

# The Three Horsemen

AI Agents Know How to Lean

Nehal Patel

G-Research

Technology & Innovation

ITALean2025

# PutnamBench: A Multilingual Competition-Mathematics Benchmark for Formal Theorem-Proving

George Tsoukalas, Jasper Lee, John Jennings, Jimmy Xin, Michelle Ding, Michael Jennings,  
Amitayush Thakur, Swarat Chaudhuri

The University of Texas at Austin

Paper

arXiv

Dataset

Leaderboard

# Putnam, 1962, B1

Let  $x^{(n)} = x(x - 1) \cdots (x - n + 1)$  for  $n$  a positive integer and let  $x^{(0)} = 1$ . Prove that

$$(x + y)^{(n)} = \sum_{k=0}^n \binom{n}{k} x^{(k)} y^{(n-k)}.$$

# Putnam, 1962, B1

```
import Mathlib

/--
Let  $x^{(n)} = x(x-1)\cdots(x-n+1)$  for  $n$  a positive integer and let  $x^{(0)} = 1.$  Prove that  $(x+y)^{(n)} = \sum_{k=0}^n \{n \choose k\} x^{(k)} y^{(n-k)}.$ 
 -/
theorem putnam_1962_b1
  (p :  $\mathbb{N} \rightarrow \mathbb{R} \rightarrow \mathbb{R}$ )
  (x y :  $\mathbb{R}$ )
  (n :  $\mathbb{N}$ )
  (h0 :  $p 0 = \text{fun } x : \mathbb{R} \Rightarrow 1$ )
  (hp :  $\forall n > 0, p n = \text{fun } x : \mathbb{R} \Rightarrow \prod_{i \in \text{Finset.range } n} (x - i)$ )
  :  $p n (x+y) = \sum_{k \in \text{Finset.range } (n+1)} \text{Nat.choose } n k * (p k x) * (p (n-k) y) :=$ 
  sorry
```

Lean (out of 660)

# PutnamBench: A Multilingual Competition-Mathematics Benchmark for Formal Theorem-Proving

George Tsoukalas, Jasper Lee, John Jennings, Jimmy Xin, Michelle Ding, Michael Jennings,  
Amitayush Thakur, Swarat Chaudhuri  
The University of Texas at Austin

[Paper](#) [arXiv](#) [Dataset](#) [Leaderboard](#)

#	Model	num-solved	compute
1	<a href="#">Aleph Prover</a>	500	pass@1, avg 1834 tool calls
2	<a href="#">Hilbert</a>	462	avg pass@1840
3	<a href="#">Seed-Prover</a>	329	MEDIUM
4	<a href="#">Ax-Prover</a> ❤	91	pass@1, avg. 100 tool calls
5	<a href="#">Goedel-Prover-V2</a> ❤	86	pass@184
6	<a href="#">DeepSeek-Prover-V2</a> ❤	47	pass@1024
7	<a href="#">GPT-5 (ReAct, 10 turns)</a>	28	pass@1, 10 tool calls
8	<a href="#">DSP+</a> ❤	23	pass@128

# AI Agents know how to Lean?

## PutnamBench Baseline

agent	model	solved
claude	opus-4.5	203
codex	gpt-5.1-codex	90
claude	sonnet-4.5	39

# PutnamBench

Lean (out of 660)

Claude Code,  
Opus 4.5



#	Model	num-solved	compute
1	<a href="#">Aleph Prover</a>	500	pass@1, avg 1834 tool calls
2	<a href="#">Hilbert</a>	462	avg pass@1840
3	<a href="#">Seed-Prover</a>	329	MEDIUM
4	<a href="#">Ax-Prover</a> ❤	91	pass@1, avg. 100 tool calls
5	<a href="#">Goedel-Prover-V2</a> ❤	86	pass@184
6	<a href="#">DeepSeek-Prover-V2</a> ❤	47	pass@1024
7	<a href="#">GPT-5 (ReAct, 10 turns)</a>	28	pass@1, 10 tool calls
8	<a href="#">DSP+</a> ❤	23	pass@128

# AI Agents know how to Lean?

## PutnamBench Baseline

agent	model	solved
claude	opus-4.5	203
codex	gpt-5.1-codex	90
claude	sonnet-4.5	39

# AI Agents know how to Lean?

## PutnamBench Baseline

agent	model	solved
craude	opus-4.5	203
codex	gpt-5.1-codex	90
cautle	sonnet-4.5	39

# AI Agents know how to Lean?

## PutnamBench Baseline

agent	model	solved
craude	opus-4.5	~0
codex	gpt-5.1-codex	~0
caudle	sonnet-4.5	~0

← → 🔍 2.0.57 — threehorsemen ⚙️ ⚙️ ⚙️ ⚙️ ⚙️ ⚙️ ⚙️ ⚙️

☰ Weave.lean U

```
1 import Mathlib
2
3
```

☰ 2.0.57

threehorsemen on 🏫 main [?] on ☁ (us-east-1) on ☈ nehal.alum@gmail.com  
❯ claude

Claude Code v2.0.57

Welcome back Nehal!



Opus 4.5 · Claude Max  
~/src/threehorsemen

Tips for getting started  
Run /init to create a CLAUDE.md file with...

Recent activity  
No recent activity

---

> Try "refactor <filepath>"

Opus 4.5 | ID 6ba00bc8 | \$ \$0.00 | threehorsemen  
main

In Weave.lean

✖ main\* 🔍 marimo ✎ 0 ⚡ 0 Live Share ⚙️ ⚙️ ⚙️ ⚙️ ⚙️ ⚙️ ⚙️ ⚙️

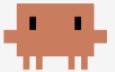
Screen Reader Optimized ⚙️ ⚙️ ⚙️

← → 🔍 2.0.57 — threehorsemen ⚡ ...

2.0.57  
threehorsemen on 🏴 main [?] on ☁ (us-east-1) on ☁ nehal.alum@gmail.com  
» claude

Claude Code v2.0.57

Welcome back Nehal!



Opus 4.5 · Claude Max  
~/src/threehorsemen

Tips for getting started  
Run /init to create a CLAUDE.md file with...

Recent activity  
No recent activity

> hi there, could you help me model weaves in Lean4? I'd like to define a weave as a structure that has one field, which is a function that maps natural numbers i, j to Z mod 2.

Let's remind ourselves what a weave is: a weave is a fabric that has vertical threads (the warps) and horizontal threads (the wefts). At their intersections, either the vertical thread is on top or the horizontal thread is on top. And so at these intersections, what the weave function tells you is which function is on top, which thread is on top, the vertical or the horizontal.

And actually, let's just keep it simple and model a weave not as a let's just deal with only periodic weaves of square weaves of size N. So we're a weave as a function that goes from  $Z \bmod N$  cross  $Z \bmod N$  to  $Z \bmod 2$ . Once we have a weave, we can define various properties of a weave. A weave is balanced if it has exactly the same number of vertical threads on top as horizontal threads on top. A weave is reversible if you can flip it over and it looks the same. Concretely, that means when you flip it over, what was on top is now on the bottom, and vice versa. So you basically want  $W$  of  $ij$  equal to one minus  $W$  of  $ij$ , but actually because you flipped it over, the  $i$ -coordinate is negated. So a weave that is reversible is one where  $W_{ij}$  equals one minus  $W_{-i,j}$ . And then finally we can prove a theorem which is: If a weave is reversible, then it is balanced. Okay, can you model all this in Lean 4? You can put it in the file called weave.lean. ■

⌚ Opus 4.5 |>ID 6ba00bc8 | 💰 \$0.00 | 📱 In Weave.lean  
threehorsemen | 🌱 main

ctrl-g to edit prompt VS Code in

✖ main\* 🔍 marimo ✎ 0 ⚠ 0 Live Share

Screen Reader Optimized 🌐

```
eweave.lean U
1 import Mathlib
```

> hi there, could you help me model weaves in Lean4? I'd like to define a weave as a structure that has one field, which is a function that maps natural numbers  $i, j$  to  $Z \text{ mod } 2$ .

Let's remind ourselves what a weave is: a weave is a fabric that has vertical threads (the warps) and horizontal threads (the wefts). At their intersections, either the vertical thread is on top or the horizontal thread is on top. And so at these intersections, what the weave function tells you is which function is on top, which thread is on top, the vertical or the horizontal.

