A Few Open Problems in Neural Theorem Proving

(in Lean)

Sean Welleck

September 5, 2024

Carnegie Mellon University

Neural theorem proving

Use neural networks to:

- Generate proofs in an interactive proof assistant

Neural theorem proving | Rapid progress

Rapid progress in methods based on language models:

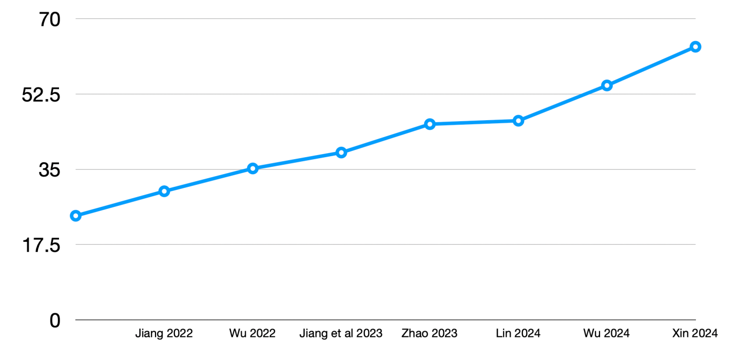


Figure 1: miniF2F benchmark performance, 2022-2024

Neural theorem proving | Rapid progress

```
theorem imo_1960_p2 (x : ℝ) (h₀ : 0 ≤ 1 + 2 * x) (h₁ : (1 - Real.sqrt (1 + 2 *
    x)) ^ 2 ≠ 0)
    (h₂ : 4 * x ^ 2 / (1 - Real.sqrt (1 + 2 * x)) ^ 2 < 2 * x + 9) : -(1 / 2)
   ≤ x ∧ x < 45 / 8 := by
  norm_num at h₀ h₁ h₂
  have h₃ : 0 ≤ 1 + 2 * x := by linarith
  have h₄ : 0 < 1 + Real.sqrt (1 + 2 * x) := by
    nlinarith [Real.sqrt_nonneg (1 + 2 * x)]
  have h₅ : 4 * x ^ 2 / (1 - Real.sqrt (1 + 2 * x)) ^ 2 < 2 * x + 9 := by
   linarith
  have h₆ : 1 - Real.sqrt (1 + 2 * x) ≠ 0 := by
    intro h
    apply h₁
    nlinarith
  have h₇ : 4 * x ^ 2 / (1 - Real.sqrt (1 + 2 * x)) ^ 2 = (1 + Real.sqrt (1 +
   2 * x)) ^ 2 := by
    field_simp [h₆]
    nlinarith [sq_sqrt (show 0 ≤ 1 + 2 * x by linarith)]
  rw [h₇] at h₅
  constructor <;> nlinarith [sq_sqrt (show 0 ≤ 1 + 2 * x by linarith)]
```

Figure 2: Generated International Math Olympiad solution in Lean (DeepSeek Prover-1.5B, Xin et al 2024)

Neural theorem proving | Lean

Why talk about Lean?

- Increasing interest from the mathematical community
- Increasing interest from the AI community
- For AI research, the choice of proof assistant matters (not ideal!)

This talk

3 open problems in neural theorem proving in Lean:

- Going beyond human data
- Going beyond competition problems
- Going beyond mathematics

1. Going beyond human data

Language model-based proving:

- **Train** a model $p_\theta(y|x)$ on a dataset $\mathcal{D} = \{(x, y)\}$, e.g.,
    - x: proof state
    - y: next tactic (next "step")
    - $\mathcal{D}$: extracted from human-written theorems and proofs

Language model-based proving:

- **Train** a model $p_\theta(y|x)$ on a dataset $\mathcal{D} = \{(x, y)\}$, e.g.,
  - x: proof state
  - y: next tactic (next “step”)
  - $\mathcal{D}$: extracted from human-written theorems and proofs
- **Generate** proofs:

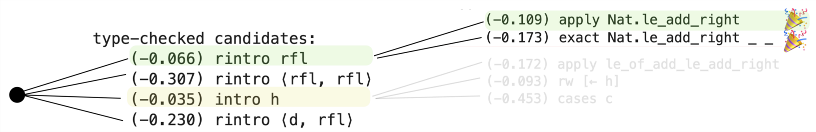


Figure 3: Best-first search

1. Going beyond human data

- Some models are already trained on $\approx$ all Lean projects!
  - E.g., Lean-GitHub [5]: data from 237 Lean 4 repos
- More human-written data will help, but difficult to scale[1]

[1] Please don't stop making more publicly available formal mathematics data!

1. Going beyond human data

Open problem I: how do we *synthesize* useful data?

- Proofs
- Theorems
- Augmentations (formal, informal, ...)
- ...

