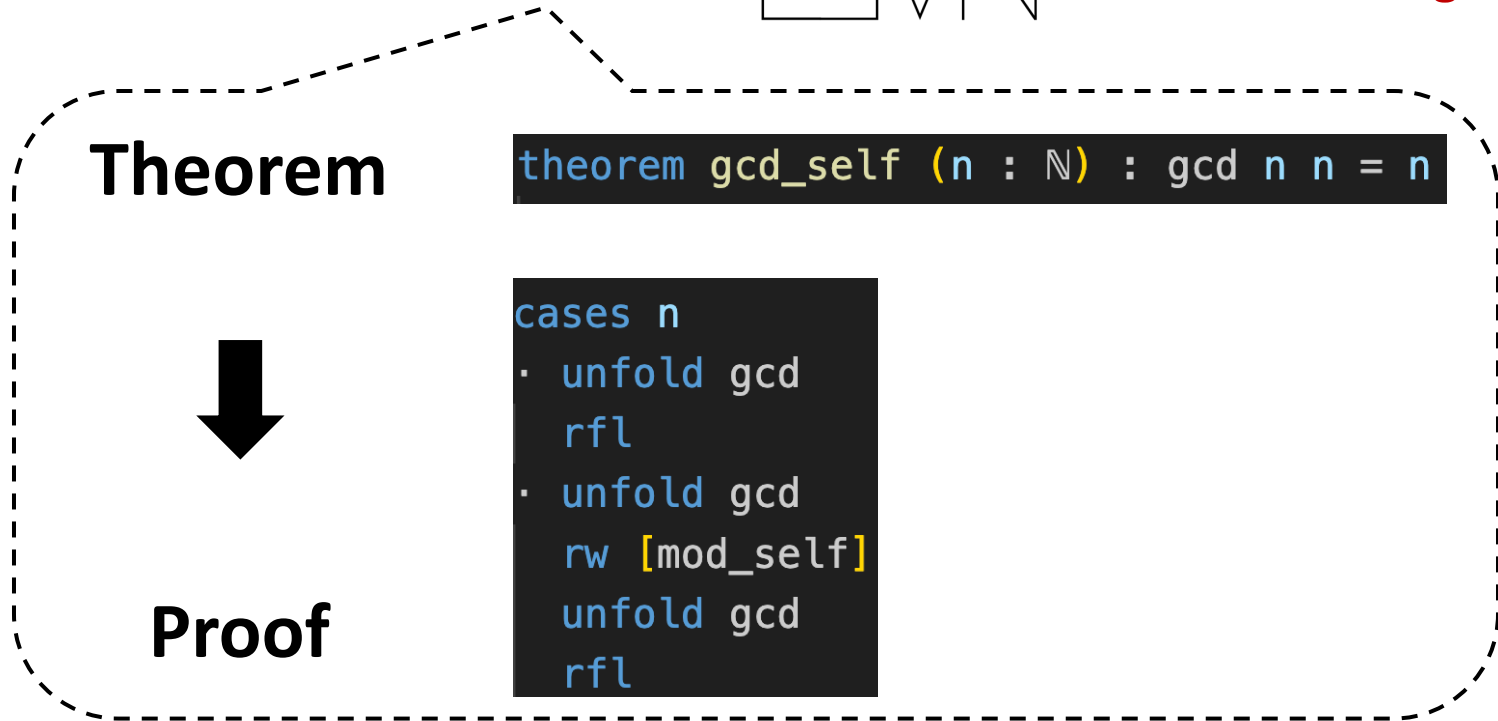
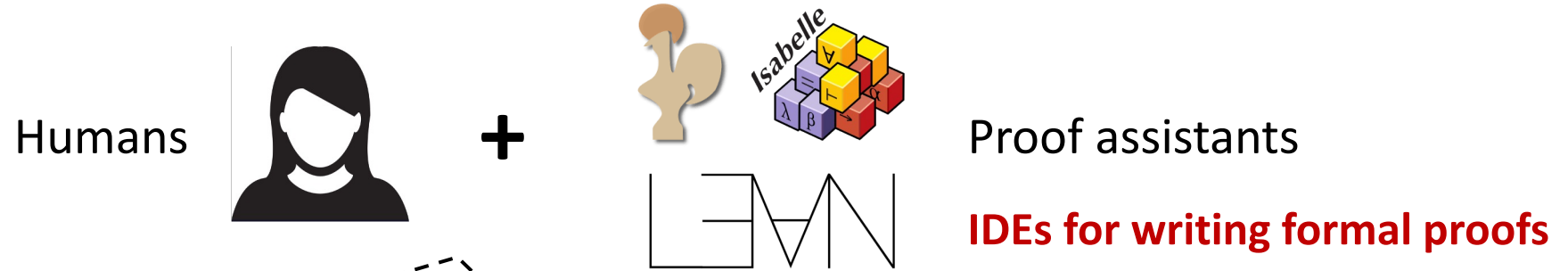
```
theorem gcd_self (n : ℕ) : gcd n n = n
```



Proof

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


Proof Assistants (Interactive Theorem Provers)



Generating Proof Steps (Tactics)

```
theorem add_abc :  $\forall a b c : \mathbb{N}, a + b + c = a + c + b := by$   
  intro a b c  
  rw [Nat.add_right_comm]
```

Generating Proof Steps (Tactics)

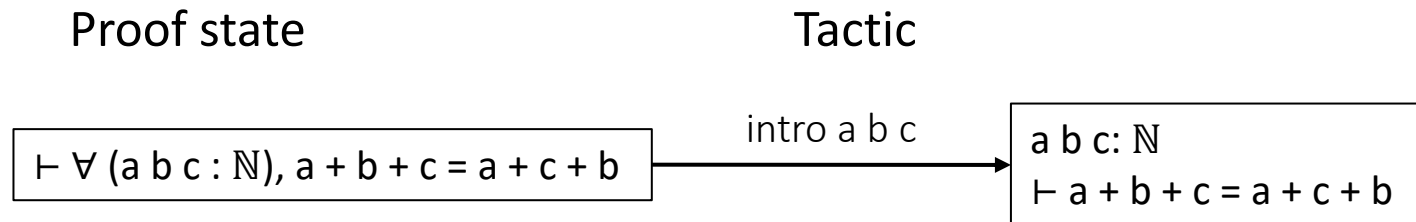
```
theorem add_abc :  $\forall a b c : \mathbb{N}, a + b + c = a + c + b :=$  by  
  intro a b c  
  rw [Nat.add_right_comm]
```

Proof state

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

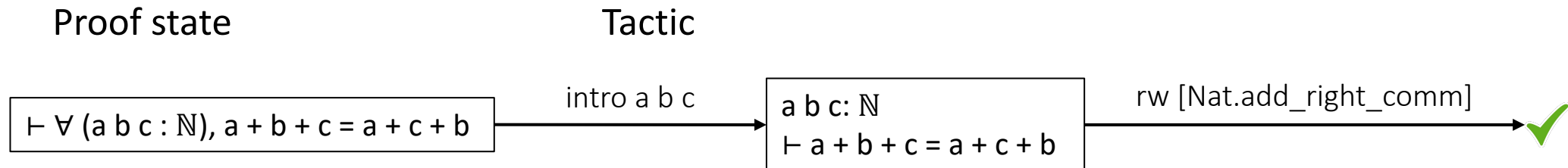
Generating Proof Steps (Tactics)

```
theorem add_abc :  $\forall a b c : \mathbb{N}, a + b + c = a + c + b := by$   
  intro a b c  
  rw [Nat.add_right_comm]
```



Generating Proof Steps (Tactics)

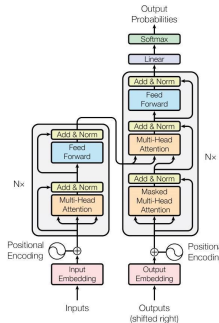
```
theorem add_abc :  $\forall a b c : \mathbb{N}, a + b + c = a + c + b := by$   
  intro a b c  
  rw [Nat.add_right_comm]
```



Generating Proof Steps (Tactics)

```
theorem add_abc :  $\forall a b c : \mathbb{N}, a + b + c = a + c + b :=$  by  
  intro a b c  
  rw [Nat.add_right_comm]
```