And actually, let's just keep it simple and model a weave not as a let's just deal with only periodic weaves of square weaves of size  $N$ . So we're a weave as a function that goes from  $Z \text{ mod } N \times Z \text{ mod } N$  to  $Z \text{ mod } 2$ . Once we have a weave, we can define various properties of a weave. A weave is balanced if it has exactly the same number of vertical threads on top as horizontal threads on top. A weave is reversible if you can flip it over and it looks the same. Concretely, that means when you flip it over, what was on top is now on the bottom, and vice versa. So you basically want  $W_{ij}$  equal to one minus  $W_{ij}$ , but actually because you flipped it over, the  $i$ -coordinate is negated. So a weave that is reversible is one where  $W_{ij}$  equals one minus  $W_{-i, -j}$ . And then finally we can prove a theorem which is: If a weave is reversible, then it is balanced. Okay, can you model all this in Lean 4? You can put it in the file called weave.lean. ■

+ X ...  
hal.alum@gmail.com

AUDE.md file with...

I'd like to define  
a function that

a fabric that has  
(the wefts). At  
on top or the  
ections, what the  
, which thread is

weave not as a  
weaves of size  $N$ .  
 $N \times Z \text{ mod } N$  to  
properties of a  
the number of  
. A weave is  
same. Concretely,  
now on the  
 $W_{ij}$  equal to one  
over, the  
ile is one where  $W$   
e can prove a  
is balanced. Okay,  
the file called

@ In Weave.lean

ctrl-g to edit prompt VS Code

threehorsemen | main

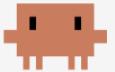
ctrl-g to edit prompt VS Code  
in

← → 🔍 2.0.57 — threehorsemen ⚡ ...

2.0.57  
threehorsemen on 🏴 main [?] on ☁ (us-east-1) on ☁ nehal.alum@gmail.com  
» claude

Claude Code v2.0.57

Welcome back Nehal!



Opus 4.5 · Claude Max  
~/src/threehorsemen

Tips for getting started  
Run /init to create a CLAUDE.md file with...

Recent activity  
No recent activity

> hi there, could you help me model weaves in Lean4? I'd like to define a weave as a structure that has one field, which is a function that maps natural numbers  $i, j$  to  $Z \bmod 2$ .

Let's remind ourselves what a weave is: a weave is a fabric that has vertical threads (the warps) and horizontal threads (the wefts). At their intersections, either the vertical thread is on top or the horizontal thread is on top. And so at these intersections, what the weave function tells you is which function is on top, which thread is on top, the vertical or the horizontal.

And actually, let's just keep it simple and model a weave not as a let's just deal with only periodic weaves of square weaves of size  $N$ . So we're a weave as a function that goes from  $Z \bmod N$  cross  $Z \bmod N$  to  $Z \bmod 2$ . Once we have a weave, we can define various properties of a weave. A weave is balanced if it has exactly the same number of vertical threads on top as horizontal threads on top. A weave is reversible if you can flip it over and it looks the same. Concretely, that means when you flip it over, what was on top is now on the bottom, and vice versa. So you basically want  $W$  of  $ij$  equal to one minus  $W$  of  $ij$ , but actually because you flipped it over, the  $i$ -coordinate is negated. So a weave that is reversible is one where  $W_{ij}$  equals one minus  $W_{-i,j}$ . And then finally we can prove a theorem which is: If a weave is reversible, then it is balanced. Okay, can you model all this in Lean 4? You can put it in the file called weave.lean. ■

⌚ Opus 4.5 |>ID 6ba00bc8 |💵 \$0.00 |💻 In Weave.lean  
threehorsemen |🌱 main  
ctrl-g to edit prompt VS Code in

✖ main\* 🔍 marimo ⚡ 0 ⚡ 0 Live Share

Screen Reader Optimized 🌐

### Weave.lean U

```
1 import Mathlib
2
3
```

### 2.0.57

threehorsemen on main [?] on (us-east-1) on nehal.alum@gmail.com  
› claude

Let's remind ourselves what a weave is: a weave is a fabric that has vertical threads (the warps) and horizontal threads (the wefts). At their intersections, either the vertical thread is on top or the horizontal thread is on top. And so at these intersections, what the weave function tells you is which function is on top, which thread is on top, the vertical or the horizontal.

And actually, let's just keep it simple and model a weave not as a let's just deal with only periodic weaves of square weaves of size  $N$ . So we're a weave as a function that goes from  $Z \text{ mod } N \times Z \text{ mod } N$  to  $Z \text{ mod } 2$ . Once we have a weave, we can define various properties of a weave. A weave is balanced if it has exactly the same number of vertical threads on top as horizontal threads on top. A weave is reversible if you can flip it over and it looks the same. Concretely, that means when you flip it over, what was on top is now on the bottom, and vice versa. So you basically want  $W$  of  $ij$  equal to one minus  $W$  of  $ij$ , but actually because you flipped it over, the  $i$ -coordinate is negated. So a weave that is reversible is one where  $W_{ij}$  equals one minus  $W_{-i, j}$ . And then finally we can prove a theorem which is: If a weave is reversible, then it is balanced. Okay, can you model all this in Lean 4? You can put it in the file called weave.lean.

- I'll help you model weaves in Lean 4. Let me first check what's currently in the file, then write the definitions and theorem.
  - **Read(Weave.lean)**
    - └ Read 3 lines
  - Now let me write the weave model with the structure, properties, and theorem:
  - **Update(Weave.lean)**

---

[Opened changes in Visual Studio Code](#)

Save file to continue...

Do you want to make this edit to **Weave.lean**?

- › 1. Yes
- 2. Yes, allow all edits during this session (shift+tab)
- 3. No, and tell Claude what to do differently (esc)

## ≡ Weave.lean 1, U

```

1 import Mathlib
2 /-
3 # Weaves
4
5 A weave models a fabric with vertical threads (warps) and horizontal
6 threads (wefts).
7 At each intersection (i, j), either the vertical thread is on top or
8 the horizontal
9 thread is on top. We represent this with ZMod 2: say 0 means vertical
10 on top,
11 1 means horizontal on top.
12
13
14 /-- A periodic square weave of size N. The function `f i j` indicates
15 which thread
16 is on top at position (i, j). -/
17 structure Weave (N : ℕ) [NeZero N] where
18   f : ZMod N → ZMod N → ZMod 2
19
20 namespace Weave
21
22 variable {N : ℕ} [NeZero N]
23
24 /-- A weave is balanced if it has exactly the same number of positions
25 where the
26 vertical thread is on top as positions where the horizontal thread is
27 on top. -/
28 def Balanced (W : Weave N) : Prop :=
29   2 * (Finset.univ.filter (fun p : ZMod N × ZMod N => W.f p.1 p.2 =
30     1).card = N * N
31
32 /-- A weave is reversible if flipping it over produces the same
33 pattern.
```

## ≡ 2.0.57

```

20 +
21 + variable {N : ℕ} [NeZero N]
22 +
23 + /-- A weave is balanced if it has exactly the same
24 + number of positions where the
25 + vertical thread is on top as positions where the
26 + horizontal thread is on top. -/
27 + def Balanced (W : Weave N) : Prop :=
28   2 * (Finset.univ.filter (fun p : ZMod N × ZMod N =>
29     W.f p.1 p.2 = 1)).card = N * N
30 +
31 + /-- A weave is reversible if flipping it over produces
32 + the same pattern.
33 + When flipped, the i-coordinate is negated and what was
34 + on top is now on bottom.
35 + So W(i,j) = 1 - W(-i, j). -/
36 + def Reversible (W : Weave N) : Prop :=
37   ∀ i j : ZMod N, W.f i j = 1 - W.f (-i) j
38 +
39 + /-- If a weave is reversible, then it is balanced.
40 +
41 + The key insight is that reversibility gives us a
42 + pairing: the map (i, j) ↦ (-i, j)
43 + is a bijection, and reversibility says W(i,j) = 1 -
44 + W(-i,j). So positions with value 1
45 + are in bijection with positions with value 0,
46 + hence they have equal count. -/
47 theorem reversible_imp_balanced (W : Weave N) (h :
48   W.Reversible) : W.Balanced := by
49   sorry
50
51 end Weave
52
```

- Let me verify this compiles and then work on the proof:

∴ Thought for 1s (**ctrl+o** to show thinking)

\* Forging... (**esc** to interrupt • 1m 25s • ↓ 1.4k tokens)

