Carnegie Mellon University

miniCTX: Neural Theorem Proving with (Long-)Contexts

Jiewen Hu, Thomas Zhu, Sean Welleck

Informal Math

Math in Natural language:

- Intuitive
- Readable
- Flexible

Problem

A calculator is broken so that the only keys that still work are the $\sin,\ \cos, \tan,\ \sin^{-1},\ \cos^{-1},$ and $\tan^{-1}$ buttons. The display initially shows 0. Given any positive rational number q, show that pressing some finite sequence of buttons will yield q. Assume that the calculator does real number calculations with infinite precision. All functions are in terms of radians.

Solution

We will prove the following, stronger statement : If m and n are relatively prime nonnegative integers such that $n > 0$, then the some finite sequence of buttons will yield $\sqrt{m/n}$.

To prove this statement, we induct strongly on $m+n$. For our base case, $m+n=1$, we have $n=1$ and $m=0$, and $\sqrt{m/n}=0$, which is initially shown on the screen. For the inductive step, we consider separately the cases $m=0, 0<m\leq n$, and $n<m$.

If $m=0$, then $n=1$, and we have the base case.

If $0<m\leq n$, then by inductive hypothesis, $\sqrt{(n-m)/m}$ can be obtained in finitely many steps; then so can

$$\cos\tan^{-1}\sqrt{(n-m)/m}=\sqrt{m/n}.$$

If $n<m$, then by the previous case, $\sqrt{n/m}$ can be obtained in finitely many steps. Since $\cos\tan^{-1}\sqrt{n/m}=\sin\tan^{-1}\sqrt{m/n}$, it follows that

$$\tan\sin^{-1}\cos\tan^{-1}\sqrt{n/m}=\sqrt{m/n}$$

can be obtained in finitely many steps. Thus the induction is complete. ■

Carnegie Mellon University

Informal Math

Math in Natural language:

- Intuitive
- Readable
- Flexible

- Ambiguous
- Hard to verify

Is $p(x)$ a function or a value?

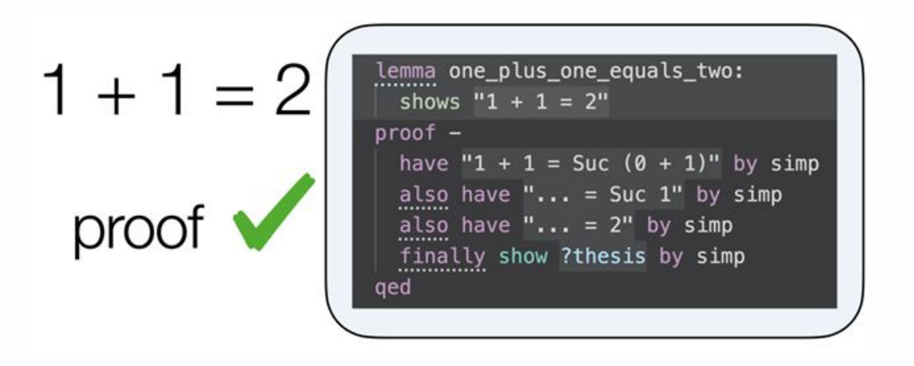
In $\mathbb{E}_{x\sim p(x)}[f(x)]$, $p(x)$ is treated as a distribution

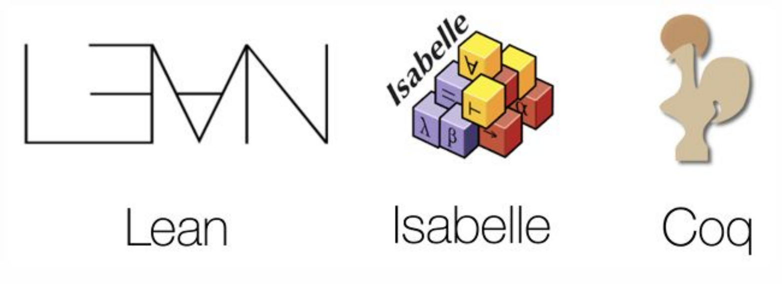
But in $p(x) = \frac{1}{\sqrt{2\pi\sigma^2}} e^{-\frac{(x-\mu)^2}{2\sigma^2}}$, $p(x)$ is a value

Formal Math

Math as **Source Code**

- Verbose
- Precise
- Verifiable
- Automatable





Carnegie Mellon University

Formal Math & Fields Medalist

1. The challenge

I want to propose a challenge: Formalize the proof of the following theorem.

Theorem 1.1 *(Clausen-S.) Let* $0 < p' < p \leq 1$ *be real numbers, let* S *be a profinite set, and let* V *be a* p*-Banach space. Let* $\mathcal{M}_{p'}(S)$ *be the space of* p'*-measures on* S*. Then*

$$\mathrm{Ext}^i_{\mathrm{Cond(Ab)}}(\mathcal{M}_{p'}(S), V) = 0$$

for $i \geq 1$.

— with this theorem, the hope that the condensed formalism can be fruitfully applied to real functional analysis stands or falls. I think the theorem is of utmost foundational importance, so being 99.9% sure is not enough.