Tactic generator



Input: Proof state

Output: Tactic

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

intro a b c

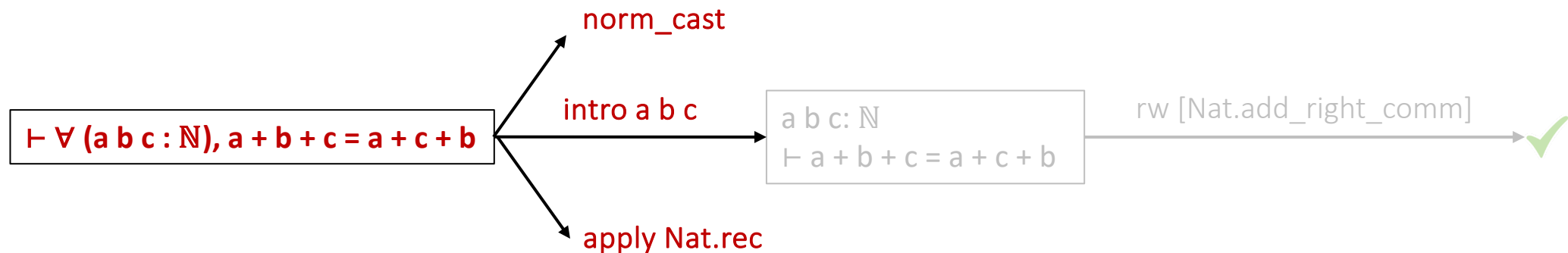
$a b c : \mathbb{N}$
 $\vdash a + b + c = a + c + b$

rw [Nat.add_right_comm]



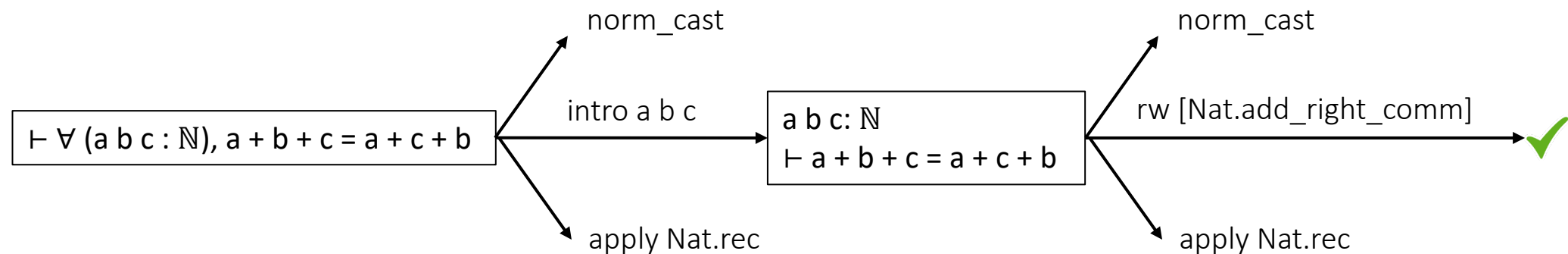
Generating Proof Steps (Tactics)

```
theorem add_abc :  $\forall a b c : \mathbb{N}, a + b + c = a + c + b := by$   
  intro a b c  
  rw [Nat.add_right_comm]
```



Searching for Proofs

```
theorem add_abc :  $\forall a b c : \mathbb{N}, a + b + c = a + c + b :=$  by  
  intro a b c  
  rw [Nat.add_right_comm]
```

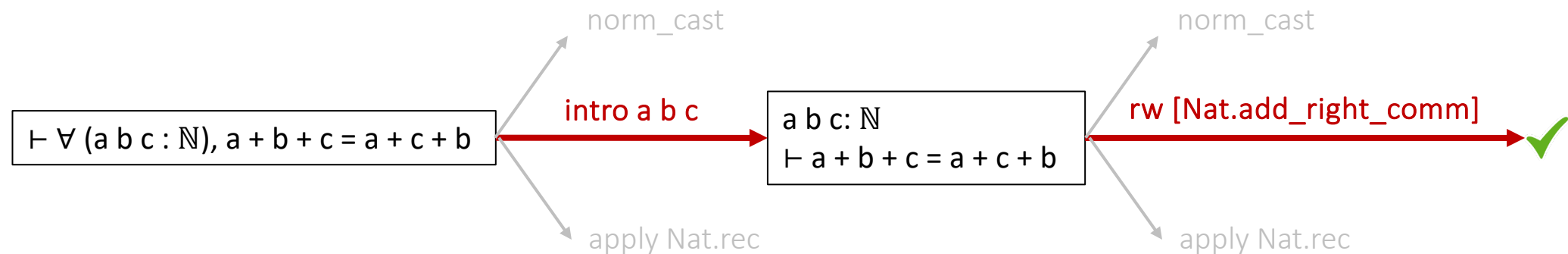


Searching for Proofs

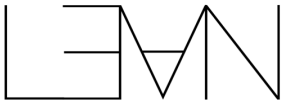
```
theorem add_abc :  $\forall a b c : \mathbb{N}, a + b + c = a + c + b :=$  by
  intro a b c
  rw [Nat.add_right_comm]
```

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

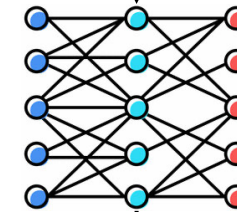
Han et al., PACT, 2022

Polu et al., 2023

Lample et al., HTPS 2022

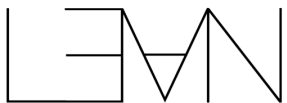
Wang et al., DT-Solver, 2023

Input: Proof state
 $n : \mathbb{N}$
 $\vdash \text{gcd } n \ n = n$



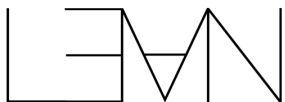
Output: Tactic (or full proof)
cases n

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

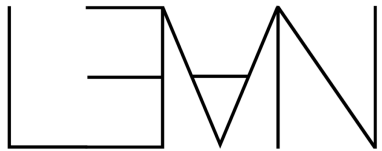
LLMs for Theorem Proving



	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

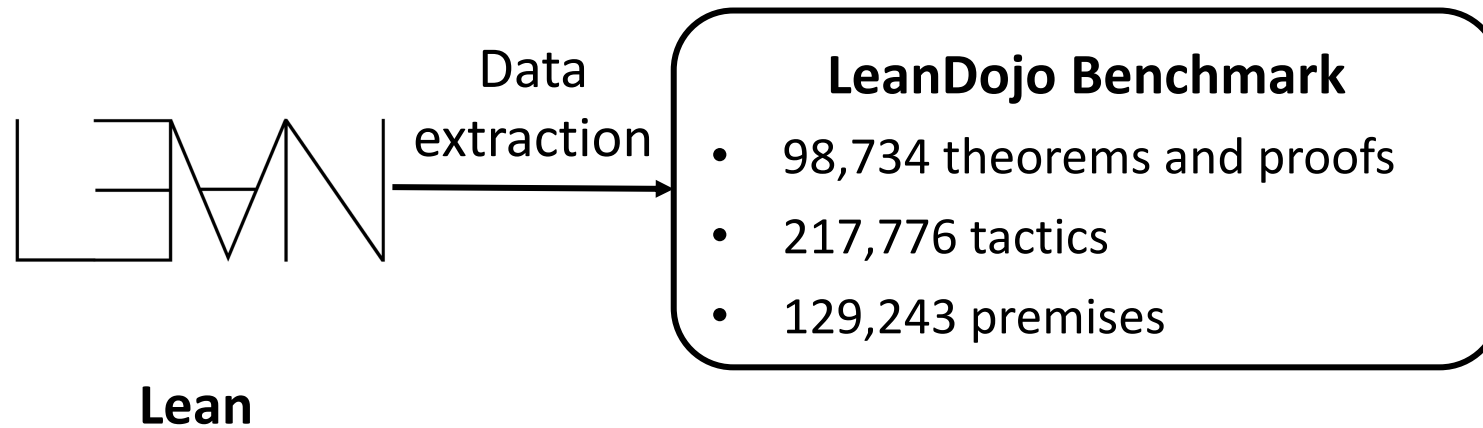
[Yang et al., "LeanDojo: Theorem Proving with Retrieval-Augmented Language Models", NeurIPS 2023]