> █

⌚ Opus 4.5 | 📁 6ba00bc8 | 💸 \$0.31 | 🗂️ threehorsemen | 🌱 main  
▶ accept edits on (shift+tab to cycle)



```
☰ Weave.lean 1, U
19 namespace Weave
20 So W(i,j) = 1 - W(-i, j). -/
21 def Reversible (W : Weave N) : Prop :=
22   ∀ i j : ZMod N, W.f i j = 1 - W.f (-i) j
23
24 /-- If a weave is reversible, then it is balanced.
25
26 The key insight is that reversibility gives us a pairing: the map (i,
27 j) ↦ (-i, j)
28 is a bijection, and reversibility says W(i,j) = 1 - W(-i,j). So
29 positions with
30 value 1 are in bijection with positions with value 0, hence they have
31 equal count. -/
32
33 theorem reversible_imp_balanced (W : Weave N) (h : W.Reversible) :
34   W.Balanced := by
35     sorry
36
37 end Weave
```

```
2.0.57
26 +   2 * (Finset.univ.filter (fun p : ZMod N × ZMod N =>
27 +     W.f p.1 p.2 = 1)).card = N * N
28 + /-- A weave is reversible if flipping it over produces
29 + the same pattern.
30 + When flipped, the i-coordinate is negated and what was
31 + on top is now on bottom.
32 + So W(i,j) = 1 - W(-i, j). -/
33 + def Reversible (W : Weave N) : Prop :=
34 +   ∀ i j : ZMod N, W.f i j = 1 - W.f (-i) j
35 +
36 + /-- If a weave is reversible, then it is balanced.
37 +
38 + The key insight is that reversibility gives us a
39 + pairing: the map (i, j) ↦ (-i, j)
40 + is a bijection, and reversibility says W(i,j) = 1 -
41 + W(-i,j). So positions with
42 + value 1 are in bijection with positions with value 0,
43 + hence they have equal count. -/
44 + theorem reversible_imp_balanced (W : Weave N) (h :
+ W.Reversible) : W.Balanced := by
45   sorry
46
47 + end Weave
48 +
```

- Let me verify this compiles and then work on the proof:

- Bash(lake build 2>&1 | head -50)
  - Running...

\* Shenaniganing... (esc to interrupt · 1m 35s · ↓ 3.2k tokens)

>

Opus 4.5 | 6ba00bc8 | \$0.33 | threehorsemen | main  
accept edits on (shift+tab to cycle)  
ctrl+b to run in background

☰ Weave.lean 9+, U

```

19  namespace Weave
39  theorem reversible_imp_balanced (W : Weave N) (h : W.Reversible) :
42    set ones := Finset.univ.filter (fun p : ZMod N × ZMod N => W.f p.1
p.2 = 1) with h_ones
43    set zeros := Finset.univ.filter (fun p : ZMod N × ZMod N => W.f p.1
p.2 = 0) with h_zeros
44    -- The negation map on the first coordinate
45    let φ : ZMod N × ZMod N → ZMod N × ZMod N := fun p => (-p.1, p.2)
46    -- φ is an involution, hence a bijection
47    have hφ_inv : ∀ p, φ(φ p) = p := fun p => by simp [φ]
48    -- Reversibility implies: W(i,j) = 1 iff W(-i,j) = 0
49    have h_flip : ∀ p : ZMod N × ZMod N, W.f p.1 p.2 = 1 ↔ W.f (-p.1)
p.2 = 0 := by
50      intro (i, j)
51      simp only
52      constructor
53        · intro h1
54          have := h i j
55          simp only [h1] at this
56          omega
57        · intro h0
58          have := h i j
59          omega
60    -- φ maps ones to zeros
61    have hφ_ones_to_zeros : ∀ p ∈ ones, φ p ∈ zeros := by
62      intro p hp
63      simp only [Finset.mem_filter, Finset.mem_univ, true_and] at hp ⊢
64      exact (h_flip p).mp hp
65    -- φ maps zeros to ones
66    have hφ_zeros_to_ones : ∀ p ∈ zeros, φ p ∈ ones := by
67      intro p hp
68      simp only [Finset.mem_filter, Finset.mem_univ, true_and] at hp ⊢
69      simp only [φ, neg_neg]
70      exact (h_flip (φ p)).mpr (by simp [φ, hp]) ✘
71    -- Therefore |ones| = |zeros|
72    have h_card_eq : ones.card = zeros.card := by
73      apply Finset.card_bii (fun p => φ p)

```



2.0.57

e  
r  
s  
n  
b  
n

```

n/.lake/packages/importGraph/.lake/build/lib/lean:/Users/nehal/src/threehorsemen/.lake/packages/LeanSearchClient/.lake/build/lib/lean:/Users/nehal/src/threehorsemen/.lake/packages/plausible/.lake/build/lib/lean:/Users/nehal/src/threehorsemen/.lake/packages/mathlib/.lake/build/lib/lean:/Users/nehal/src/threehorsemen/.lake/build/lib/lean/Users/nehal/.elan/toolchains/leanprover--lean4---v4.26.0-rc2/bin/lean
/n/.lake/packages/importGraph/.lake/build/lib/lean:/Users/nehal/src/threehorsemen/.lake/build/lib/lean/Weave.olean -i /Users/nehal/src/threehorsemen/.lake/build/lib/lean/Weave.ilean -c /Users/nehal/src/threehorsemen/.lake/build/ir/Weave.c --setup /Users/nehal/src/threehorsemen/.lake/build/ir/Weave.setup.json --json
error: Weave.lean:56:6: omega could not prove the goal:
No usable constraints found. You may need to unfold definitions so `omega` can see linear arithmetic facts about `Nat` and `Int`, which may also involve multiplication, division, and modular remainder by constants.
error: Weave.lean:59:6: omega could not prove the goal:
No usable constraints found. You may need to unfold definitions so `omega` can see linear arithmetic facts about `Nat` and `Int`, which may also involve multiplication, division, and modular remainder by constants.
error: Weave.lean:64:24: Application type mismatch: The argument
  hp
has type
... +140 lines (ctrl+o to see all)
└ Found 20 new diagnostic issues in 1 file (ctrl+o to expand)
● The proof needs significant reworking. Let me rewrite it with proper tactics for ZMod 2 arithmetic:
∴ Thought for 2s (ctrl+o to show thinking)
* Shenaniganing... (esc to interrupt • 2m 31s • ↓ 8.3k tokens)
```

> [ ]

⌚ Opus 4.5 | 📄 6ba00bc8 | 💸 \$0.64 | 📱 threehorsemen | 🌿 main  
▶ accept edits on (shift+tab to cycle)

### Weave.lean 5, U

```

19  namespace Weave
39  theorem reversible_imp_balanced (W : Weave N) (h : W.Reversible) :
42    set ones := Finset.univ.filter (fun p : ZMod N × ZMod N => W.f p.1
p.2 = 1) with h_ones
43    set zeros := Finset.univ.filter (fun p : ZMod N × ZMod N => W.f p.1
p.2 = 0) with h_zeros
44    -- The negation map on the first coordinate
45    let φ : ZMod N × ZMod N → ZMod N × ZMod N := fun p => (-p.1, p.2)
46    -- φ is an involution, hence a bijection
47    have hφ_inv : ∀ p, φ(φ p) = p := fun p => by simp [φ]
48    -- Reversibility implies: W(i,j) = 1 iff W(-i,j) = 0
49    have h_flip : ∀ p : ZMod N × ZMod N, W.f p.1 p.2 = 1 ↔ W.f (-p.1)
p.2 = 0 := by
50      intro (i, j)
51      simp only
52      have href := h i j
53      constructor
54        · intro h1
55          rw [h1] at href
56          simp only [ZMod.one_val, Nat.lt_irrefl, ↑reduceIte] at href ⊢
exact sub_eq_zero.mp href
57        · intro h0
58          rw [h0] at href
59          simp only [ZMod.val_zero, Nat.lt_one_iff, ↑reduceIte, sub_zero]
at href
60          exact href
61          -- φ maps ones to zeros
62          have hφ_ones_to_zeros : ∀ p ∈ ones, φ p ∈ zeros := by
63            intro p hp
64            simp only [h_ones, h_zeros, Finset.mem_filter, Finset.mem_univ,
true_and] at hp ⊢
65            exact (h_flip p).mp hp
66          -- φ maps zeros to ones
67          have hφ_zeros_to_ones : ∀ p ∈ zeros, φ p ∈ ones := by
68            intro p hp
69            simp only [h_ones, h_zeros, Finset.mem_filter, Finset.mem_univ,
true_and] at hp ⊢