Liquid Tensor Experiment posted by
Peter Scholze (December 2020)

Terence Tao
@tao@mathstodon.xyz

Finished formalizing in #Lean4 the proof of an actual new theorem (Theorem 1.3) in my recent paper arxiv.org/abs/2310.05328 :

Terence Tao's Lean formalization project (October 2023)

Carnegie Mellon University

Formal Math & ML

February 2, 2022 Milestone

Solving (some) formal math olympiad problems

Read paper ↗

OpenAI

OpenAI (2022)

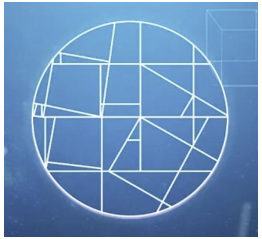
RESEARCH

AI achieves silver-medal standard solving International Mathematical Olympiad problems

25 JULY 2024

AlphaProof and AlphaGeometry teams

DeepMind

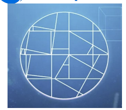


AlphaProof (2024)

Benchmarking Gap

Highschool competitions (AMC / AIME / IMO)

DeepMind



OpenAI

Theorem to prove

```
theorem imo_1964_p1_2 (n : ℕ) : ¬7 ∣ 2 ^ n + 1
```

- Self-contained
- Uses basic math techniques
- Limited domain (algebra, number theory)

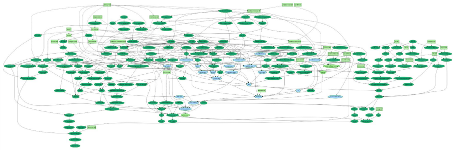
Real projects

New lemmas & definitions

```
/-- The real-valued version of a measure. Maps infinite measure sets to zero. Use as `μ.real s`. -/
protected def Measure.real (s : Set α) : ℝ := (μ s).toReal
```

```
lemma IsUniform.measureReal_preimage_sub_zero (Uunif : IsUniform A U) (Umeas : Measurable U)
    (Vunif : IsUniform B V) (Vmeas : Measurable V) (h_indep : IndepFun U V) :
    (ℙ : Measure Ω).real ((U - V) ⁻¹' {0})
     = Nat.card (A ∩ B : Set G) / (Nat.card A * Nat.card B) := by
```

```
lemma sum_mul_log_div_leq {a b : ι → ℝ} (ha : ∀ i ∈ s, 0 ≤ a i) (hb : ∀ i ∈ s, 0 ≤ b i)
    (habs : ∀ i ∈ s, b i = 0 → a i = 0) :
    (∑ i ∈ s, a i) * log ((∑ i ∈ s, a i) / (∑ i ∈ s, b i)) ≤ ∑ i ∈ s, a i * log (a i / b i) := by
```



Theorem to prove

```
/-- The polynomial Freiman-Ruzsa (PFR) conjecture: if `A` is a subset of an elementary abelian
2-group of doubling constant at most `K`, then `A` can be covered by at most `2 * K ^ 12` cosets of
a subgroup of cardinality at most `|A|`. -/
theorem PFR_conjecture (h₀A : A.Nonempty) (hA : Nat.card (A + A) ≤ K * Nat.card A) :
    ∃ (H : Submodule (ZMod 2) G) (c : Set G),
     Nat.card c < 2 * K ^ 12 ∧ Nat.card H ≤ Nat.card A ∧ A ⊆ c + H := by
```

- Part of a project
- Uses unseen lemmas
- Various domains

Carnegie Mellon University

Benchmarking Gap

Highschool competitions (AMC / AIME / IMO)

DeepMind

OpenAI

Theorem to prove

```
theorem imo_1964_p1_2 (n : ℕ) : ¬7 ∣ 2 ^ n + 1
```

- Self-contained
- Uses basic math techniques
- Limited domain (algebra, number theory)

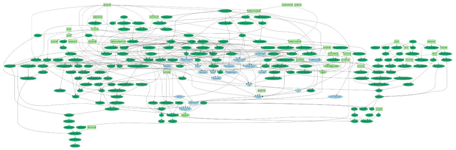
Real projects

New lemmas & definitions

```
/-- The real-valued version of a measure. Maps infinite measure sets to zero. Use as `μ.real s`. -/
protected def Measure.real (s : Set α) : ℝ := (μ s).toReal
```

```
lemma IsUniform.measureReal_preimage_sub_zero (Uunif : IsUniform A U) (Umeas : Measurable U)
    (Vunif : IsUniform B V) (Vmeas : Measurable V) (h_indep : IndepFun U V) :
    (ℙ : Measure Ω).real ((U - V) ⁻¹' {0})
     = Nat.card (A ∩ B : Set G) / (Nat.card A * Nat.card B) := by
```

```
lemma sum_mul_log_div_leq {a b : ι → ℝ} (ha : ∀ i ∈ s, 0 ≤ a i) (hb : ∀ i ∈ s, 0 ≤ b i)
    (habs : ∀ i ∈ s, b i = 0 → a i = 0) :
    (∑ i ∈ s, a i) * log ((∑ i ∈ s, a i) / (∑ i ∈ s, b i)) ≤ ∑ i ∈ s, a i * log (a i / b i) := by
```