LeanDojo

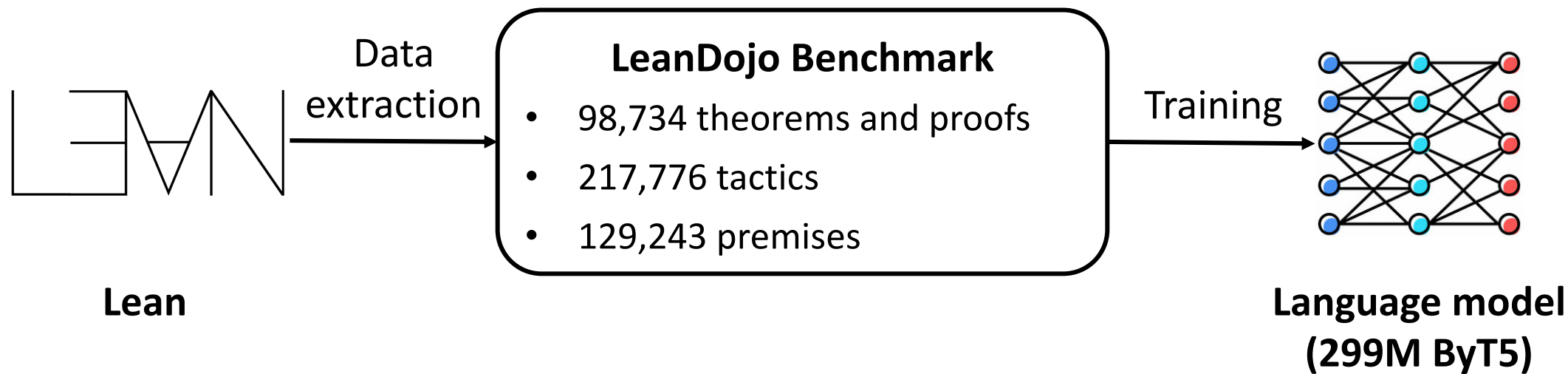


Lean

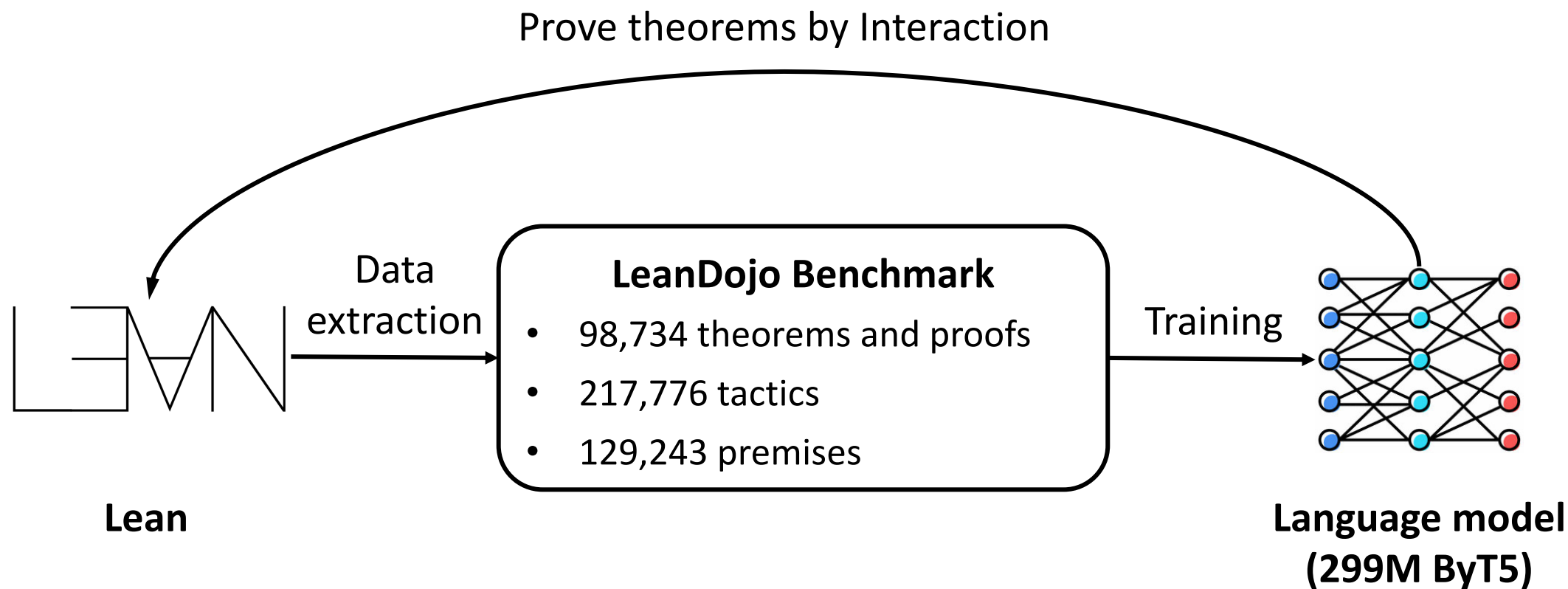
LeanDojo



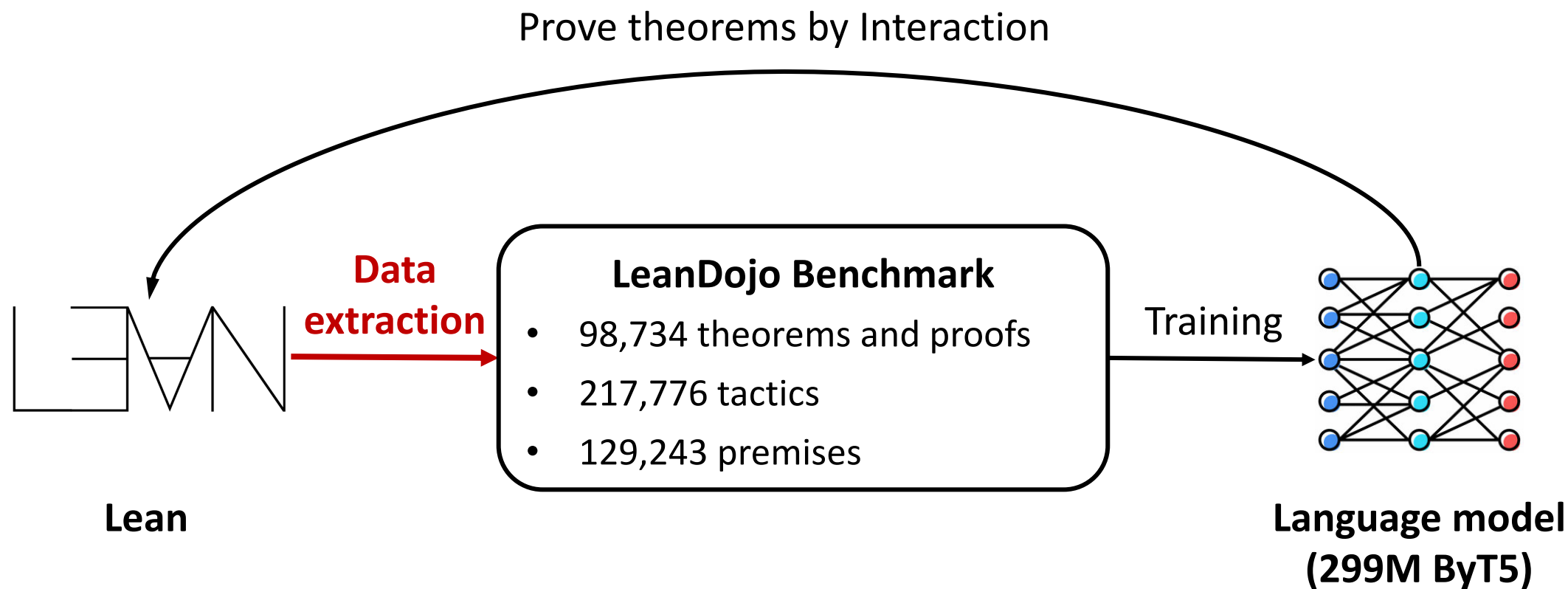
LeanDojo



LeanDojo



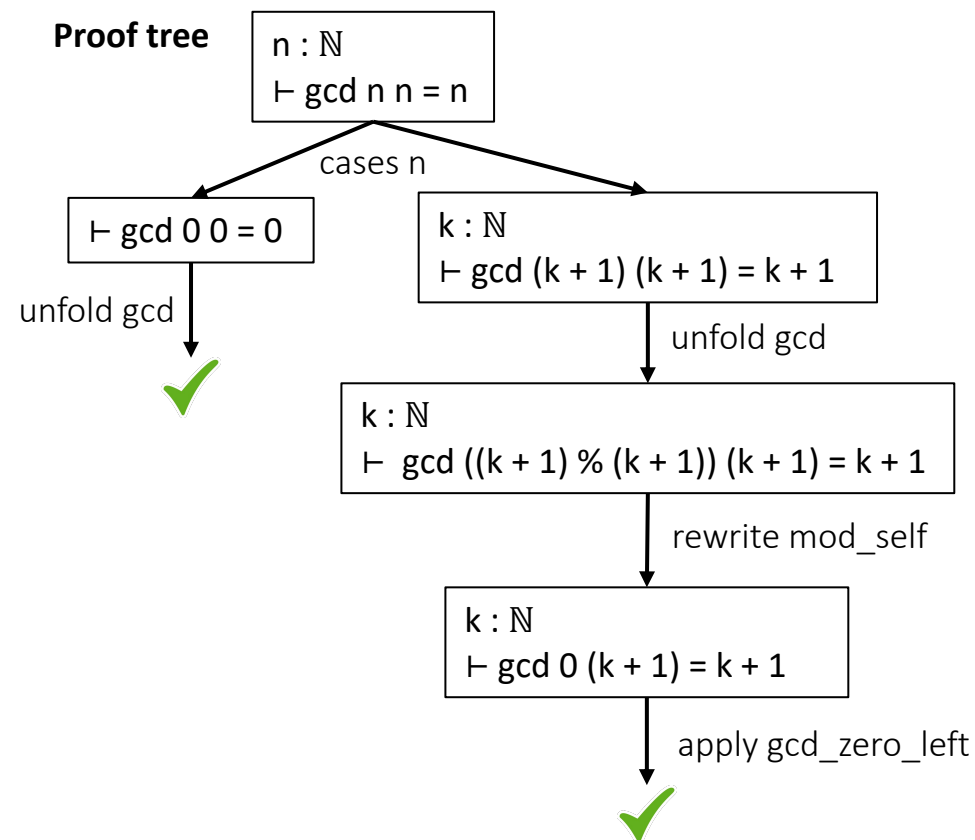
LeanDojo



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

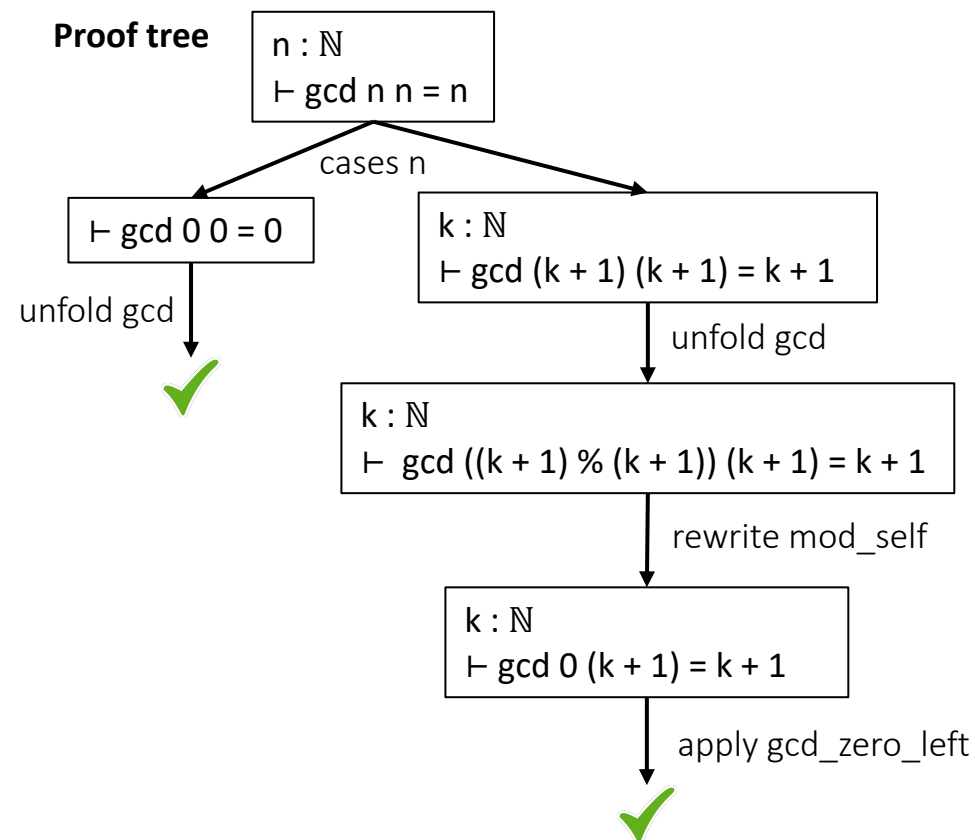