```

### 2.0.57

```

85      have h_partition : ones.card + zeros.card = N * N
:= by
86        have h_disjoint : Disjoint ones zeros := by
87          rw [Finset.disjoint_filter]
88          - intro _ _ h1 h0
89          - omega
89          + intro p _ h1 h0
90          + rw [h1] at h0
90          + exact one_ne_zero h0
91        have h_union : ones ∪ zeros = Finset.univ := by
92          ext p
93          simp only [Finset.mem_union,
93          Finset.mem_filter, Finset.mem_univ, true_and]
94          constructor
95          · intro _; trivial
96          · intro _
97            rcases (W.f p.1 p.2).val_lt_two with h | h
98            · right; exact ZMod.val_eq_zero.mp h
98            · left; exact ZMod.val_eq_one.mp h
100       have := Finset.card_union_of_disjoint h_disjoint
101       rw [h_union, Finset.card_univ, Fintype.card_prod,
101       ZMod.card, ZMod.card] at this
102       omega
103       simp only [Finset.mem_union, h_ones, h_zeros,
103       Finset.mem_filter, Finset.mem_univ,
103       true_and, true_iff]
104       fin_cases (W.f p.1 p.2) <>; simp
105       have hunion_card := Finset.card_union_of_disjoint
105       h_disjoint
106       rw [h_union, Finset.card_univ, Fintype.card_prod,
106       ZMod.card, ZMod.card] at hunion_card
107       exact hunion_card.symm
108       -- Combining: 2 * |ones| = N * N
109       omega
110
111     ∵ Thought for 1s (ctrl+o to show thinking)
112
113     * Shenaniganing... (esc to interrupt • 3m 3s • ↓ 9.8k tokens)
114
115     > █
116
117     🎨 Opus 4.5 | 📁 6ba00bc8 | 💸 $0.82 | 📱 threehorsemen | 🌳 main
117     ►► accept edits on (shift+tab to cycle)

```

main\*
marimo
5 ▲ 0
Live Share

Screen Reader Optimized
Screen Reader Optimized

2.0.57 — threehorsemen

Weave.lean 9+, U

```

19 namespace Weave
20 equal_count. -/
21 theorem reversible_imp_balanced (W : Weave N) (h : W.Reversible) :
22 W.Balanced := by
23   unfold Balanced
24   -- Let ones = positions with value 1, zeros = positions with value 0
25   set ones := Finset.univ.filter (fun p : ZMod N × ZMod N => W.f p.1
26     p.2 = 1) with h_ones
27   set zeros := Finset.univ.filter (fun p : ZMod N × ZMod N => W.f p.1
28     p.2 = 0) with h_zeros
29   -- The negation map on the first coordinate
30   let φ : ZMod N × ZMod N → ZMod N × ZMod N := fun p => (-p.1, p.2)
31   -- φ is an involution, hence a bijection
32   have hφ_inv : ∀ p, φ (φ p) = p := fun p => by simp [φ]
33   -- Reversibility implies: W(i,j) = 1 iff W(-i,j) = 0
34   have h_flip : ∀ p : ZMod N × ZMod N, W.f p.1 p.2 = 1 ↔ W.f (-p.1)
35     p.2 = 0 := by
36     intro {i, j}
37     have hrev := h i j
38     rw [zmod2_eq_one_sub_iff] at hrev
39     constructor
40     · intro h1
41       by_contra h0
42       push_neg at h0
43       have : W.f (-i) j ≠ 0 := h0
44       interval_cases (W.f (-i) j)
45         · exact this rfl
46         · exact hrev (h1.symm.trans rfl)
47     · intro h0
48       by_contra h1
49       push_neg at h1
50       have : W.f i j ≠ 1 := h1
51       interval_cases (W.f i j)
52         · exact hrev (h0.symm.trans rfl).symm
53         · exact this rfl
54       -- φ maps ones to zeros
55       have hφ_ones_to_zeros : ∀ p ∈ ones, φ p ∈ zeros := by
56         intro p hp
57         simp only [h_ones, h_zeros, Finset.mem_filter,
58           Finset.mem_univ, true_and] at hp ⊢
59         have : W.f (p.1)
60         p.2 = 1 ↔ W.f (-p.1) p.2 = 0 := h_flip (φ p)
61         simp only [neg_neg, φ] at this
62         exact this.mpr hp
63         have key : W.f (-p.1)
64         p.2 = 1 ↔ W.f (-p.1) p.2 = 0 := h_flip (φ p)
65         simp only [neg_neg] at key
66         exact key.mpr hp
67         -- Therefore |ones| = |zeros|
68         have h_card_eq : ones.card = zeros.card := by
69           apply Finset.card_bij (fun p _ => φ p)
70
71         have h_partition : ones.card + zeros.card = N * N :=
72           by
73             have h_disjoint : Disjoint ones zeros := by
74               rw [Finset.disjoint_filter]
75             intro p _ h1 h0
76             intro _ h1 h0
77             rw [h1] at h0
78             exact one_ne_zero h0
79             have h_union : ones ∪ zeros = Finset.univ := by
80               ext p
81               simp only [Finset.mem_union, h_ones, h_zeros,
82                 Finset.mem_filter, Finset.mem_univ,
83                 true_and, true_iff]
84             fin_cases (W.f p.1 p.2) <>; simp
85             interval_cases (W.f p.1 p.2) <>; simp
86             have hunion_card := Finset.card_union_of_disjoint
87               h_disjoint
88             rw [h_union, Finset.card_univ, Fintype.card_prod,
89               ZMod.card, ZMod.card] at hunion_card
90             exact hunion_card.symm
91
92             ∵ Thought for 3s (ctrl+o to show thinking)
93             ∵ Shenaniganing... (esc to interrupt • 4m 4s • ↑ 11.8k tokens)
94
95             >
96             Opus 4.5 | 6ba00bc8 | $1.10 | threehorsemen | main
97             ►► accept edits on (shift+tab to cycle)

```

Screen Reader Optimized

Weave.lean U

```

19  namespace Weave
47  theorem reversible_imp_balanced (W : Weave N) (h : W.Reversible) :
55    have hφ_inv : ∀ p, φ(φ p) = p := fun p => by simp [φ]
56    -- Reversibility implies: W(i,j) = 1 iff W(-i,j) = 0
57    have h_flip : ∀ p : ZMod N × ZMod N, W.f p.1 p.2 = 1 ↔ W.f (-p.1)
      p.2 = 0 := by
        intro {i, j}
        have hrev := h i j
        rw [zmod2_eq_one_sub_iff] at hrev
        constructor
        · intro h1
          rcases zmod2_eq_zero_or_one (W.f (-i) j) with h0 | h0'
            · exact h0
            · exfalso; exact hrev (h1.trans h0'.symm)
        · intro h0
          rcases zmod2_eq_zero_or_one (W.f i j) with h1 | h1'
            · exfalso; exact hrev (h0.trans h1.symm).symm
            · exact h1'
        -- φ maps ones to zeros
        have h_φ_ones_to_zeros : ∀ p ∈ ones, φ p ∈ zeros := by
          intro p hp
          simp only [h_ones, h_zeros, Finset.mem_filter, Finset.mem_univ,
          true_and] at hp ⊢
          exact (h_flip p).mp hp
        -- φ maps zeros to ones
        have h_φ_zeros_to_ones : ∀ p ∈ zeros, φ p ∈ ones := by
          intro p hp
          simp only [h_ones, h_zeros, Finset.mem_filter, Finset.mem_univ,
          true_and] at hp ⊢
          have key : W.f (φ p).1 (φ p).2 = 1 ↔ W.f (-φ p).1 (φ p).2 = 0 :=
          h_flip (φ p)
          simp only [φ, neg_neg] at key
          exact key.mpr hp
        -- Therefore |ones| = |zeros|
        have h_card_eq : ones.card = zeros.card := by
          apply Finset.card_bij (fun p _ => φ p)
          · exact h_φ_ones_to_zeros

```

2.0.57

109      simp only [ZMod.card] at hunion\_card

- Bash(lake build 2>&1)
  - ✓ [7709/7710] Built Weave (7.2s)  
Build completed successfully (7710 jobs).
- The proof compiles successfully. Let me show you the complete file:
- Read(Weave.lean)
  - Read 116 lines
- The model is complete and compiles successfully. Here's a summary of what's in Weave.lean:

**Structure:**

- Weave N - A periodic square weave of size N, represented as a function  $f : \text{ZMod } N \rightarrow \text{ZMod } N \rightarrow \text{ZMod } 2$

**Definitions:**

- Balanced - A weave where exactly half the positions have value 1 (horizontal on top) and half have value 0 (vertical on top).  
Formally:  $2 * |\{p : W.f p = 1\}| = N * N$
- Reversible - A weave that looks the same when flipped over. When flipped, the  $i$ -coordinate negates and top/bottom swap. Formally:  $\forall i j, W.f i j = 1 - W.f (-i) j$

**Theorem:**

- reversible\_imp\_balanced - If a weave is reversible, then it is balanced.