Theorem to prove

```
/-- The polynomial Freiman-Ruzsa (PFR) conjecture: if `A` is a subset of an elementary abelian
2-group of doubling constant at most `K`, then `A` can be covered by at most `2 * K ^ 12` cosets of
a subgroup of cardinality at most `|A|`. -/
theorem PFR_conjecture (h₀A : A.Nonempty) (hA : Nat.card (A + A) ≤ K * Nat.card A) :
    ∃ (H : Submodule (ZMod 2) G) (c : Set G),
     Nat.card c < 2 * K ^ 12 ∧ Nat.card H ≤ Nat.card A ∧ A ⊆ c + H := by
```

- Part of a project
- Uses unseen lemmas
- Various domains

Carnegie Mellon University

Benchmarking Gap

Highschool competitions
(AMC / AIME / IMO)

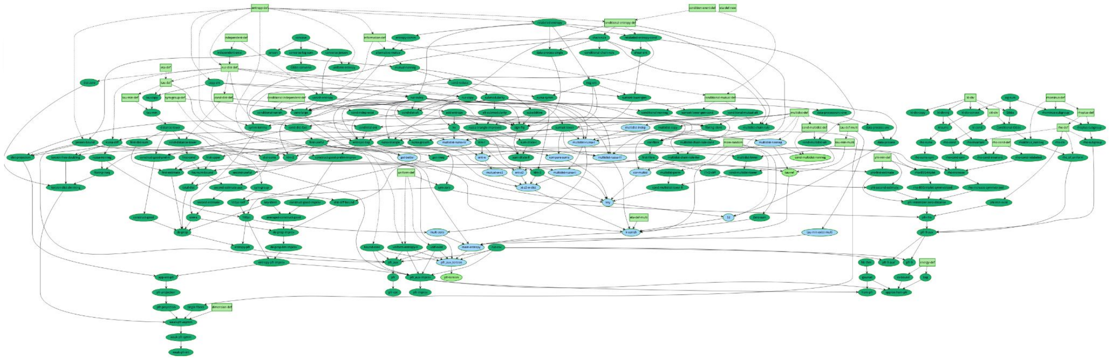
DeepMind OpenAI

Theorem to prove

```
theorem imo_1964_p1_2 (n : ℕ) : ¬7 ∣ 2 ^ n + 1
```

- Self-contained
- Uses basic math techniques
- Limited domain (algebra, number theory)

Real projects



- Part of a project
- Uses unseen lemmas
- Various domains

Benchmarking Gap

Highschool competitions (AMC / AIME / IMO)

DeepMind

OpenAI

Theorem to prove

```
theorem imo_1964_p1_2 (n : ℕ) : ¬7 ∣ 2 ^ n + 1
```

- Self-contained
- Uses basic math techniques
- Limited domain (algebra, number theory)

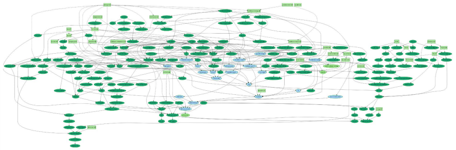
Real projects

New lemmas & definitions

```
/-- The real-valued version of a measure. Maps infinite measure sets to zero. Use as `μ.real s`. -/
protected def Measure.real (s : Set α) : ℝ := (μ s).toReal

lemma IsUniform.measureReal_preimage_sub_zero (Uunif : IsUniform A U) (Umeas : Measurable U)
    (Vunif : IsUniform B V) (Vmeas : Measurable V) (h_indep : IndepFun U V) :
    (ℙ : Measure Ω).real ((U - V) ⁻¹' {0})
      = Nat.card (A ∩ B : Set G) / (Nat.card A * Nat.card B) := by

lemma sum_mul_log_div_leq {a b : ι → ℝ} (ha : ∀ i ∈ s, 0 ≤ a i) (hb : ∀ i ∈ s, 0 ≤ b i)
    (habs : ∀ i ∈ s, b i = 0 → a i = 0) :
    (∑ i ∈ s, a i) * log ((∑ i ∈ s, a i) / (∑ i ∈ s, b i)) ≤ ∑ i ∈ s, a i * log (a i / b i) := by
```



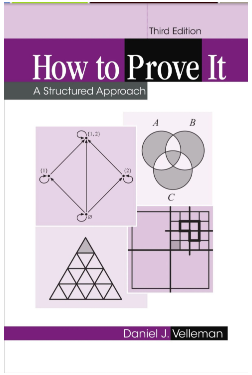
Theorem to prove

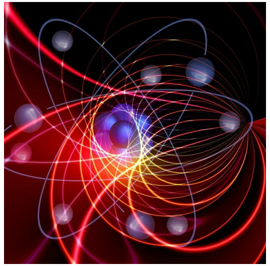
```
/-- The polynomial Freiman-Ruzsa (PFR) conjecture: if `A` is a subset of an elementary abelian
2-group of doubling constant at most `K`, then `A` can be covered by at most `2 * K ^ 12` cosets of
a subgroup of cardinality at most `|A|`. -/
theorem PFR_conjecture (h₀A : A.Nonempty) (hA : Nat.card (A + A) ≤ K * Nat.card A) :
    ∃ (H : Submodule (ZMod 2) G) (c : Set G),
     Nat.card c < 2 * K ^ 12 ∧ Nat.card H ≤ Nat.card A ∧ A ⊆ c + H := by
```