1. Going beyond human data

Not a new problem; common methods:

- Statement autoformalization [Wu et al 2022 [4]]
  - Informal theorem $\rightarrow$ formal theorem
- Expert iteration [Polu et al 2022 [3]]
  - Generate proofs with a model, train on successful ones, iterate

1. Going beyond human data

Not a new problem; common methods:

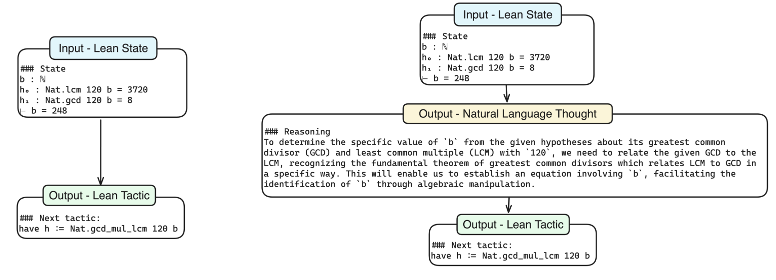
- Statement autoformalization [Wu et al 2022 [4]]
  - Informal theorem $\rightarrow$ formal theorem
- Expert iteration [Polu et al 2022 [3]]
  - Generate proofs with a model, train on successful ones, iterate

Used in several state-of-the-art methods, e.g. DeepSeek-Prover 1.5, AlphaProof

1. Going beyond human data | Lean-STaR

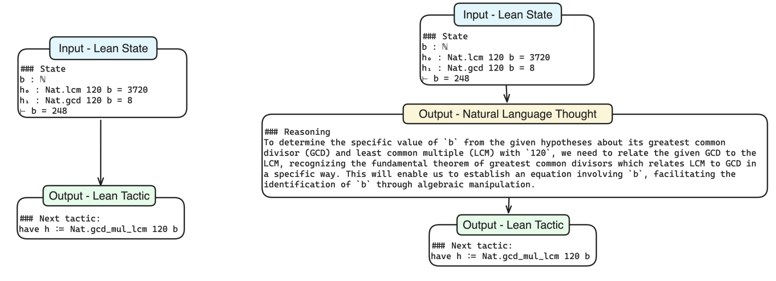
Lean-STaR: Learning to Interleave Thinking and Proving
Haohan Lin, Zhiqing Sun, Yiming Yang, Sean Welleck
https://arxiv.org/abs/2407.10040

1. Going beyond human data | Lean-STaR



Can we do better by interleaving *informal* steps of reasoning? (right)

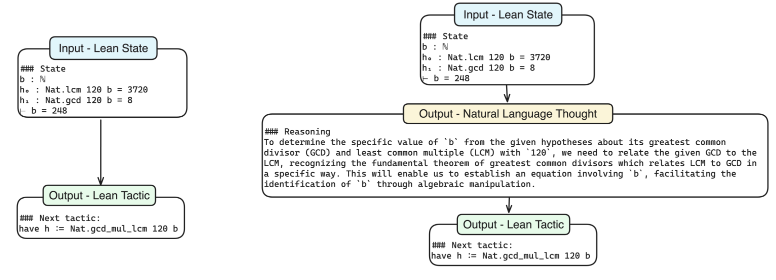
1. Going beyond human data | Lean-STaR



Why?

- Plan proof steps
- Diversify search space
- More tokens can give more computational capacity

1. Going beyond human data | Lean-STaR



Why?

- Plan proof steps
- Diversify search space
- More tokens can give more computational capacity

Data doesn't exist! We need to synthesize it.

Lean-STaR (Self-taught reasoner[2])

Step 1: generate an informal "thought" with an off-the-shelf language model retrospectively

- (state, tactic) $\rightarrow$ thought

Train an initial model on a dataset of such examples:

- $p_\theta^0(thought, tactic|state)$

[2] Inspired by *STaR: Bootstrapping Reasoning with Reasoning*, Zelikman et al 2022

Step 2: generate proofs with the model

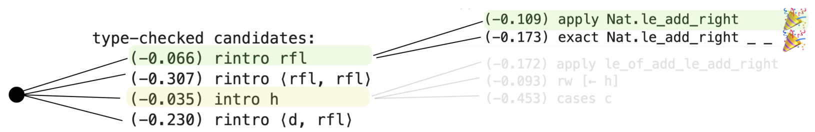


Figure 4: Best-first search: difficult to score (thought, tactic) candidates

Step 2: generate proofs with the model

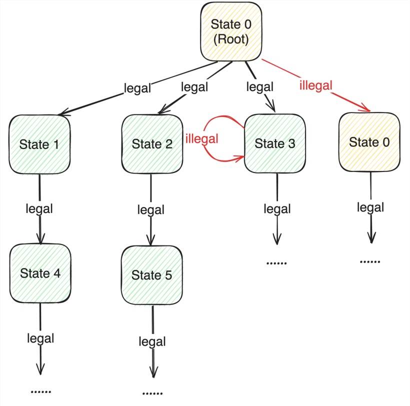


Figure 5: New sampling method

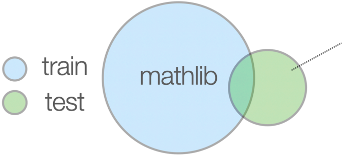
Step 3: train on the successful proofs, and repeat:[3]

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

[3] I.e. Expert Iteration [Polu et al 2022 [3]]

1. Going beyond human data | Lean-STaR

- **miniF2F** [7]: competition problems (AMC, AIME, IMO)