The proof works by showing that the map  $\phi(i, j) = (-i, j)$  is an involution that swaps positions with value 1 and positions with value 0 (due to reversibility). Since this is a bijection, both sets have equal cardinality, and since they partition all  $N^2$  positions, each must have exactly  $N^2/2$  elements.

---

> █

Opus 4.5 | 6ba00bc8 | \$1.88 | threehorsemen | main  
► accept edits on (shift+tab to cycle)

BasicWidget.lean — warpsnwefts

BasicWidget.lean WnW/Demo/BasicWidget.lean/{} WarpsnWefts ▶ ⚡ ∀ 🔎 🤖 ✎ × ...

```
12  namespace WarpsnWefts
13  def herringboneDemo : Weave 12 :=
14  | Patterns.herringbone (n := 12) (segmentLen := 3) (over := 2) (under
15  := 2)
16
17  def diamondDemo : Weave 24 :=
18  | Patterns.diamond (n := 24) (halfDiagonal := 5)
19
20  def sampleColoredWeave : ColoredWeave 4 3 2 where
21  | weave := sampleWeave
22  | warpColoring := fun i => (i.val : ZMod 3)
23  | weftColoring := fun j => (j.val : ZMod 2)
24
25
26  #weave_widget sampleWeave
27
28  #weave_widget twill sync, Nehal Patel (4 weeks ago)
29
30  #weave_widget plainDemo
31
32  #weave_widget risingTwillDemo
33
34  #weave_widget herringboneDemo
35
36  #weave_widget diamondDemo
37
38
39  #colored_weave_widget sampleColoredWeave
40
41  end WarpsnWefts
42
43
44
45
46
47
48
49
50
51
52
53
54
55
56
57
58
59
60
61
```

Lean InfoView

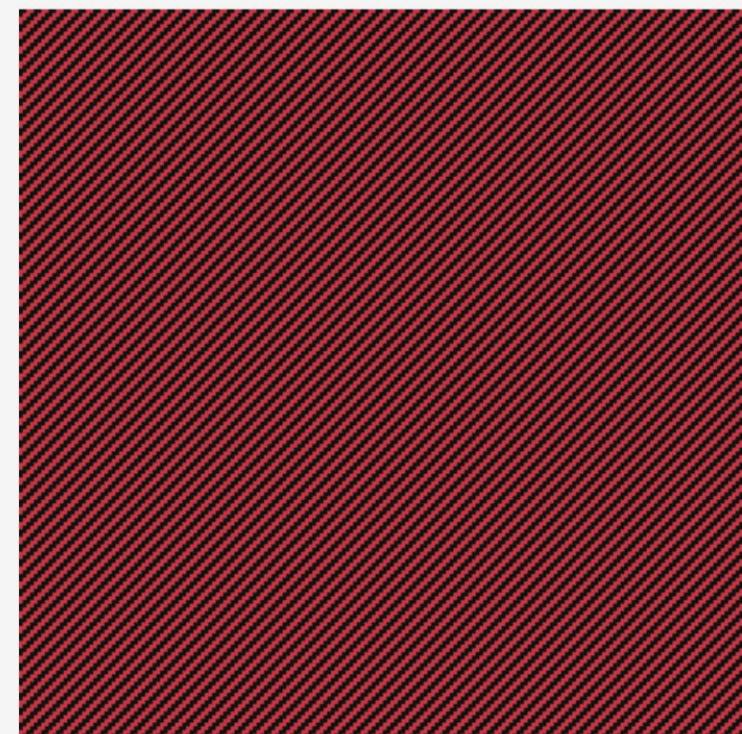
BasicWidget.lean:47:16

Expected type

← Weave 4

Weave (size 4): The weave has period 4x4.

0 0 1 1  
0 1 1 0  
1 1 0 0  
1 0 0 1



All Messages

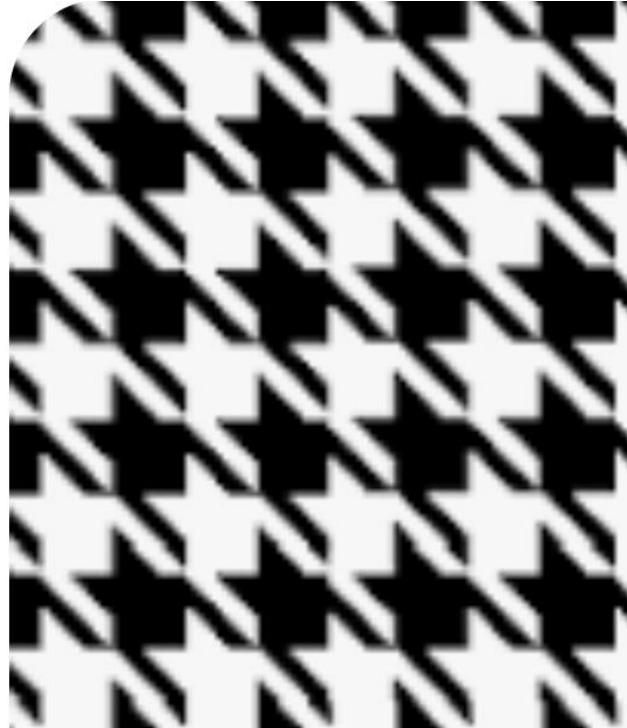
Restart File

BookBuilder.lean M BookBuilder.lean/...

1 import VersoManual  
2 import Book.Everything  
3  
4 open Verso.Genre Manual  
5 open Verso Code External  
6  
7 open Verso.Output.Html in  
8  
9  
10 def config : Config where  
11 emitTeX := false  
12 emitHtmlSingle := false  
13 emitHtmlMulti := true  
14 htmlDepth := 2  
15 sourceLink := some "https://github.com/habmeus-papadum/warpsnwefts" setup book, Nehal Pat  
16 issueLink := some "https://github.com/habmeus-papadum/warpsnwefts/issues"  
17 linkTargets := fun st => st.localTargets ++ st.remoteTargets  
18 def main := manualMain (%doc Book.Everything) (config := config.addKaTeX)  
19

# Full Demo

- Refactor into multiple files
- Multi-Modal, Paste Images
- Answer Questions
  - inline
    - e.g. “what is coeSort?”
  - Markdown / latex



theorem [ProbabilityTheory.strong\\_law\\_ae](#)

source

```
{ $\Omega$  : Type u_1} {m $\Omega$  : MeasurableSpace  $\Omega$ }  
{ $\mu$  : MeasureTheory.Measure  $\Omega$ } {E : Type u_2} [NormedAddCommGroup E]  
[NormedSpace ℝ E] [CompleteSpace E] [MeasurableSpace E]  
[BorelSpace E] (x : ℙ →  $\Omega$  → E)  
(hint : MeasureTheory.Integrable (x 0)  $\mu$ )  
(hindep :  
  Pairwise  
    (Function.onFun (fun (x1 x2 :  $\Omega$  → E) => IndepFun x1 x2  $\mu$ ) x)  
 )  
(hident : ∀ (i : ℙ), IdentDistrib (x i) (x 0)  $\mu$   $\mu$ ) :  
 ∀m ( $\omega$  :  $\Omega$ ) ∂ $\mu$ ,  
 Filter.Tendsto  
 (fun (n : ℙ) => (↑n)-1 • ∑ i ∈ Finset.range n, x i  $\omega$ )  
 Filter.atTop (nhds (f (x :  $\Omega$ ), x 0 x ∂ $\mu$ ))
```

# Manually Solving PutnamBench w/ Agents

```
# OpenAI                                bash
codex -a never exec --sandbox=danger-full-access {prompt}

# Anthropic
claude --dangerously-skip-permissions -d -p {prompt}

# Google
gemini --yolo -d -p {prompt}
```

Replace the sorry in 'solutions\_replaced\_new/putnam\_1965\_a5\_sol.lean' with a valid proof.  
Toggle sidebar ⌘B  
Compile the file using `lake env lean solutions\_replaced\_new/putnam\_1965\_a5\_sol.lean` to make sure your changes are valid, does not produce errors, and that all sorries are removed. If you run `lake build` on the entire project, you will definitely see that many other files have sorries. This is ok.