```
theorem gcd_self (n : nat) : gcd n n = n :=  
begin  
  cases n,  
  { unfold gcd },  
  unfold gcd,  
  rewrite mod_self,  
  apply gcd_zero_left  
end
```



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

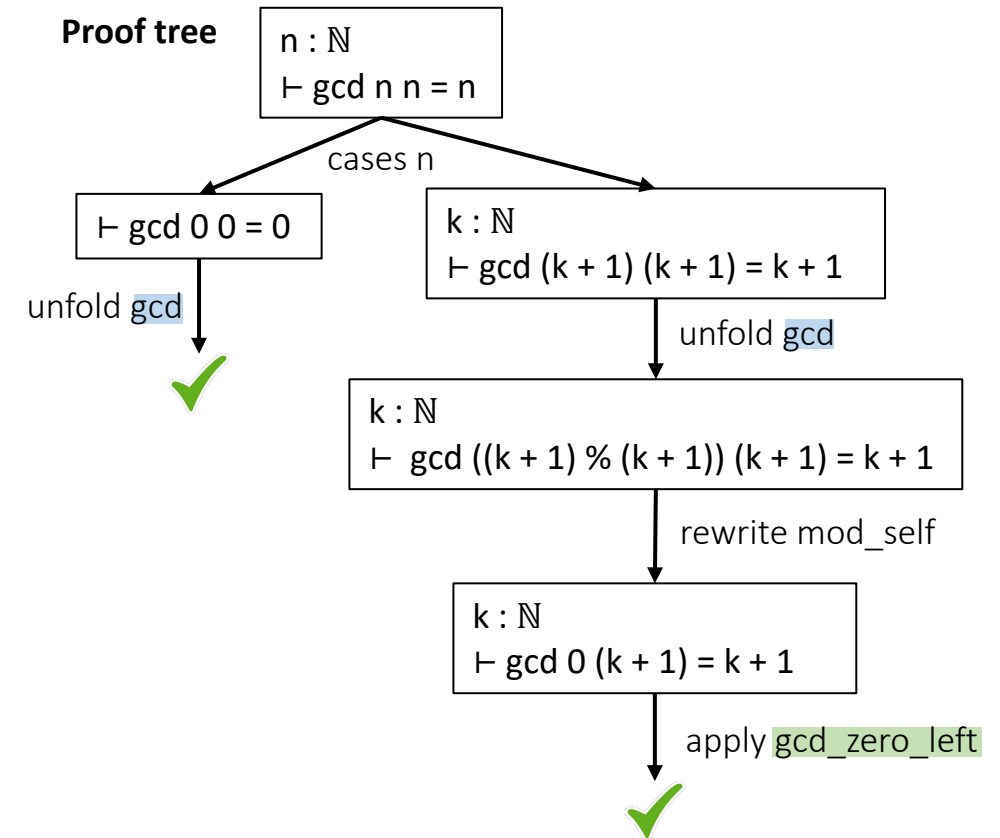
- Tactics rely on **premises**
 - Lemmas
 - Definitions