Problem **1959 IMO Problems/Problem 1**

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

↕

```
theorem imo_1959_p1
  (n : ℕ)
  (h₀ : 0 < n) :
  Nat.gcd (21*n + 4) (14*n + 3) = 1 := by sorry
```

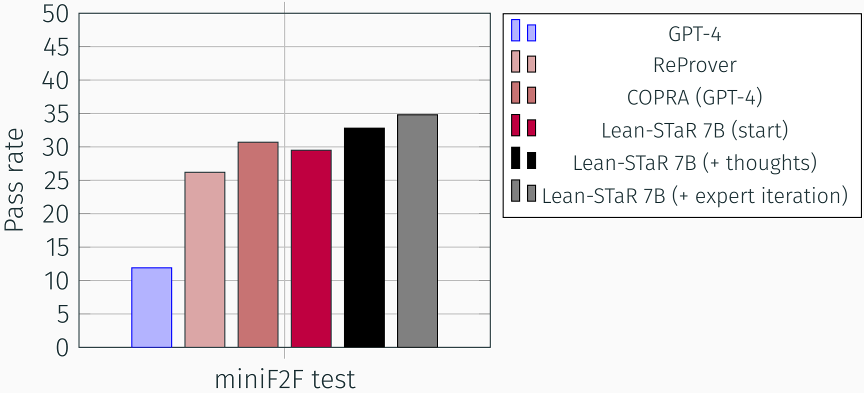


Figure 6: MiniF2F test

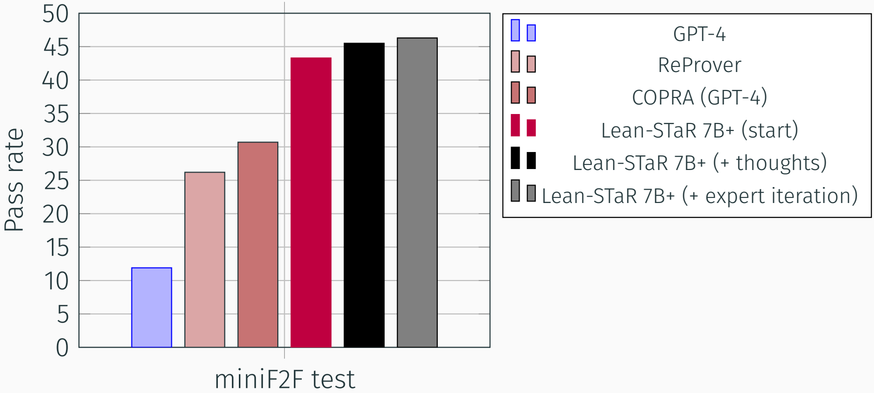


Figure 7: MiniF2F test

```
theorem amc12a_2002_p6 (n : ℕ) (h₀ : 0 < n)
    : ∃ m, m > n ∧ ∃ p, m * p ≤ m + p :=
    use n + 1, by linarith only [h₀] -- Reasoning: To demonstrate the existence of
    such an `m` and `p` that satisfy the given inequalities, we leverage the
    property of natural numbers that ensures an increase in `n` from `0` leads to
    the possibility of finding such `m` and `p`. This approach is supported by the
    fundamental property of natural numbers and their arithmetic operations,
    suggesting that for a given `n` greater than `0`, there exists a successor `m`
    and a number `p` that satisfy the conditions outlined.
    exact ⟨0, by simp ⟩ -- Reasoning: To demonstrate that there exists an \(n \in ℕ
    \) such that \((n + 1) * p \leq n + 1 + p\) holds for some \(p \in ℕ\), we
    recognize that setting \(p = 0\) satisfies the inequality by the properties of
    zero in arithmetic. This is because \(n + 1 > 0\) by assumption, ensuring the
    left-hand side is non-negative and the right-hand side simplifies to \(n +
    1\), making the inequality trivially true.
```