<important>

You are NOT allowed to introduce axioms or modify any files other than 'solutions\_replaced\_new/putnam\_1965\_a5\_sol.lean'. You are not allowed to change any supporting definitions. You may only complete the proof stated in `solutions\_replaced\_new/putnam\_1965\_a5\_sol.lean`.

</important>

```
# Compile
lake env lean -o Solution.olean solutions_replaced_new/putnam_1962_b1_sol.lean

# SafeVerify (https://github.com/GasStationManager/SafeVerify)
lake env lean --run Verify.lean Problem.olean Solution.olean
```

30 minute budget, ~\$1-4 per problem

## SafeVerify Failures – Mostly `native Decide`

	agent	model	compiles	safeverified
0	claude	opus-4.5	234	203
2	codex	gpt-5.1	92	90
1	claude	sonnet-4.5	45	39

```
have hgcd : Nat.gcd 90 92 = 2 := by native Decide
```

lean

```
→ theorem putnam_1990_a6 :  
  ((Finset.univ : Finset <| Finset (Set.Icc 1 10) × Finset (Set.Icc 1 10)).filter  
   fun ⟨S, T⟩ ↦ (∀ s ∈ S, T.card < s) ∧ (∀ t ∈ T, S.card < t)).card =  
  ((17711) : ℕ) :=
```

by

-- The search space is finite (only  $2^{10}$  choices for each subset), so we verify the count by  
native Decide

# Agentic Internals: Tool Calling

<b>tool_name</b>	<b>count</b>	<b>duration</b>
shell_command	52	0.002
update_plan	2	0
<b>command_name</b>	<b>count</b>	<b>duration</b>
rg	26	0
sed	16	0
lake	6	17.6
nl	2	0
ls	1	0
tail	1	0

# “Premise Selection”

```
rg -n "num_pos" -g"*.lean" -n .
```

0.00 seconds

**Output:**

```
Exit code: 1
```

```
Wall time: 0 seconds
```

```
Output:
```

# “Premise Selection”

```
rg -n "num_pos" .lake/packages/mathlib -m 20
```

0.00 seconds

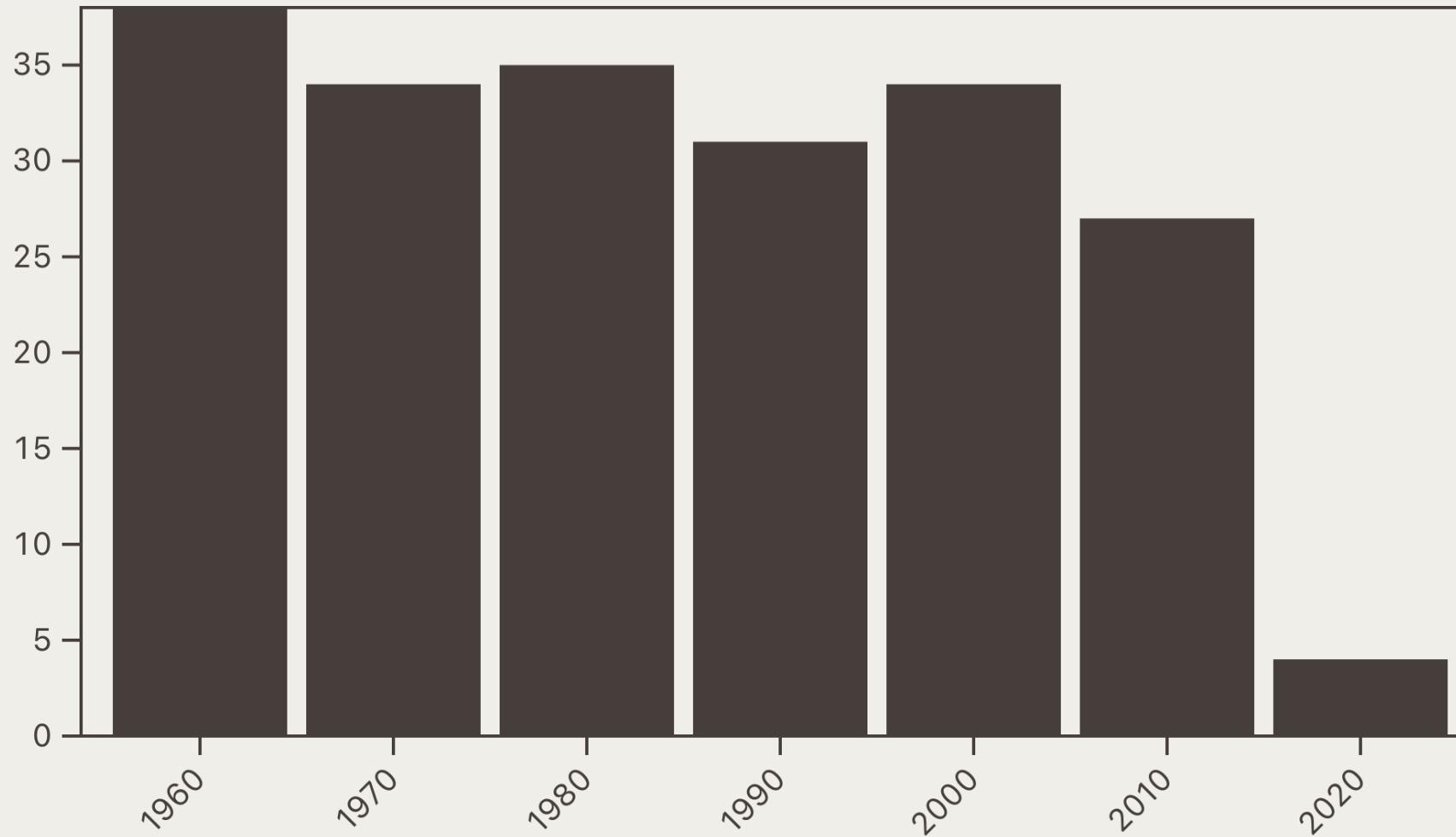
```
rg -n "num_pos" .lake/packages/mathlib -m 20
```

Output:

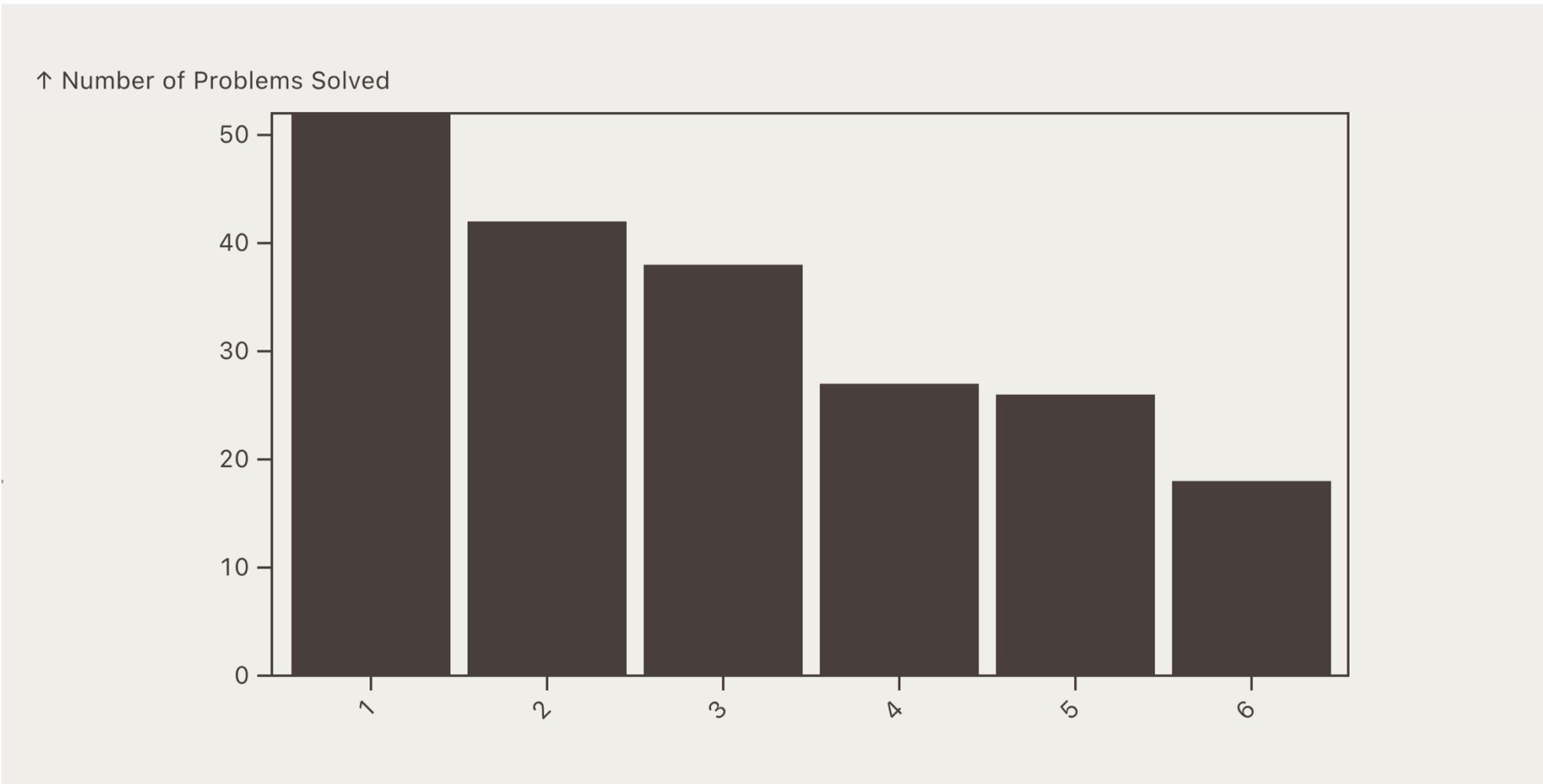
```
.lake/packages/mathlib/MathlibTest/linarith.lean:188: linarith [Rat.num_pos.mpr hx, h]
.lake/packages/mathlib/MathlibTest/linarith.lean:196: linarith only [Rat.num_pos.mpr hx]
.lake/packages/mathlib/MathlibTest/linarith.lean:197: linarith only [Rat.num_pos.mpr hx, h]
.lake/packages/mathlib/Archive/IMO/IMO2013Q5.lean:84: have num_pos : 0 < q.num := Rat.num_pos.mpr hq
.lake/packages/mathlib/Archive/IMO/IMO2013Q5.lean:87:   (0 : ℝ) < q.num := Int.cast_pos.mpr num_pos
.lake/packages/mathlib/Archive/IMO/IMO2013Q5.lean:88:   _ = ((q.num.natAbs : ℤ) : ℝ) := congr_arg
.lake/packages/mathlib/Archive/IMO/IMO2013Q5.lean:89:   _ ≤ f q.num.natAbs := (H4 q.num.natAbs ((@
.lake/packages/mathlib/Archive/IMO/IMO2013Q5.lean:90:   _ = f q.num := by rw [Nat.cast_natAbs, abs
.lake/packages/mathlib/Archive/IMO/IMO2013Q5.lean:237: have : 0 < x.num := by rwa [Rat.num_pos]
.lake/packages/mathlib/Mathlib/Algebra/Order/Ring/Unbundled/Rat.lean:222:@[simp] lemma num_pos {a : ℚ
.lake/packages/mathlib/Mathlib/Data/Real/Embedding.lean:195: exact lt_csSup_of_lt (ratLt'_bddAbove (
.lake/packages/mathlib/Mathlib/Tactic/Positivity/Basic.lean:449:private alias (_, NNRat.num_pos_of_pos
.lake/packages/mathlib/Mathlib/Tactic/Positivity/Basic.lean:462:   | .positive pa => return .positiv
.lake/packages/mathlib/Mathlib/Tactic/Positivity/Basic.lean:486:private alias (_, num_pos_of_pos) :=
.lake/packages/mathlib/Mathlib/Tactic/Positivity/Basic.lean:500:   | .positive pa => pure <| .positiv
.lake/packages/mathlib/Mathlib/Data/NNRat/Defs.lean:325:@[simp] lemma num_pos : 0 < q.num ↔ 0 < q :=
.lake/packages/mathlib/Mathlib/Data/Rat/Cast/Order.lean:29: exact div_pos (Int.cast_pos.2 <| num_pos
.lake/packages/mathlib/Mathlib/Data/Rat/Floor.lean:269: have q_num_pos : 0 < q.num := Rat.num_pos.mp
.lake/packages/mathlib/Mathlib/Data/Rat/Floor.lean:271: have q_num_abs_eq_q_num : (q.num.natAbs : ℤ)
.lake/packages/mathlib/Mathlib/Data/Rat/Floor.lean:276:   field_simp [q_inv, this, ne_of_gt q_num_pos
.lake/packages/mathlib/Mathlib/Data/Rat/Floor.lean:281:     mod_cast Rat.num_div_eq_of_coprime q_n
.lake/packages/mathlib/Mathlib/Data/Rat/Floor.lean:283:     simp only [Nat.cast_natAbs, abs_of_nonneg
.lake/packages/mathlib/Mathlib/Data/Rat/Floor.lean:298:     · exact mod_cast Rat.num_div_eq_of_coprime
.lake/packages/mathlib/Mathlib/Data/Rat/Floor.lean:301:       exact mod_cast Rat.den_div_eq_of_coprime
```