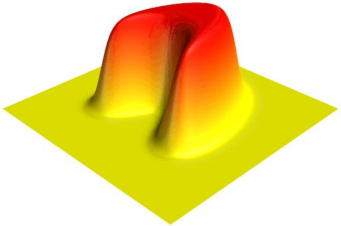
- Part of a project
- Uses unseen lemmas
- Various domains

Carnegie Mellon University

Testing on Real Projects







Lean Library
Mathlib 4

Math Projects
Prime Number Theorem,
Polynomial Freiman-Ruzsa

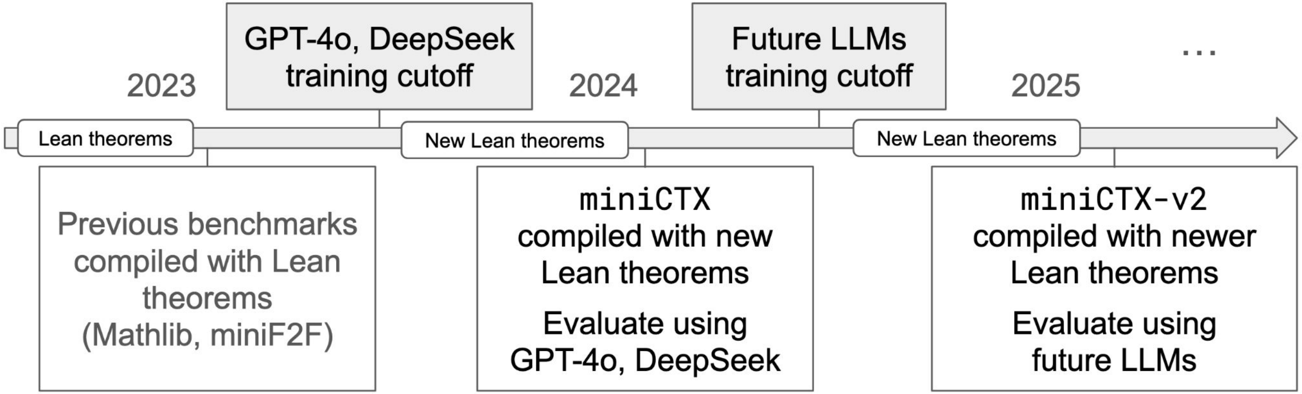
Math Textbook
How to Prove It

Physics
HepLean

Scientific Computing
SciLean

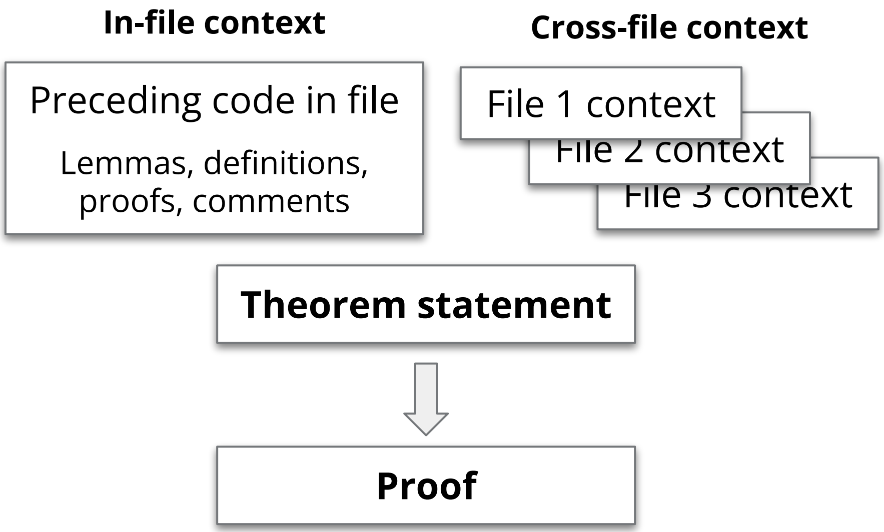
Avoid Data Contamination

We maintain benchmark integrity by using most recent theorems



Problem Formulation

- Task: (**theorem**, **context**) → **proof**



miniCTX Benchmark

Data splits:

- Prime Number Theorem +
- **Polynomial Freiman–Ruzsa**
- Mathlib
- How to Prove It
- High Energy Physics
- Scientific Computing

miniCTX Benchmark

Data splits:

- Prime Number Theorem +
- **Polynomial Freiman–Ruzsa**
- Mathlib
- How to Prove It
- High Energy Physics
- Scientific Computing

Polynomial Freiman-Rusza:

Valid

- Theorem 1
- **Theorem 2**

...

- Theorem 50

Test

- Theorem 1
- Theorem 2

...