Figure 8: Example generated thoughts and proof from Lean-STaR

```
theorem aime_1983_p2 (x p : ℝ) (f : ℝ → ℝ) (h0 : 0 < p ∧ p < 15) (h1 : p ≤ x ∧ x
    ≤ 15)
    (h2 : f x = abs (x - p) + abs (x - 15) + abs (x - p - 15))
    : 15 ≤ f x :=
    cases' h1 with h1 h2 -- Reasoning: Given a function \(f\) defined over the real
    numbers, where \(f(x)\) equals the sum of the absolute values of the
    differences \(x - p\), \(x - 15\), and \(x - p - 15\), along with conditions
    on \(p\) and \(x\) that ensure \(x\) lies between \(p\) and \(15\) and \(p\)
    is positive, we aim to show that \(f(x) \geq 15\). To demonstrate this, we
    consider all possible cases for the position of \(x\) relative to \(p\) and
    \(15\), specifically at \(p\) itself, between \(p\) and \(15\), and at \(15\).
    By evaluating \(f(x)\) for each case, we can establish a lower bound of
    \(f(x)\) that is greater than or equal to \(15\).
    simp [h0, abs_of_nonneg (sub_nonneg.2 h1), abs_of_nonneg (sub_nonneg.2 h2), *]
    -- Reasoning: The objective is to show that \(f(x)\) is at least 15 for \(x\)
    within the bounds \(p \le x \le 15\), where \(p\) is positive but less than
    15. The provided expression for \(f(x)\) suggests using properties of absolute
    values and inequalities to simplify and analyze \(f(x)\). By applying
    simplification rules for absolute values and inequalities, along with the
    given bounds, we aim to demonstrate the desired inequality \(f(x) \ge 15\).
    simp only [abs_of_nonneg] at * -- Reasoning: The objective is to establish an
    inequality involving an expression with absolute values and a sum, where all
    terms are functions of \(x\). By recognizing that the absolute value functions
    are always non-negative, we can simplify the given expression by removing the
    absolute value symbols, thus focusing on the arithmetic manipulation of the
    terms to demonstrate the inequality.
    simp only [abs_of_nonneg, sub_add_eq_add_sub] at * -- Reasoning: The objective
    is to demonstrate that the expression `15 ≤ x - p + |x - 15| + |x - p - 15|`
    holds given the conditions on `p`, `x`, and the function `f`. Since the
    expression in question involves absolute values which inherently involve
    non-negativity, manipulating these expressions to highlight their
    non-negativity or equality is a logical initial step. This involves
    recognizing that the absolute value of a difference can be rewritten as the
    difference of absolute values, which simplifies the manipulation of the
    expression to prove the inequality.
    congr 1 -- Reasoning: To establish the inequality `15 ≤ x + |x - 15| + |x - p
    - 15| - p`, we recognize that the left-hand side of the inequality involves a
    mix of additions and subtractions of real numbers. By introducing a congruence
    transformation that simplifies operations involving absolute values, we aim to
    align both sides of the inequality in a form that directly reflects the
    properties of real numbers under these operations.
    simp only [abs_of_nonpos (sub_nonpos_of_le h2), abs_of_nonneg (sub_nonneg_of_le
    h1)] -- Reasoning: The objective is to prove an inequality involving absolute
    values and subtractions that aligns the given conditions and constraints.
    Simplification using the properties of absolute values and the provided bounds
    can directly lead to the desired conclusion by expressing the absolute values
    in terms of their underlying expressions and applying the given constraints.
    simp [abs_of_pos, *] -- Reasoning: To prove the given inequality involving
    absolute values, simplification using algebraic properties and given
    hypotheses will directly lead us to the desired inequality by recognizing the
    algebraic manipulation and application of the known bounds `h1` and `h2`, as
    well as the non-negativity of `p`.
    rw [abs_of_nonpos] -- Reasoning: The objective is to demonstrate that `15 ≤ 15
    + |x - p - 15| - p` under the given constraints on `p` and `x`. Given `p < 15`
    and `p ≤ x ≤ 15`, it's apparent that `p - 15` is non-positive. This
    observation allows us to apply the property that the absolute value of a
    non-positive number is equal to its negative value, thereby simplifying the
    expression on the right-hand side of the inequality.
    all_goals linarith -- Reasoning: The goal involves demonstrating a relationship
    between expressions involving inequalities and subtraction. The simplification
    of these expressions into a form that directly compares their numerical values
    can lead to a straightforward application of known inequalities and algebraic
    properties, demonstrating the inequality's validity under the given conditions.
```

Figure 9: Example generated thoughts and proof from Lean-STaR

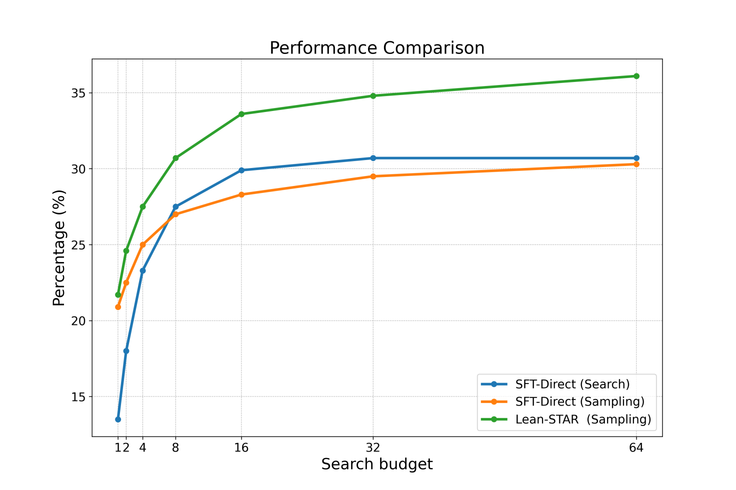


Figure 10: Increasing the search budget is more effective with thoughts

This talk

3 open problems in neural theorem proving in Lean:

- Going beyond human data
  - Synthesizing data: problems, proofs, plans, ...
- **Going beyond competition problems**
- Going beyond mathematics

2. Going beyond competition problems

Lots of exciting progress! Some methods can solve IMO problems!

However, not much impact on proving in practice.

2. Going beyond competition problems

Accessibility gap:

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

Accessibility gap:

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

However, there are model-agnostic tools available to plug into!