data/nat/gcd.lean

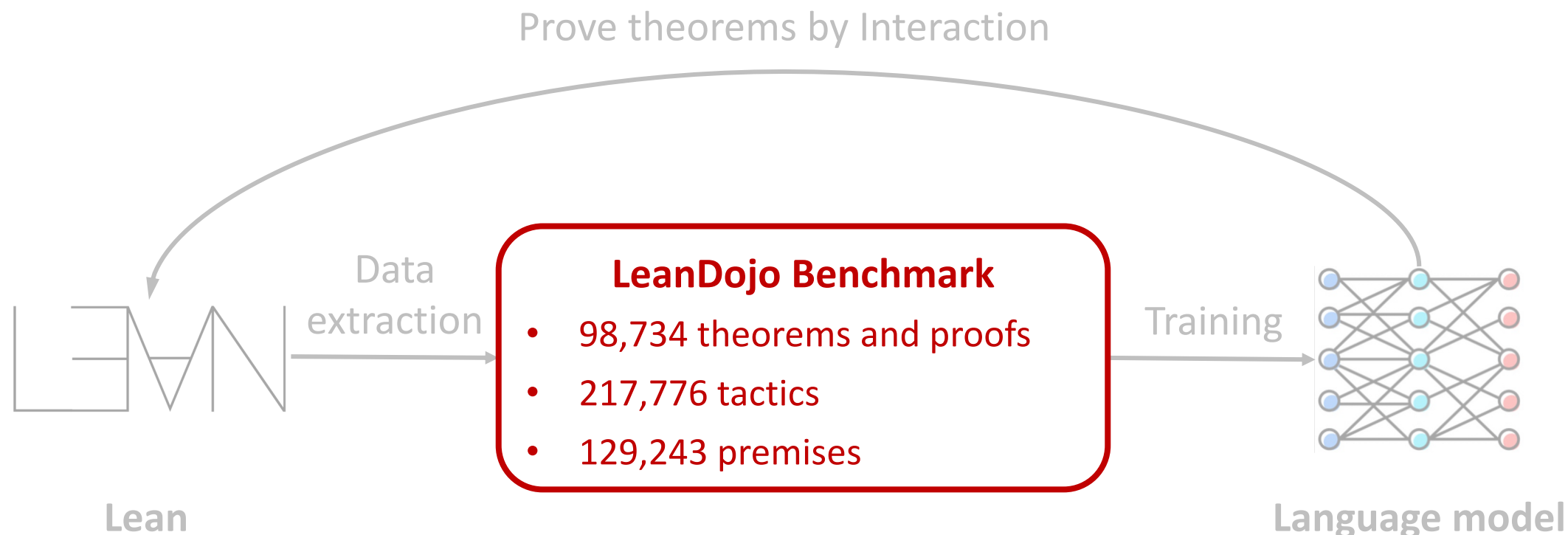
```
def gcd : nat → nat → nat -- gcd z y
| 0 y := y -- Case 1: z == 0
| (x + 1) y := gcd (y % (x + 1)) (x + 1) -- Case 2: z > 0

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

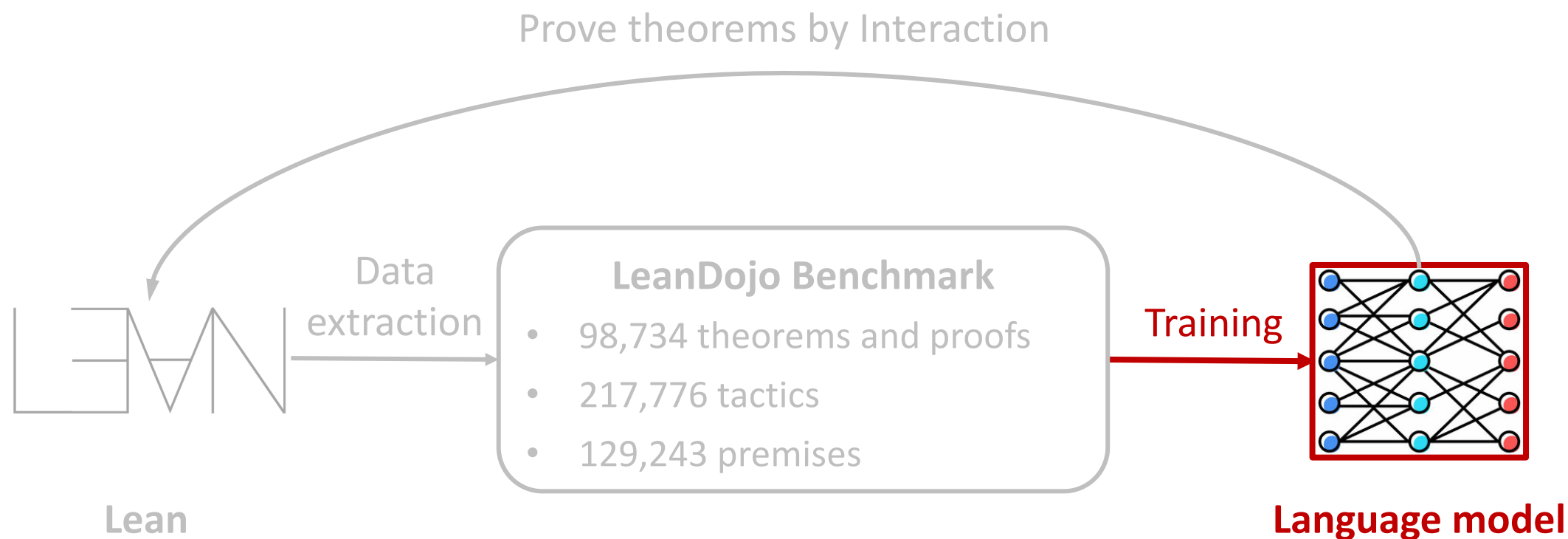
```
lemma conj_conj_mul : (a.conj * b).conj = b.conj * a := begin  
  rw [conj_mul, conj_conj]  
end
```

Test

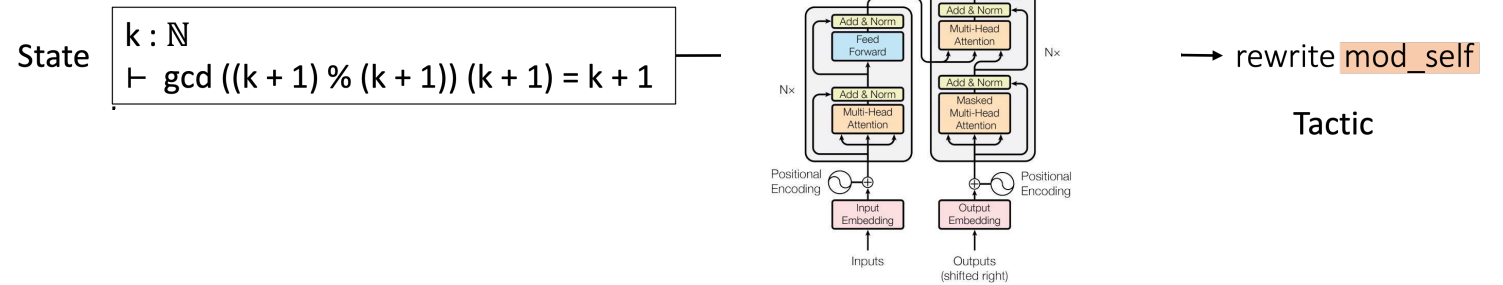
```
lemma conj_mul_conj : (a * b.conj).conj = b * a.conj := begin  
  rw [conj_mul, conj_conj]  
end
```

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

LeanDojo



Existing Models for Tactic Generation



[Vaswani et al., NeurIPS 2017]

ReProver: Retrieval-Augmented Prover

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