- Theorem 50

Carnegie Mellon University

miniCTX Benchmark

Data splits:

- Prime Number Theorem +
- **Polynomial Freiman–Ruzsa**
- Mathlib
- How to Prove It
- High Energy Physics
- Scientific Computing

Polynomial Freiman-Rusza:

Valid
- Theorem 1
- **Theorem 2**

...
- Theorem 50

Test
- Theorem 1
- Theorem 2

...
- Theorem 50

In-file context: Preceding code

```
import Mathlib.Probability.Kernel.Composition.Comp
import PFR.Mathlib.Probability.Kernel.Disintegration

open Function MeasureTheory Real
open scoped ENNReal NNReal Topology ProbabilityTheory

namespace ProbabilityTheory.Kernel

section
variable {α β γ δ ε : Type*} {_ : MeasurableSpace α} {_ : MeasurableSpace β}
    {_ : MeasurableSpace γ} {_ : MeasurableSpace δ} {_ : MeasurableSpace ε}

variable {Ω S T : Type*} [mΩ : MeasurableSpace Ω]
  [MeasurableSpace S] [MeasurableSpace T] [MeasurableSpace γ]
  {κ : Kernel T S} {μ : Measure T} {X : Ω → S} {Y : Ω → β}

lemma map_map (κ : Kernel α β) {f : β → γ} (hf : Measurable f) {g : γ → δ} (hg : Measurable g) :
    map (map κ f) g = map κ (g ∘ f) := by
  ext x s _
  rw [map_apply _ hg, map_apply _ hf, map_apply _ (hg.comp hf), Measure.map_map hg hf]

⋮

 @[simp]
 lemma fst_deleteRight (κ : Kernel α (β × γ × δ)) : fst (deleteRight κ) = fst κ := by
   rw [deleteRight_eq, fst_map_prod, fst_eq]
   exact measurable_fst.comp measurable_snd
```

Cross-file context: Lemmas & definitions from other files

```
/-- The real-valued version of a measure. Maps infinite measure sets to zero. Use as `μ.real s`. -/
protected def Measure.real (s : Set α) : ℝ := (μ s).toReal
```

```
/-- Given two independent random variables `U` and `V` uniformly distributed respectively on `A`
and `B`, then `U = V` with probability `#(A ∩ B) / #A · #B`. -/
lemma IsUniform.measureReal_preimage_sub_zero (Uunif : IsUniform A U) (Umeas : Measurable U)
    (Vunif : IsUniform B V) (Vmeas : Measurable V) (h_indep : IndepFun U V) :
    (ℙ : Measure Ω).real ((U - V) ⁻¹' {0})
      = Nat.card (A ∩ B : Set G) / (Nat.card A * Nat.card B) := by
```

```
/-- If $S$ is a finite set, and $a_s$, $b_s$ are non-negative for $s\in S$, then
  $$\sum_{s\in S} a_s \log\frac{a_s}{b_s}\ge
    \left(\sum_{s\in S}a_s\right)\log\frac{\sum_{s\in S} a_s}{\sum_{s\in S} b_s}$$.
  We require additionally that, if $b_s=0$, then $s_s=0$ as otherwise the right hand side should
  morally be infinite, which it can't be in the formalization using real numbers. -/
lemma sum_mul_log_div_leq {a b : ι → ℝ} (ha : ∀ i ∈ s, 0 ≤ a i) (hb : ∀ i ∈ s, 0 ≤ b i)
    (habs : ∀ i ∈ s, b i = 0 → a i = 0) :
    (∑ i ∈ s, a i) * log ((∑ i ∈ s, a i) / (∑ i ∈ s, b i)) ≤ ∑ i ∈ s, a i * log (a i / b i) := by
```

Theorem statement

```
lemma snd_deleteRight (κ : Kernel α (β × γ × δ)) : snd (deleteRight κ) = fst (snd κ)
```

miniCTX Benchmark

Data splits:

- Prime Number Theorem +
- **Polynomial Freiman–Ruzsa**
- Mathlib
- How to Prove It
- High Energy Physics
- Scientific Computing

Polynomial Freiman-Rusza:

Valid
- Theorem 1
- **Theorem 2**

…
- Theorem 50

Test
- Theorem 1
- Theorem 2

…
- Theorem 50

In-file context:
Preceding code

```
import Mathlib.Probability.Kernel.Composition.Comp
import PFR.Mathlib.Probability.Kernel.Disintegration

open Function MeasureTheory Real
open scoped ENNReal NNReal Topology ProbabilityTheory

namespace ProbabilityTheory.Kernel

section
variable {α β γ δ ε : Type*} {_ : MeasurableSpace α} {_ : MeasurableSpace β}
    {_ : MeasurableSpace γ} {_ : MeasurableSpace δ} {_ : MeasurableSpace ε}

variable {Ω S T : Type*} [mΩ : MeasurableSpace Ω]
  [MeasurableSpace S] [MeasurableSpace T] [MeasurableSpace γ]
  {κ : Kernel T S} {μ : Measure T} {X : Ω → S} {Y : Ω → β}

lemma map_map (κ : Kernel α β) {f : β → γ} (hf : Measurable f) {g : γ → δ} (hg : Measurable g) :
    map (map κ f) g = map κ (g ∘ f) := by
  ext x s _
  rw [map_apply _ hg, map_apply _ hf, map_apply _ (hg.comp hf), Measure.map_map hg hf]

⋮

@[simp]
lemma fst_deleteRight (κ : Kernel α (β × γ × δ)) : fst (deleteRight κ) = fst κ := by
  rw [deleteRight_eq, fst_map_prod, fst_eq]
  exact measurable_fst.comp measurable_snd
```