2. Going beyond competition problems

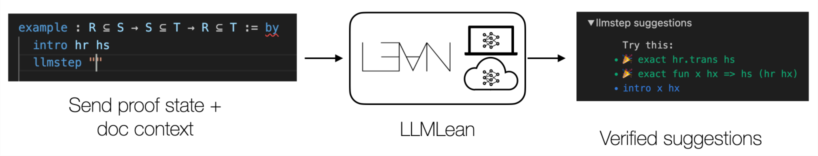


Figure 11: https://github.com/cmu-l3/llmlean

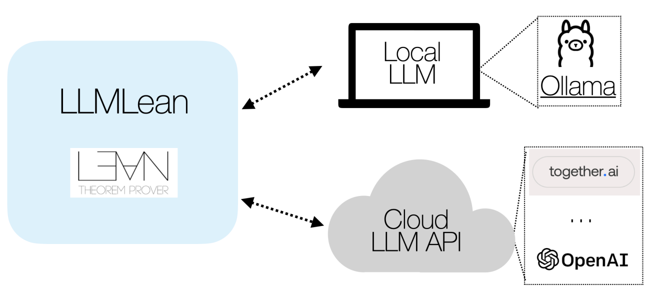


Figure 12: https://github.com/cmu-l3/llmlean

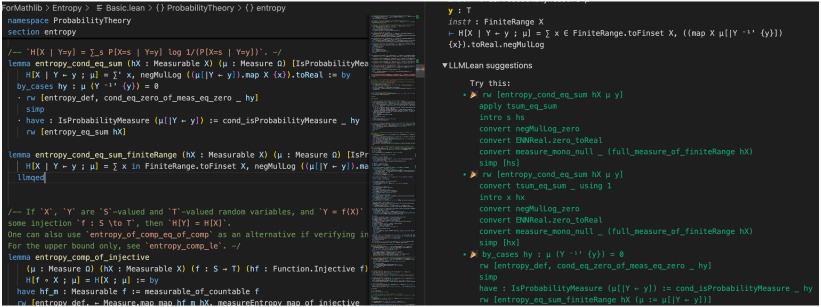


Figure 13: Example on Polynomial Freiman Rusza Conjecture project
https://github.com/cmu-l3/llmlean

Benchmarking gap:

- Benchmark improvements (e.g., on competition problems) do not measure improvement in real-world proving conditions

2. Going beyond competition problems

Figure 14: Interview questions ≠ real code development

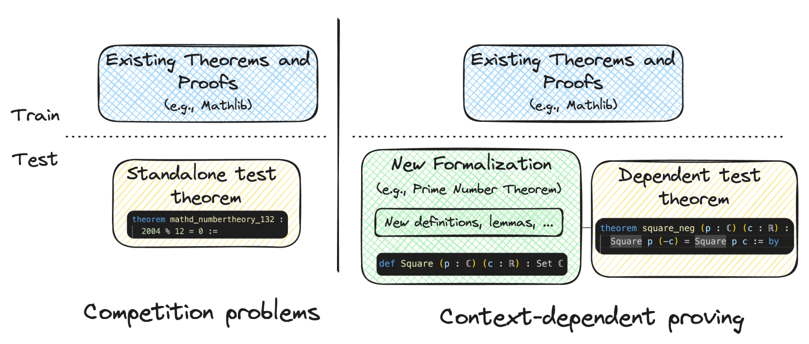


Figure 15: Competition problems $\neq$ real proof development

2. Going beyond competition problems

Real-world proving is **context-dependent**:

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

2. Going beyond competition problems

Generalization to new contexts is studied in other proof assistants, e.g., online setting[4], testing on held-out repositories[5]

Not a focus for state-of-the-art models/benchmarks in Lean!

[4] Tactician [2], Graph2Tac [1]
[5] CoqGym [6]

miniCTX: Neural Theorem Proving with (Long-)Contexts
Jiewen Hu, Thomas Zhu, Sean Welleck
https://www.arxiv.org/abs/2408.03350

miniCTX:

Collect (context, theorem) examples from real Lean projects:[6]

- “Future mathlib”: theorems added after a time cutoff
- Recent projects: PFR, PrimeNumberTheorem
- Textbook exercises: How To Prove It, Math 2001

[6] + tools for easily adding new projects: https://github.com/cmu-l3/ntp-toolkit

2. Going beyond competition problems | miniCTX

miniCTX:

Collect (context, theorem) examples from real Lean projects:[6]

- "Future mathlib": theorems added after a time cutoff
- Recent projects: PFR, PrimeNumberTheorem
- Textbook exercises: How To Prove It, Math 2001

Goal: generalize to new theorems/contexts/repositories

[6] + tools for easily adding new projects: https://github.com/cmu-l3/ntp-toolkit

Context:

- Preceding code in the file
- All accessible premises
- Repository metadata (to recover any other code)

Does context actually matter? A simple experiment.