All *accessible premises*
in the math library

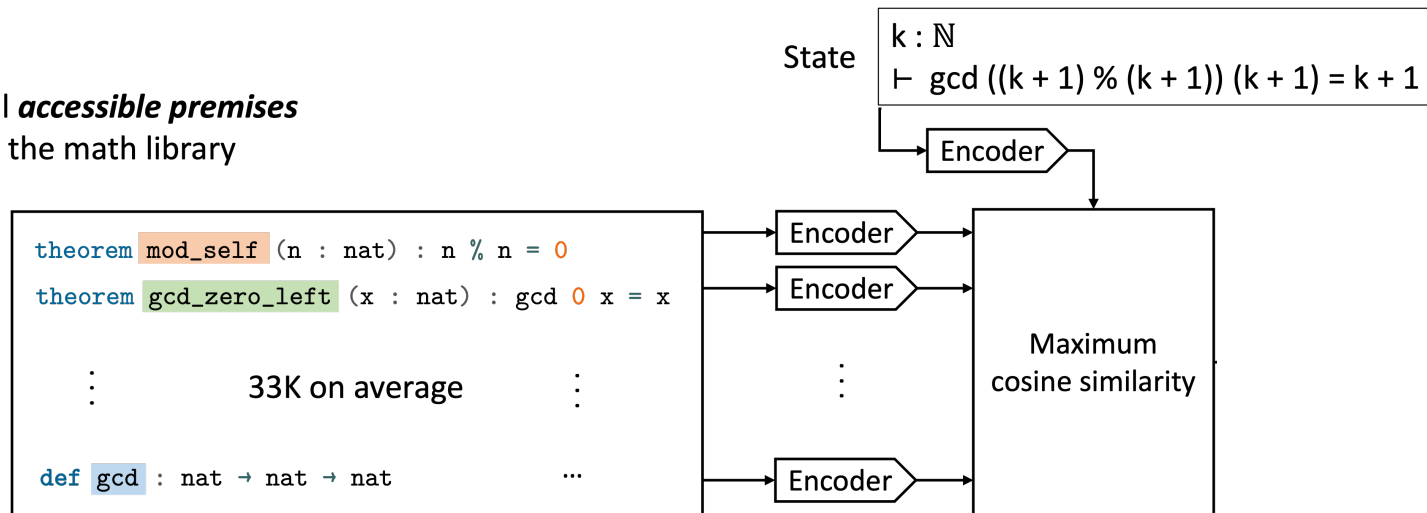
```
theorem mod_self (n : nat) : n % n = 0
theorem gcd_zero_left (x : nat) : gcd 0 x = x
  ⋮
  33K on average
  ⋮
def gcd : nat → nat → nat
```

State $\boxed{\begin{array}{l} k : \mathbb{N} \\ \vdash \text{gcd } ((k + 1) \% (k + 1)) (k + 1) = k + 1 \end{array}}$

ReProver: Retrieval-Augmented Prover

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

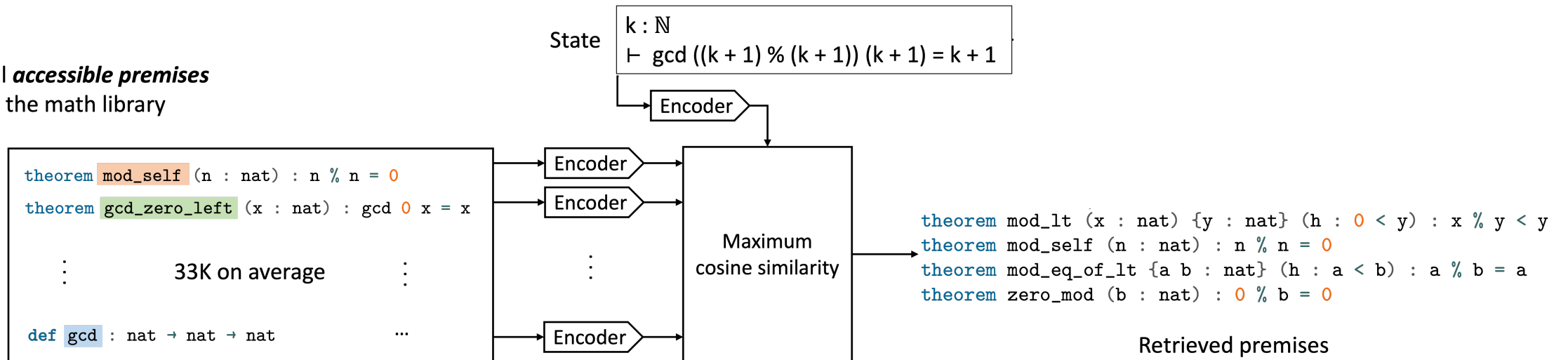
All *accessible premises*
in the math library



ReProver: Retrieval-Augmented Prover

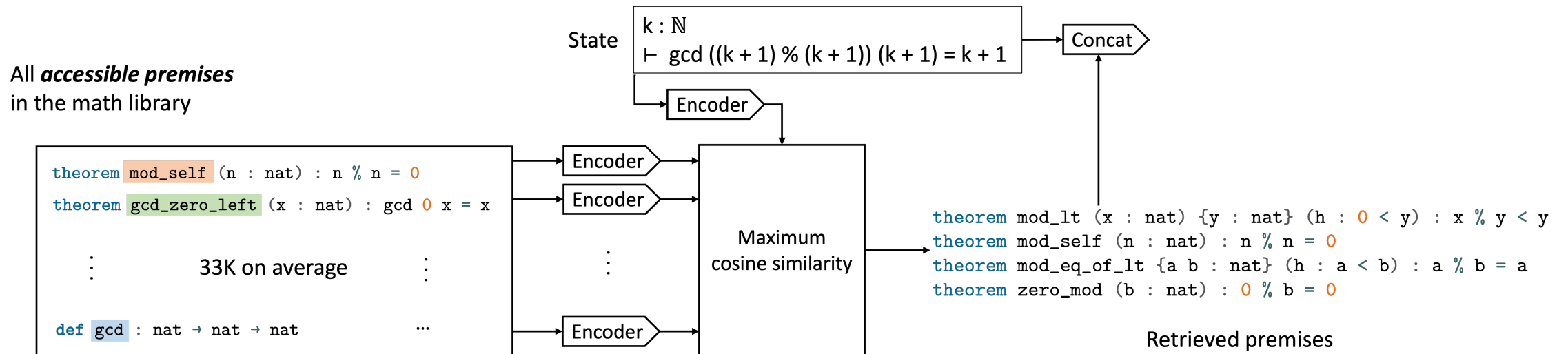
- Given a state, we retrieve premises from the set of **all accessible premises**

All *accessible premises*
in the math library



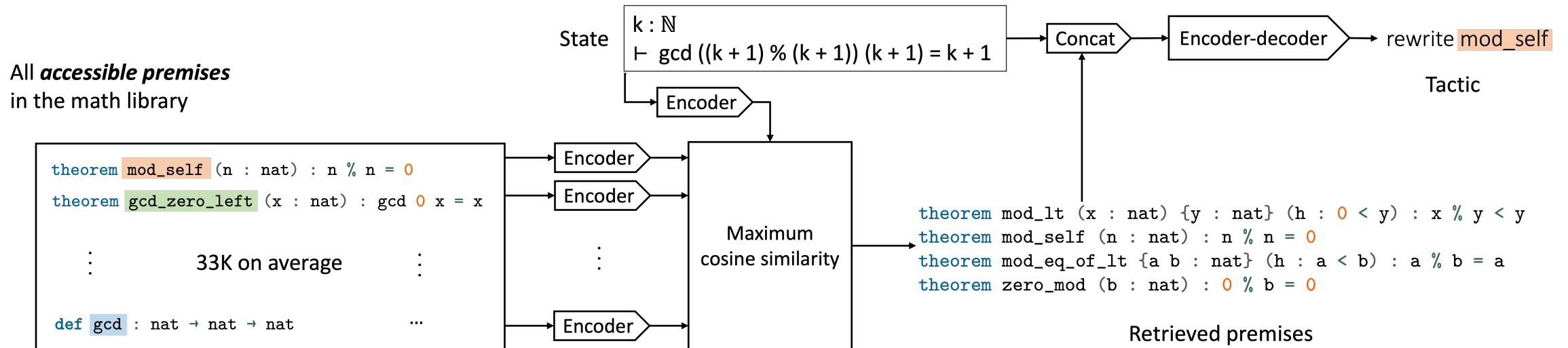