Cross-file context:
Lemmas & definitions from other files

```
/-- The real-valued version of a measure. Maps infinite measure sets to zero. Use as `μ.real s`. -/
protected def Measure.real (s : Set α) : ℝ := (μ s).toReal
```

```
/-- Given two independent random variables `U` and `V` uniformly distributed respectively on `A`
and `B`, then `U = V` with probability `#(A ∩ B) / #A · #B`. -/
lemma IsUniform.measureReal_preimage_sub_zero (Uunif : IsUniform A U) (Umeas : Measurable U)
    (Vunif : IsUniform B V) (Vmeas : Measurable V) (h_indep : IndepFun U V) :
    (ℙ : Measure Ω).real ((U - V) ⁻¹' {0})
      = Nat.card (A ∩ B : Set G) / (Nat.card A * Nat.card B) := by
```

```
/-- If $S$ is a finite set, and $a_s$, $b_s$ are non-negative for $s\in S$, then
  $$\sum_{s\in S} a_s \log\frac{a_s}{b_s}\ge
    \left(\sum_{s\in S}a_s\right)\log\frac{\sum_{s\in S} a_s}{\sum_{s\in S} b_s}$$.
  We require additionally that, if $b_s=0$, then $s_s=0$ as otherwise the right hand side should
  morally be infinite, which it can't be in the formalization using real numbers. -/
lemma sum_mul_log_div_leq {a b : ι → ℝ} (ha : ∀ i ∈ s, 0 ≤ a i) (hb : ∀ i ∈ s, 0 ≤ b i)
    (habs : ∀ i ∈ s, b i = 0 → a i = 0) :
    (∑ i ∈ s, a i) * log ((∑ i ∈ s, a i) / (∑ i ∈ s, b i)) ≤ ∑ i ∈ s, a i * log (a i / b i) := by
```

Theorem statement

```
lemma snd_deleteRight (κ : Kernel α (β × γ × δ)) : snd (deleteRight κ) = fst (snd κ)
```

Proof

```
rw [deleteRight_eq, snd_map_prod, snd_eq, fst_eq, map_map _ measurable_snd measurable_fst]
· rfl
· exact measurable_fst
```

Does Context Actually Matter?

No context model

Ignore context

Lean proof state

```
...
κ : Kernel α (β × γ × δ)
⊢ snd (deleteRight κ) = fst (snd κ)
```

↓

Proof step ("tactic")

```
rw [deleteRight_eq]
```

File context model

In-file context

```
import Mathlib.Probability.Kernel.Composition.Comp
import PFR.Mathlib.Probability.Kernel.Disintegration

open Function MeasureTheory Real
open scoped ENNReal NNReal Topology ProbabilityTheory

namespace ProbabilityTheory.Kernel

section
variable {α β γ δ ε : Type*} {_ : MeasurableSpace α} {_ : MeasurableSpace β}
    {_ : MeasurableSpace γ} {_ : MeasurableSpace δ} {_ : MeasurableSpace ε}

variable {Ω S T : Type*} [mΩ : MeasurableSpace Ω]
  [MeasurableSpace S] [MeasurableSpace T] [MeasurableSpace γ]
  {κ : Kernel T S} {μ : Measure T} {X : Ω → S} {Y : Ω → β}

lemma map_map (κ : Kernel α β) {f : β → γ} (hf : Measurable f) {g : γ → δ} (hg : Measurable g) :
    map (map κ f) g = map κ (g ∘ f) := by
  ext x s _
  rw [map_apply _ hg, map_apply _ hf, map_apply _ (hg.comp hf), Measure.map_map hg hf]

                                  ⋮

 @[simp]
 lemma fst_deleteRight (κ : Kernel α (β × γ × δ)) : fst (deleteRight κ) = fst κ := by
   rw [deleteRight_eq, fst_map_prod, fst_eq]
   exact measurable_fst.comp measurable_snd
```

Lean proof state

```
...
κ : Kernel α (β × γ × δ)
⊢ snd (deleteRight κ) = fst (snd κ)
```

↓

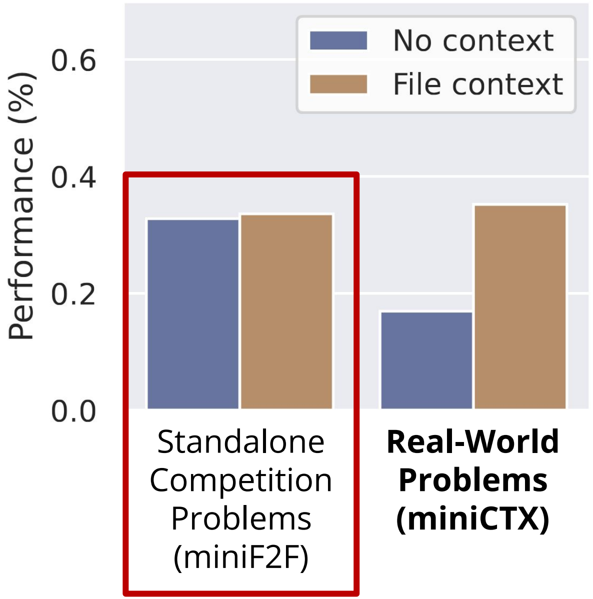
Proof step ("tactic")

```
rw [deleteRight_eq]
```

Carnegie Mellon University

Does Context Actually Matter?