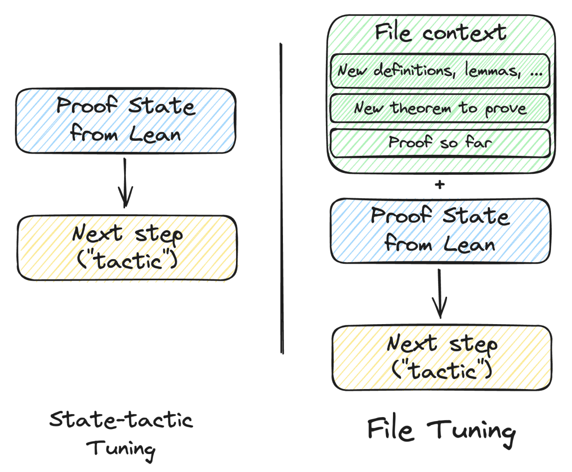


Figure 16: "File tuning": train on (preceding code, state, next-tactic) examples

2. Going beyond competition problems | miniCTX

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

	MiniF2F	MiniCTX				
Models	**Test**	**Prime**	**PFR**	**Mathlib**	**HTPI**	**Avg.**
GPT-4o (full proof)	-	1.15%	5.56%	2.00%	9.73%	5.59%
GPT-4o (+ context)	-	13.79%	1.85%	18.00%	31.89%	22.07%
State-tactic prompting	28.28%	19.54%	5.56%	16.00%	19.15%	20.61%
State-tactic tuning	32.79%	11.49%	5.56%	22.00%	5.95%	9.31%
File tuning	**33.61%**	**32.18%**	**5.56%**	**34.00%**	**38.38%**	**31.65%**

File-tuned model is deployed in LLMLean:

LLM on your laptop:

1. Install ollama.
2. Pull a language model:

```
ollama pull wellecks/ntpctx-llama3-8b
```

Figure 17: https://github.com/cmu-l3/llmlean

2. Going beyond competition problems | deployment

Several open-source artifacts:

- Data/models: `https://huggingface.co/l3lab`
- Data extraction: `https://github.com/cmu-l3/ntp-toolkit`
- Evaluation: `https://github.com/cmu-l3/minictx-eval`

Many approaches to explore in the future:

- "File tuning": context is preceding code
- Premise selection: context is a set of definitions and theorems
- Full repo: context is all other code in the repository
- ...

2. Going beyond competition problems

Many other potential tools beyond proof completion!

This talk

3 open problems in neural theorem proving in Lean:

- Going beyond human data
  - Synthesizing data
- Going beyond competition problems
  - Have actual tools as a goal
- Going beyond mathematics

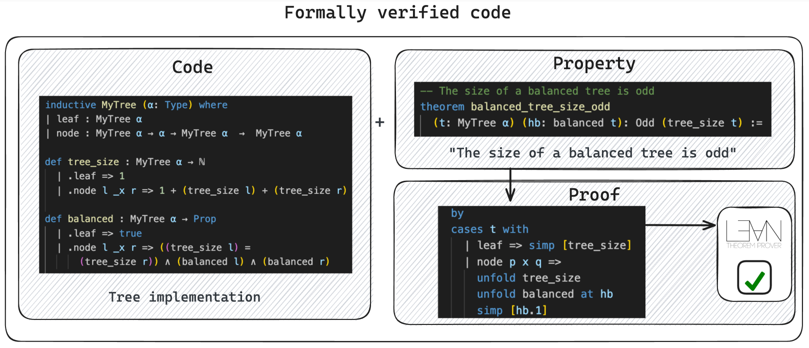
miniCodeProps: a Minimal Benchmark for Proving Code Properties
Evan Lohn, Sean Welleck

https://arxiv.org/abs/2406.11915

3. Going beyond mathematics

Interactive theorem provers

- Mathematics:
  - Math as code
  - Guarantees on proof correctness
- Code:
  - Prove properties of code


Formally verified code
Code
inductive MyTree (α: Type) where
| leaf : MyTree α
| node : MyTree α → α → MyTree α → MyTree α
def tree_size : MyTree α → ℕ
| .leaf => 1
| .node l _x r => 1 + (tree_size l) + (tree_size r)
def balanced : MyTree α → Prop
| .leaf => true
| .node l _x r => ((tree_size l) =
(tree_size r)) ∧ (balanced l) ∧ (balanced r)
Tree implementation
+
Property
-- The size of a balanced tree is odd
theorem balanced_tree_size_odd
(t: MyTree α) (hb: balanced t): Odd (tree_size t) :=
"The size of a balanced tree is odd"
Proof
by
cases t with
| leaf => simp [tree_size]
| node p x q =>
unfold tree_size
unfold balanced at hb
simp [hb.1]
LEAN
THEOREM PROVER

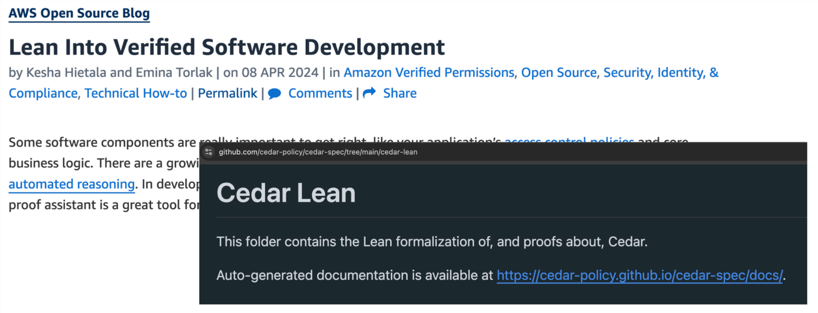


Figure 18: https://aws.amazon.com/blogs/opensource/lean-into-verified-software-development/

3. Going beyond mathematics

AI/neural theorem proving for program verification is actively studied in other proof assistants, such as Coq and Isabelle.