# Claude Code, Opus 4.5, by Decade

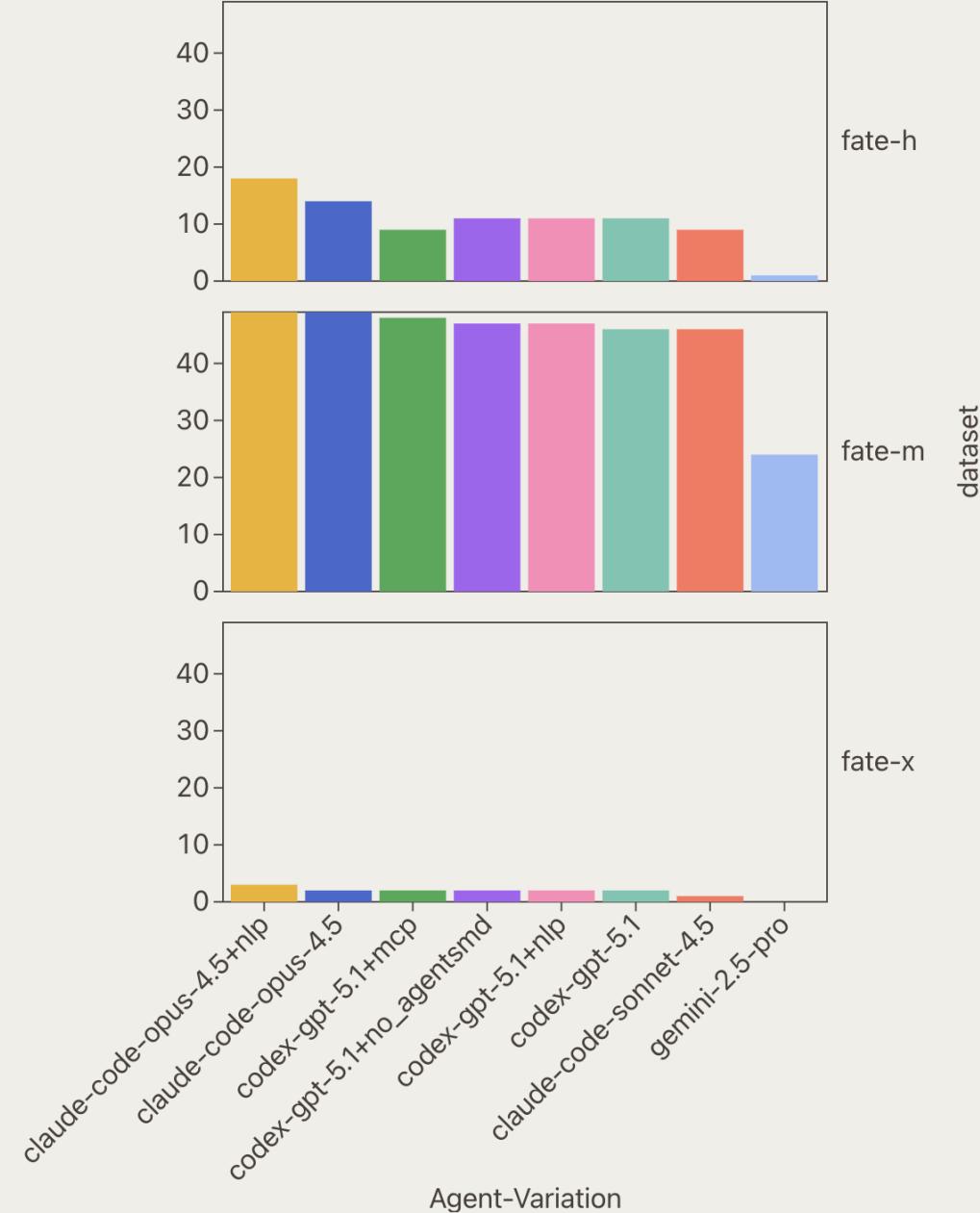
↑ Number of Problems Solved



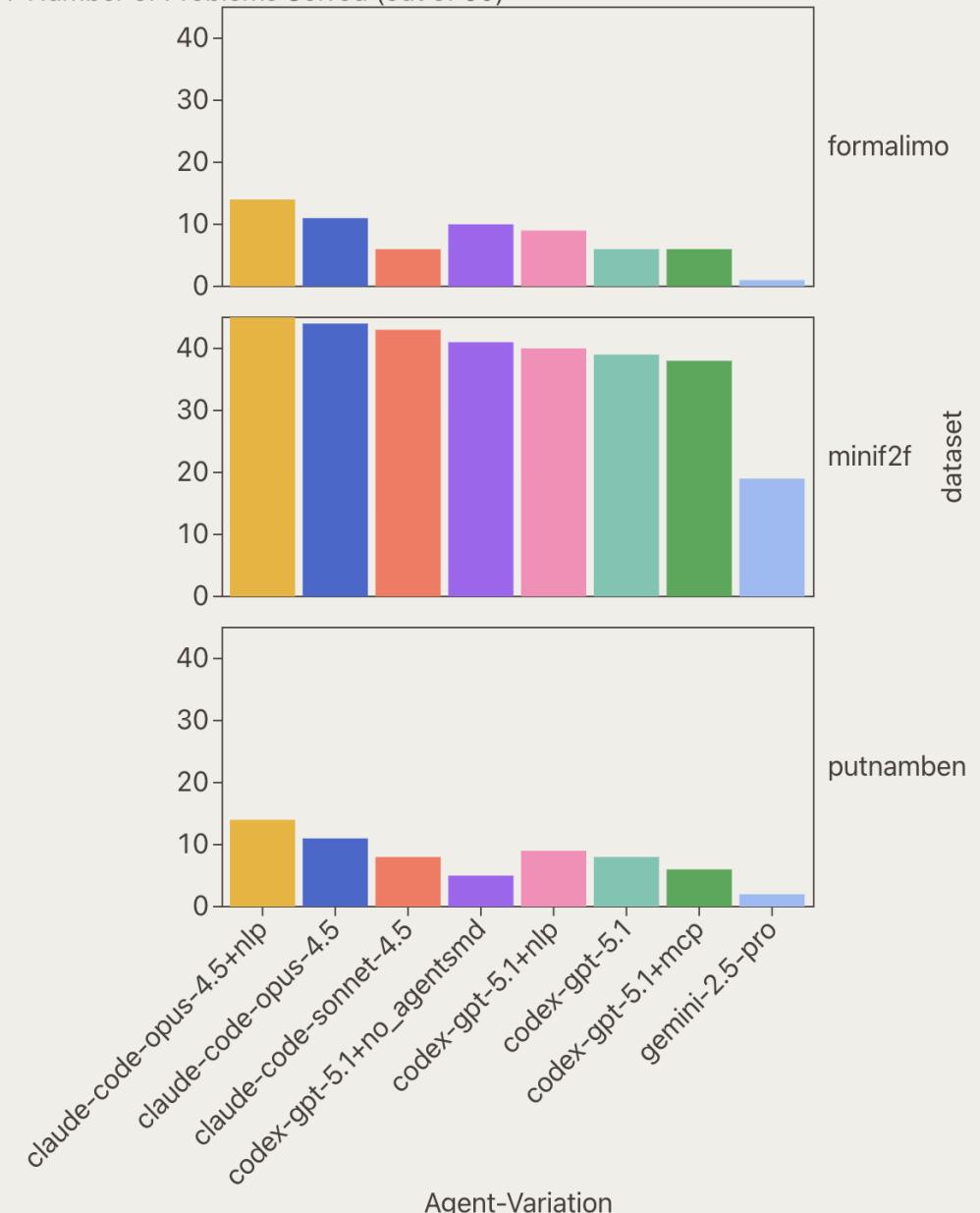
# Claude Code, Opus 4.5, by Difficulty



↑ Number of Problems Solved (out of 50)



↑ Number of Problems Solved (out of 50)



# Minimal Perturbation: Natural Language Proof

Check Type			
<b>natural_language_proof</b>	<b>compiles_sorry_free</b>	<b>safe_verified</b>	
true		225	213
false		234	203

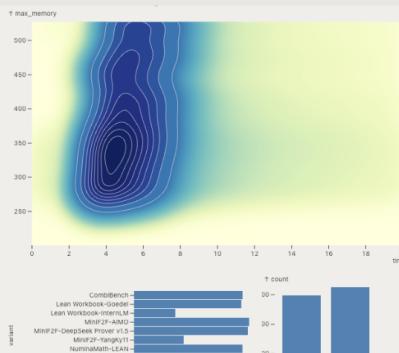
## Welcome!

The [G-Research Technology Innovation Group](#) scouts emerging technologies that have the potential to transform quantitative finance.

In this blog, we share some of our early explorations into [Lean 4](#) (an interactive theorem prover and programming language) and large language models (LLMs) trained to work with Lean 4. Our goal is to introduce readers to some of the practical insights and background knowledge we've gathered—things that aren't always easy to find, but that help fill in important gaps when entering this field. Over the past few months, we've been reading papers, experimenting with datasets, and running our own model tests in collaboration with academic researchers, and these posts capture some of the lessons learned along the way. While our long-term aim is to publish more formal research, this blog offers a snapshot of the ideas and experiments that we've been developing to date.

### Datasets

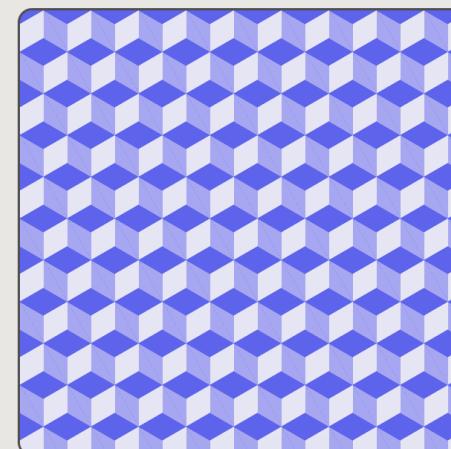
Explore various datasets used for training and evaluating Lean 4 models.



### Lean 4 LLM Internals

Coming soon...

Dissect the internals of current Lean 4 LLMs.



### The Three Horsemen

The Three Horsemen ([claude](#), [codex](#) & [gemini](#)) know how to Lean...

agent	model	solved
claude	opus-4.5	203
codex	gpt-5.1	90
claude	sonnet-4.5	39

### Repl Cryogenics

Using [CRIU](#) for REPL checkpointing.

```
> echo "magic incantation" && export UV_GDF_20_RUNNING=0  
magic incantation  
> echo "magic incantation" && repl1 && luke env .....,/lukae/build/bin/repl  
Launch a repl  
(Default: /opt/lukae/build/bin/repl)  
(Default: Import Mathbin/Algebra.BigOperators/Reals/Reals/Nat)  
(Default: 0)  
(*end*: *theorem mathd_algebra_450:n (x : Nat)� (h : 2 * (2 * (2 * (2 * x))) = 48) :n x = 3 := by sorry*, "env": N)
```

# Future Directions

- Improve the Baseline
  - MCP, Skills, Delegation
- Real Formalization
  - In Collaboration
  - First Step: Redo prior work
- Other Modalities
  - Writing Tactics
  - Conjectures
  - Syntax
  - Pedagogy
    - Lean 4 Tutoring
    - Warps'n'Wefts