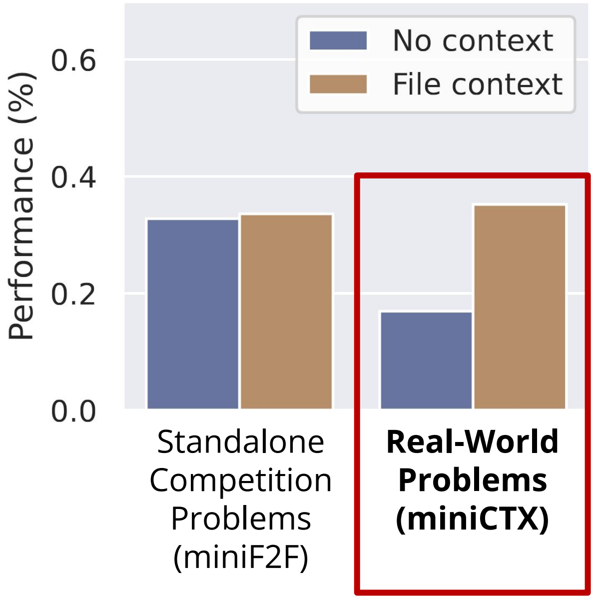
- Similar standalone competition (miniF2F) performance
- Much better real-world (miniCTX) performance



Carnegie Mellon University

Does Context Actually Matter?

- Similar standalone competition (miniF2F) performance
- Much better real-world (miniCTX) performance



Does Context Actually Matter?

- Similar standalone competition (miniF2F) performance
- Much better real-world (miniCTX) performance

Problem:

```
theorem Set.right_not_mem_uIoo {a b : ℝ} :
    b ∉ Set.uIoo a b := by
```

No context model:

```
rw [Set.uIoo_def]
```

❌

Problem:

```
theorem Set.right_not_mem_uIoo {a b : ℝ} :
    b ∉ Set.uIoo a b := by
```

Context contains analogous proof

```
...
theorem Set.left_not_mem_uIoo {a b : ℝ} :
    a ∉ Set.uIoo a b := by
  rintro ⟨h1, h2⟩
  exact (left_lt_sup.mp h2) (le_of_not_le (inf_lt_left.mp h1))
...
```

File context model:

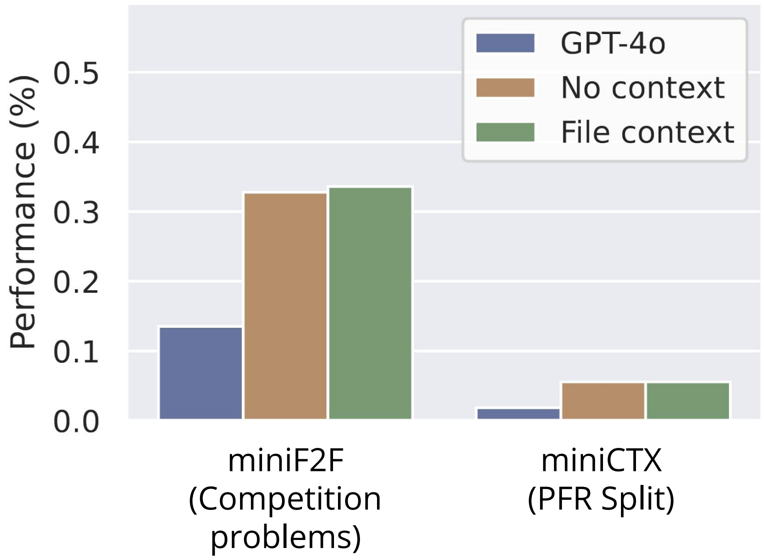
```
rintro ⟨h1, h2⟩
exact (right_lt_sup.mp h2) (le_of_not_le (inf_lt_right.mp h1))
```

✅

Carnegie Mellon University

Open Challenges

- Difficulty of research math



Carnegie Mellon University

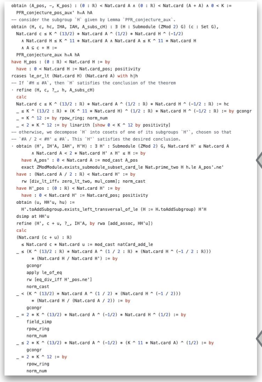
Open Challenges

- Difficulty of research math
- Harder proofs & more dependencies

Example miniF2F proof:

```
have hd : d = 15 / 2 := by
  linarith
have ha : a = -15 := by
  linarith [h0, hd]
linarith [ha, hd]
```

Example miniCTX proof:



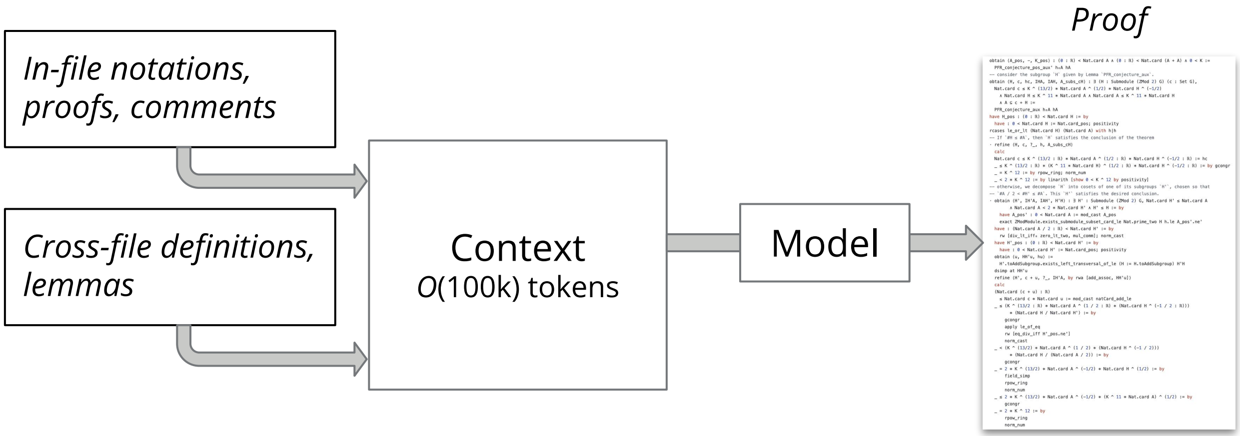
In-file notations, proofs, comments

Cross-file definitions, lemmas

Carnegie Mellon University

Open Challenges

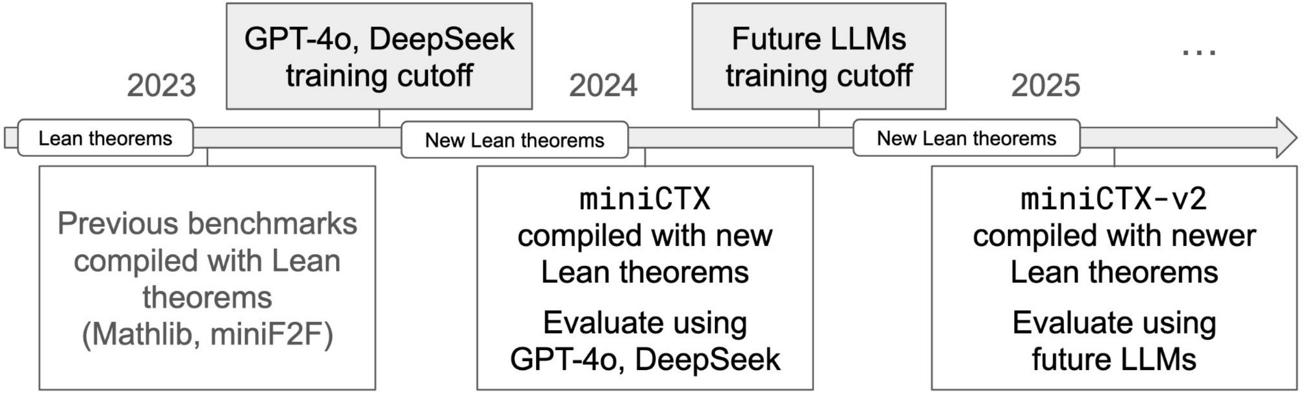
- Difficulty of research math
- Harder proofs & more dependencies
- Integrating different contexts



Carnegie Mellon University

Automatic Updates

- We periodically update miniCTX with newer theorems to stay ahead of LLM training
- We release miniCTX-v2 with theorems after November 2024
- Data is extracted automatically



Carnegie Mellon University

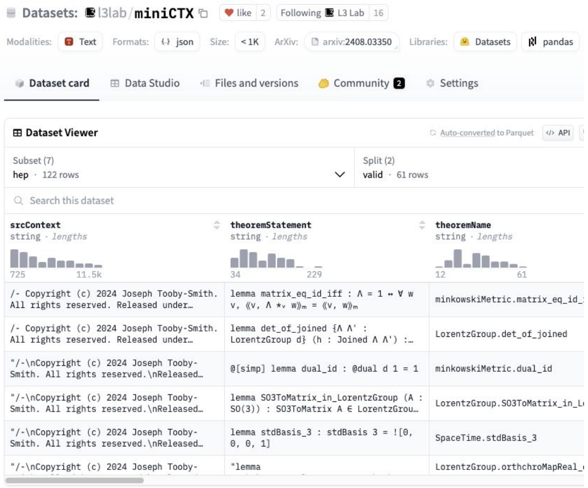
Toolkit & Resources

We open-source the miniCTX benchmark, training data, and evaluation and data extraction code

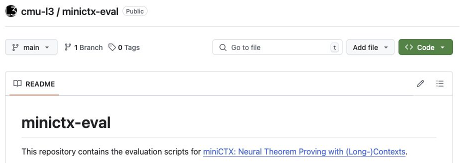
Project page

cmu-l3.github.io/minictx

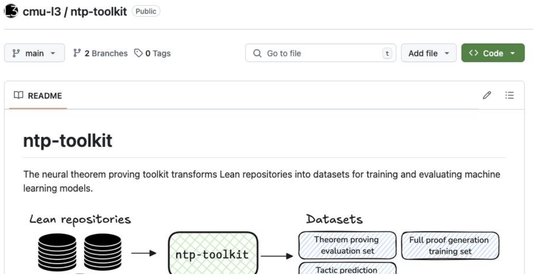
miniCTX benchmark



Evaluation code



Automated data extraction



Carnegie Mellon University