Not in Lean!

3. Going beyond mathematics

Our question:

- What is the simplest program verification scenario that:
    - Is a subproblem of the full 'verification problem'
    - Breaks current neural theorem proving methods

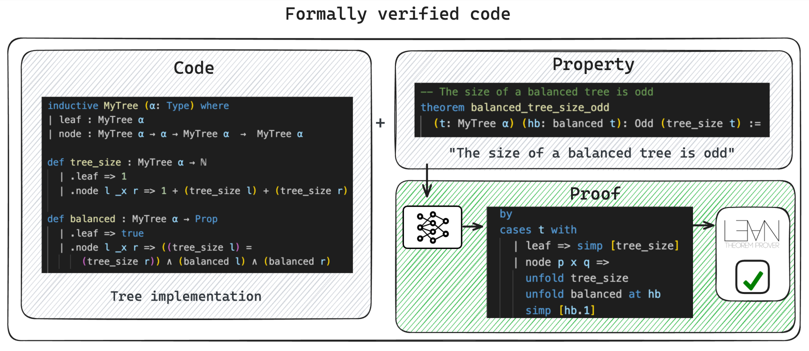
3. Going beyond mathematics

Our question:

- What is the simplest program verification scenario that:
    - Is a subproblem of the full ‘verification problem’
    - Breaks current neural theorem proving methods

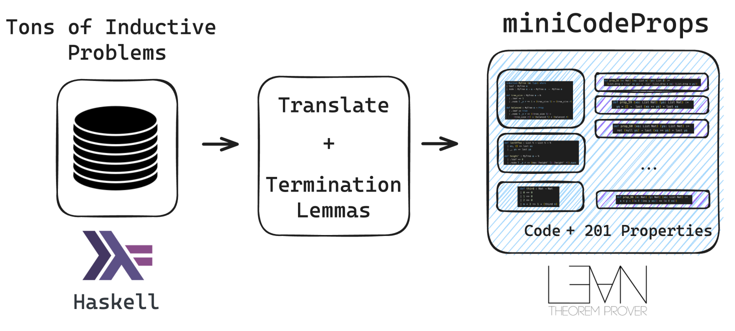
“Simple”:

- Self-contained, no complex dependencies
- Relatively small (fast, cheap evaluation)



Subproblem: theorem proving! Given (code, property), generate proof

Code blocks and 201 properties from *Tons of Inductive Problems*[7], translated from Haskell to Lean.



[7] https://tip-org.github.io/, Claessen et al 2015

MiniCodeProps

- Implementation + properties about lists, trees, and heaps
- Classified into difficulties:
  - Easy: Data structure properties
  - Medium: Termination properties
  - Hard: Sorting algorithm properties

Evaluation:

- Given property and all dependent code, generate a proof

Models:

- GPT-4o: generate full proof, 32 attempts + 1 round of refinement
- ntp-ctx: generate a proof via best-first search

https://github.com/cmu-l3/minicodeprops-eval

Model	Easy	Medium & Hard	Overall
GPT-4o (32 samples)	75.6% (65/86)	4.34% (5/115)	34.8% (70/201)
+ refinement	77.9% (67/86)	6.96% (8/115)	37.3% (75/201)
ntp-context-1.3B	72.1% (62/86)	8.69% (10/115)	35.8% (72/201)

Figure 19: Baselines perform well on easy properties

Model	Easy	Medium & Hard	Overall
GPT-4o (32 samples)	75.6% (65/86)	4.34% (5/115)	34.8% (70/201)
+ refinement	77.9% (67/86)	6.96% (8/115)	37.3% (75/201)
ntp-context-1.3B	72.1% (62/86)	8.69% (10/115)	35.8% (72/201)

Figure 20: Poor performance on medium/hard properties

```
import Mathlib

def butlast : List α → List α
  | [] => []
  | [_x] => []
  | x::xs => x::(butlast xs)

def butlastConcat : List α → List α → List α
  | xs, [] => butlast xs
  | xs, ys => xs ++ butlast ys

theorem prop_49 (xs: List Nat) (ys: List Nat) :
  (butlast (xs ++ ys) = butlastConcat xs ys) := by
  induction ys generalizing xs
  case nil =>
    simp [butlast, butlastConcat]
  case cons y ys ih =>
    simp [butlast, butlastConcat]
    induction xs
    case nil =>
      simp [butlast, butlastConcat]
    case cons x xs ih' =>
      simp [butlast, butlastConcat, List.cons_append, ih']
```

Figure 21: Successful proof (GPT-4o)

```
import Mathlib

def filter : List Nat → (Nat → Bool) → List Nat
| [], _f => []
| x::xs, f => if f x then x::(filter xs f) else (filter xs f)

theorem qsort_term2 (x:Nat) (xs: List Nat) :
List.length (filter xs fun y => decide (y > x)) < Nat.succ (List.length xs) := by
  induction xs with
  | nil =>
    simp [filter, Nat.zero_lt_succ]
  | cons y ys ih =>
    simp only [filter]
    split_ifs with h
    · simp only [List.length]
      exact Nat.succ_lt_succ ih
    · simp only [List.length]
      exact Nat.lt_succ_of_lt ih
```