ReProver: Retrieval-Augmented Prover

- 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



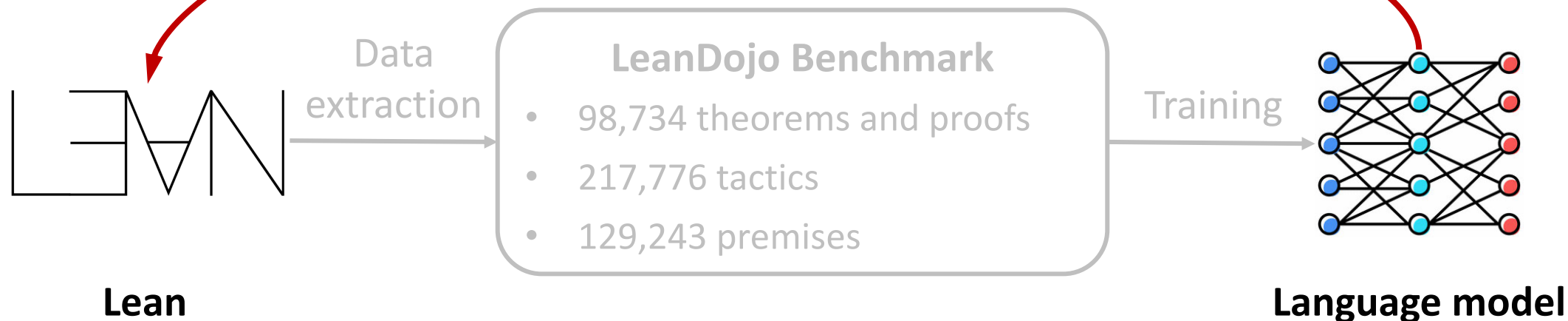
ReProver: Retrieval-Augmented Prover

- 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

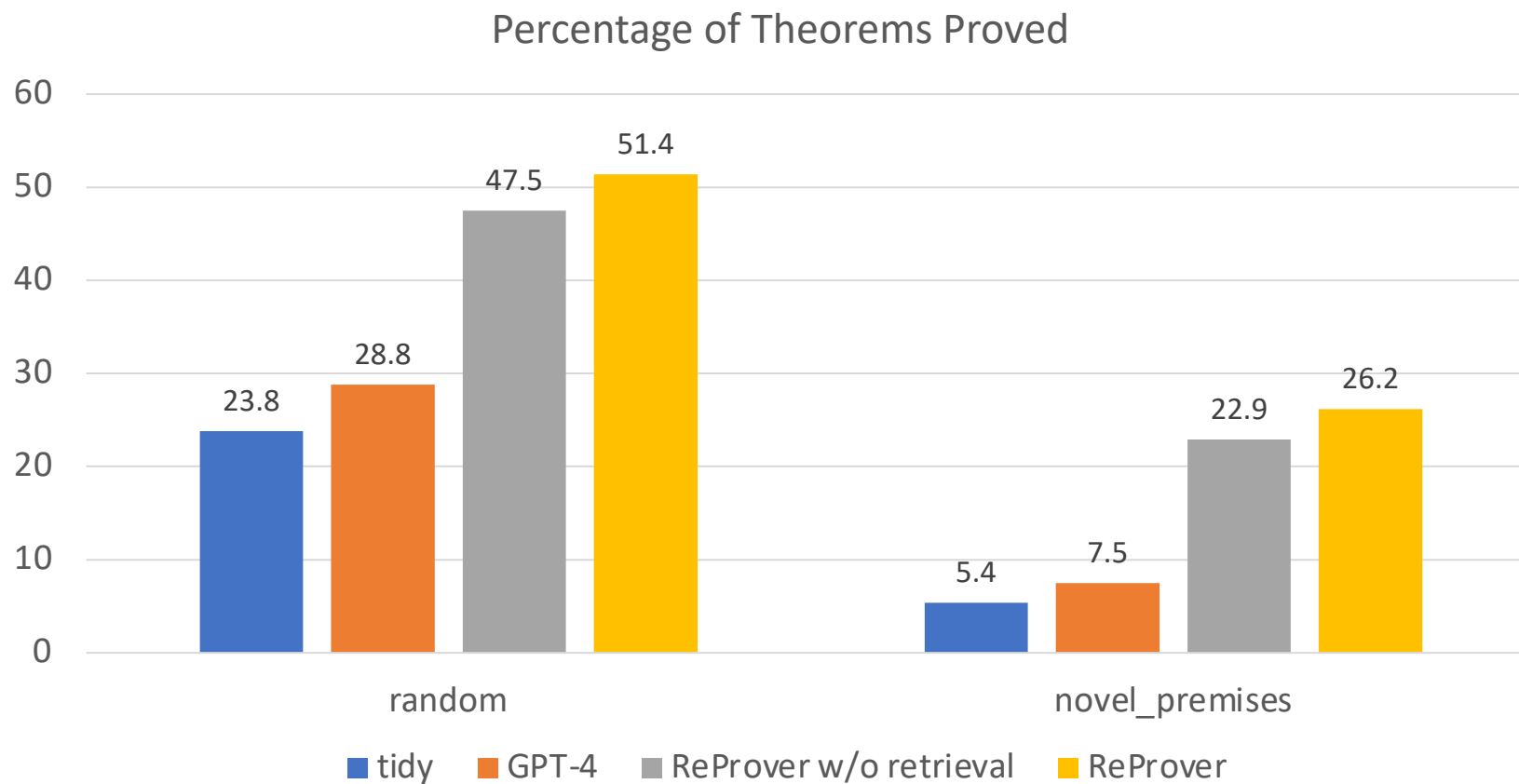


LeanDojo

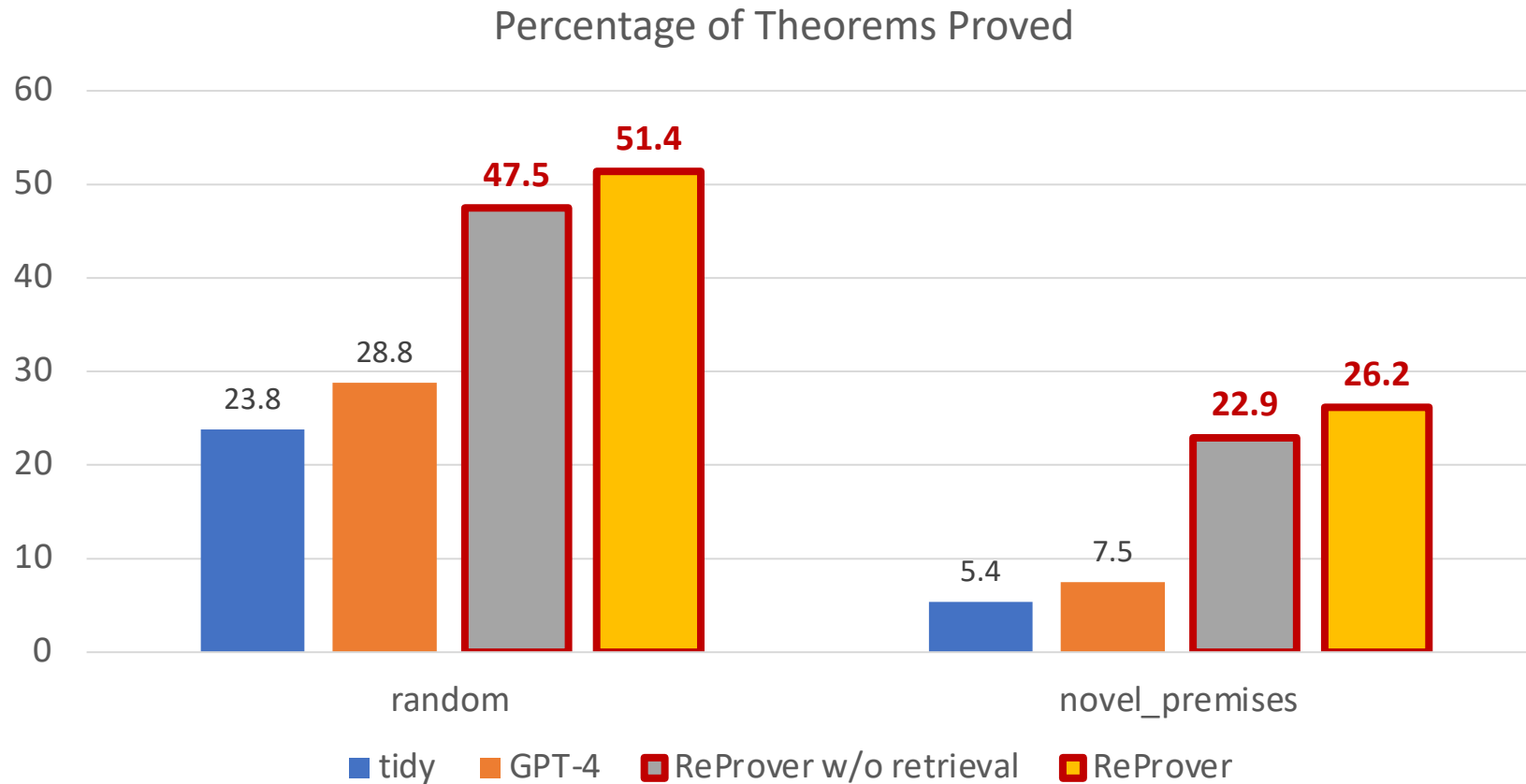
Prove theorems by Interaction



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

[Azerbaiyev 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

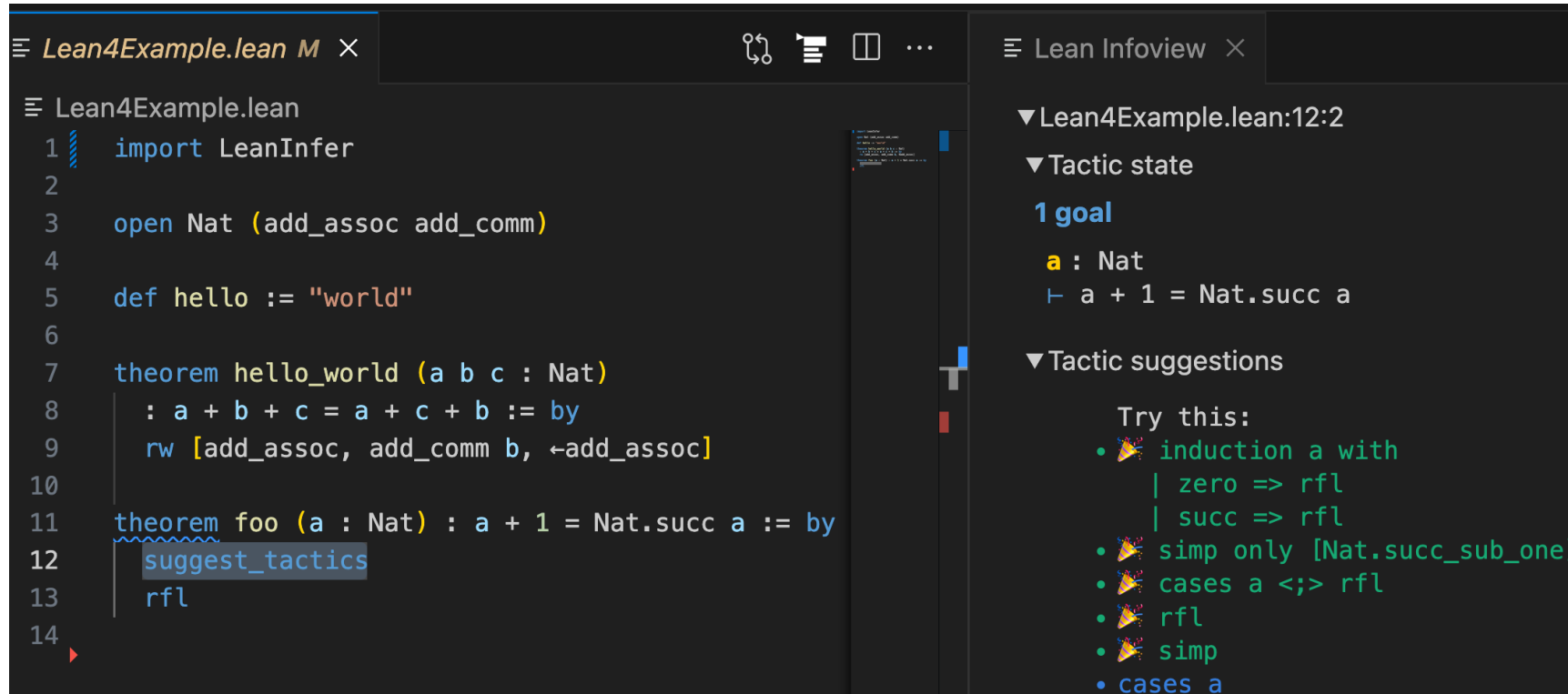
theorem exercise_11_2_13 (a b : ) :
  (of_int a : gaussian_int) of_int b → a b :=
begin
  contrapose,
  simp,
end

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



The screenshot shows the Lean IDE interface. The main editor displays the following Lean code:

```
1 import LeanInfer
2
3 open Nat (add_assoc add_comm)
4
5 def hello := "world"
6
7 theorem hello_world (a b c : Nat)
8   : a + b + c = a + c + b := by
9   rw [add_assoc, add_comm b, ←add_assoc]
10
11 theorem foo (a : Nat) : a + 1 = Nat.succ a := by
12   suggest_tactics
13   rfl
14
```

The right-hand pane, titled "Lean Infoview", shows the current state of the proof at line 12, column 2:

- ▼ Lean4Example.lean:12:2
- ▼ Tactic state
- 1 goal**
- a** : Nat
- ├ a + 1 = Nat.succ a
- ▼ Tactic suggestions
- Try this:
- induction a with
| zero => rfl
| succ => rfl
- simp only [Nat.succ_sub_one]
- cases a <;> rfl
- rfl
- simp
- cases a

[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

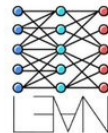
lean-dojo/ LeanDojo



Tool for data extraction and interacting with Lean programmatically.

7 Contributors, 2 Used by, 13 Discussions, 344 Stars, 41 Forks

lean-dojo/ LeanCopilot



Native Neural Network Inference in Lean

5 Contributors, 3 Issues, 1 Discussion, 67 Stars, 4 Forks

Hugging Face

Downloads last month
446



kaiyuy/leandojo-lean4-tacgen-byt5-small like 5

Text2Text Generation, Transformers, PyTorch, t5, Inference Endpoints, text-generation-inference, License: mit

Team



Aidan Swope



Alex Gu



Rahul Chalamala



Peiyang Song



Shixing Yu



Saad Godil

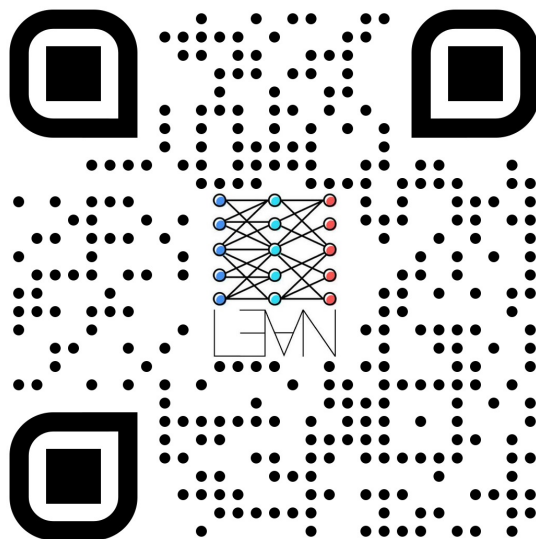


Ryan Prenger



Anima Anandkumar

LeanDojo



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

Prove theorems by Interaction