Figure 22: Successful proof (GPT-4o)

```
def pairwise : List (List Nat) → List (List Nat)
| xs::ys::xss => lmerge xs ys :: pairwise xss
| xss => xss

def pairwise_len : List (List Nat) → Nat
| xs => if (Odd xs.length) then xs.length/2 + 1 else xs.length/2

lemma len_pairwise (xs: List (List Nat)): 2 * (pairwise xs).length = (if (Odd xs.length) then xs.length + 1 else xs.length) := by
  generalize hxl : xs.length = xl;
  split_ifs with h1;
  case pos;
    induction xl using Nat.strong_induction_on generalizing xs with
    | h xls ih =>
        rw [← hxl] at h1;
        cases xs with
        | nil => simp at h1;
        | cons head2 tail2 =>
          cases tail2 with
          | nil => unfold pairwise; simp at hxl; rw [←hxl]; simp;
          | cons head3 tail3 =>
            unfold pairwise;
            rw [←hxl, List.length_cons, List.length_cons, Nat.succ_eq_add_one _, Nat.succ_eq_add_one _]
            ring_nf;
            simp;
            have hodd: Odd tail3.length := by
              rw [List.length_cons, List.length_cons, Nat.succ_eq_add_one _, Nat.add_assoc, Nat.odd_add] at h1;
              apply h1.2; simp;
            have tmp := ih tail3.length (by rw [←hxl]; simp; linarith;) tail3 rfl hodd
            ring_nf at tmp;
            rw [tmp];
            ring_nf;
  case neg;
    induction xl using Nat.strong_induction_on generalizing xs with
    | h xls ih =>
        rw [← hxl] at h1;
        cases xs with
        | nil => unfold pairwise; simp [← hxl]
        | cons head2 tail2 =>
          cases tail2 with
          | nil => simp at h1;
          | cons head3 tail3 =>
            unfold pairwise;
            rw [←hxl, List.length_cons, List.length_cons, List.length_cons, Nat.succ_eq_add_one _, Nat.succ_eq_add_one _];
            ring_nf;
            simp;
            simp at h1;
            have heven: Even tail3.length := by
              rw [Nat.add_assoc, Nat.even_add] at h1;
              apply h1.2; simp;
            simp at ih;
            have tmp := ih tail3.length (by rw [←hxl]; simp; linarith;) tail3 rfl heven
            ring_nf at tmp;
            exact tmp;
```


Figure 23: Human-written proof showing potential length of proofs

This talk

3 open problems in neural theorem proving in Lean:

- Going beyond human data
  - Synthesizing data: problems, proofs, plans, ...
- Going beyond competition problems
  - Have actual tools as a goal
- Going beyond mathematics
  - Program verification

Thank you!

Haohan Lin (Tsinghua)
Evan Lohn (CMU)
Jiewen Hu (CMU)
Zhiqing Sun (CMU)
Yiming Yang (CMU)
Thomas Zhu (CMU)

Lean-STaR: Learning to Interleave Thinking and Proving.
Haohan Lin, Zhiqing Sun, Yiming Yang, Sean Welleck, 2024.

miniCTX: Neural Theorem Proving with (Long-)Contexts.
Jiewen Hu, Thomas Zhu, Sean Welleck, 2024.

miniCodeProps: a Minimal Benchmark for Proving Code Properties.
Evan Lohn, Sean Welleck, 2024.

Sean Welleck (CMU)
Learning, Language, and Logic (L3) Lab

References i

- L. Blaauwbroek, M. Olšák, J. Rute, F. I. S. Massolo, J. Piepenbrock, and V. Pestun.
  Graph2tac: Online representation learning of formal math concepts.
  In *Forty-first International Conference on Machine Learning*, 2024.
- L. Blaauwbroek, J. Urban, and H. Geuvers.
  The Tactician: A Seamless, Interactive Tactic Learner and Prover for Coq, page 271–277.
  Springer International Publishing, 2020.

- S. Polu, J. M. Han, K. Zheng, M. Baksys, I. Babuschkin, and I. Sutskever.
  Formal mathematics statement curriculum learning.
  In *The Eleventh International Conference on Learning Representations*, 2023.
- Y. Wu, A. Q. Jiang, W. Li, M. N. Rabe, C. E. Staats, M. Jamnik, and C. Szegedy.
  Autoformalization with large language models.
  In A. H. Oh, A. Agarwal, D. Belgrave, and K. Cho, editors, *Advances in Neural Information Processing Systems*, 2022.
- Z. Wu, J. Wang, D. Lin, and K. Chen.
  Lean-github: Compiling github lean repositories for a versatile lean prover, 2024.

K. Yang and J. Deng.
Learning to prove theorems via interacting with proof assistants.
ArXiv, abs/1905.09381, 2019.

K. Zheng, J. M. Han, and S. Polu.
minif2f: a cross-system benchmark for formal olympiad-level mathematics.
In *International Conference on Learning Representations*, 2022.