So, Computers can Prove Theorems (in Lean), What's Next?

Alex J. Best

Harmonic

December 9, 2025

Outline

- Computers can already prove non-trivial theorems in systems like Lean.
- Recent AI systems (e.g. Aristotle) can autonomously produce multi-thousand-line formal proofs.

Outline

- Computers can already prove non-trivial theorems in systems like Lean.
- Recent AI systems (e.g. Aristotle) can autonomously produce multi-thousand-line formal proofs.
- Which parts of your workflow should change, and which still need a human input?

Outline

- Computers can already prove non-trivial theorems in systems like Lean.
- Recent AI systems (e.g. Aristotle) can autonomously produce multi-thousand-line formal proofs.
- Which parts of your workflow should change, and which still need a human input?
- What new projects can we now take on, if we use these systems effectively?

Outline

- Computers can already prove non-trivial theorems in systems like Lean.
- Recent AI systems (e.g. Aristotle) can autonomously produce multi-thousand-line formal proofs.
- Which parts of your workflow should change, and which still need a human input?
- What new projects can we now take on, if we use these systems effectively?
- What adjustments are needed to integrate these new sources of formal proofs into libraries like mathlib?

What sort of theorems can computers prove?

- **Classical bespoke automation:**
  - `linarith`: applies a well-known algorithm for linear (in)equalities.
  - Highly reliable, but limited to a narrow class of problems.

What sort of theorems can computers prove?

- **Classical bespoke automation:**
  - `linarith`: applies a well-known algorithm for linear (in)equalities.
  - Highly reliable, but limited to a narrow class of problems.
- **Powerful but hand-written procedures:**
  - `grind` and similar tactics: large, extensible search procedures.
  - Still fundamentally based on algorithms we designed by hand.
  - Uses the above as a building block

What sort of theorems can computers prove?

- **Classical bespoke automation:**
  - `linarith`: applies a well-known algorithm for linear (in)equalities.
  - Highly reliable, but limited to a narrow class of problems.
- **Powerful but hand-written procedures:**
  - `grind` and similar tactics: large, extensible search procedures.
  - Still fundamentally based on algorithms we designed by hand.
  - Uses the above as a building block
- **AI systems:**
  - Models that search for proofs, guided by learned heuristics instead of rules.
  - Adapt via learning from unstructured data, or better yet, from experience, to continually improve.
  - Able to solve challenging problems and productively operate autonomously for far longer periods of time.
  - Often more flexible than bespoke automation or hand-written procedures. Can make interacting with a formal system closer to interacting in natural language.
  - May use the above as building blocks, but build higher level abstractions and structure on top of them.

Examples of AI-generated proofs

- Users of systems like Aristotle using it to obtain both novel proofs, and autoformalise existing arguments. Solutions to problems of Erdős via Boris Alexeev and others [1][2].

[1] https://xenaproject.wordpress.com/2025/12/05/formalization-of-erdos-problems/
[2] https://terrytao.wordpress.com/2025/12/08/the-story-of-erdos-problem-126/
[3] https://x.com/nasqret

Examples of AI-generated proofs

- Users of systems like Aristotle using it to obtain both novel proofs, and autoformalise existing arguments. Solutions to problems of Erdős via Boris Alexeev and others [1][2].

```
/-
The Erdos conjecture is true: under the given conditions, every sufficiently large integer is representable.
-/
theorem erdos_conjecture_true (k : ℕ) (d : Fin k → ℕ)
    (h_ge : ∀ i, 2 ≤ d i)
    (h_sum : 1 ≤ ∑ i, (1 : ℚ) / (d i - 1)) :
    ∀ n, ∃ a : Fin k → ℕ,
    (∀ i, ((d i).digits (a i)).toFinset ⊆ {0, 1}) ∧
    n = ∑ i, a i := by
  -- By the properties of the sequence u_seq, every natural number can be represented as a sum of its terms.
  have h_dense : ∀ n : ℕ, ∃ s : Finset ℕ, n = ∑ j ∈ s, u_seq d j := by
    -- Apply Brown's criterion with the given hypotheses.
    apply browns_criterion;
    · apply u_seq_monotone;
      · -- Apply the lemma that states if the sum of reciprocals is at least 1, then k cannot be zero.
        apply k_ne_zero_of_sum_eq_one; assumption;
      · exact fun i => le_trans ( by norm_num ) ( h_ge i );
    · exact?;
    · apply_rules [ u_seq_gap ];
      · aesop;
        norm_num at h_sum;
  -- By definition of $u_seq$, each term in the sum $\sum_{j \in s} u_seq d j$ is of the form $d_i^{e_i}$ for some $i$ and $e_i$.
  have h_terms : ∀ s : Finset ℕ, ∃ a : Fin k → ℕ, (∀ i, ((d i).digits (a i)).toFinset ⊆ {0, 1}) ∧ ∑ j ∈ s, u_seq d j = ∑ i, a i := by
    -- Apply the lemma `digits_of_subset_sum_u_seq` to the set `s`.
    intros s
    apply digits_of_subset_sum_u_seq;
    · rintro rfl; norm_num at h_sum;
    · exact fun i => le_trans ( by norm_num ) ( h_ge i );
  exact fun n => by obtain ⟨ s, hs ⟩ := h_dense n; obtain ⟨ a, ha₁, ha₂ ⟩ := h_terms s; exact ⟨ a, ha₁, hs.trans ha₂ ⟩ ;
```

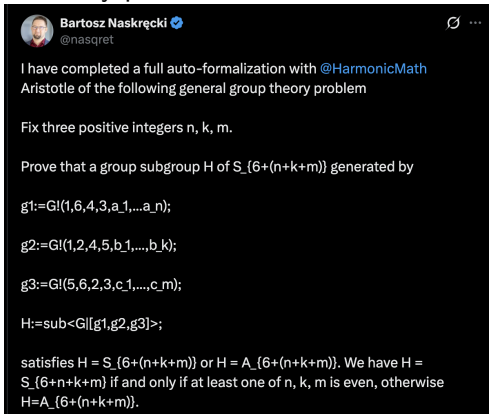
Examples of AI-generated proofs

- Users of systems like Aristotle using it to obtain both novel proofs, and autoformalise existing arguments. Solutions to problems of Erdős via Boris Alexeev and others [1][2].
- Has been used to check the output of informal models (and correct them when needed).
- Has been used to autoformalize papers "in real time", as they are written as part of the discovery process[3].

[1] https://xenaproject.wordpress.com/2025/12/05/formalization-of-erdos-problems/
[2] https://terrytao.wordpress.com/2025/12/08/the-story-of-erdos-problem-126/
[3] https://x.com/nasqret

Examples of AI-generated proofs

- Users of systems like Aristotle using it to obtain both novel proofs, and autoformalise existing arguments. Solutions to problems of Erdős via Boris Alexeev and others [1][2].
- Has been used to check the output of informal models (and correct them when needed).
- Has been used to autoformalize papers "in real time", as they are written as part of the discovery process[3].



Examples of AI-generated proofs

- Users of systems like Aristotle using it to obtain both novel proofs, and autoformalise existing arguments. Solutions to problems of Erdős via Boris Alexeev and others [1][2].
- Has been used to check the output of informal models (and correct them when needed).
- Has been used to autoformalize papers "in real time", as they are written as part of the discovery process[3].

```
2605  /-
2606  If n, k, m are all odd, H is the alternating group.
2607  -/
2608  lemma H_eq_alternating_of_all_odd (n k m : ℕ) (hn : 1 < n) (hk : 0 < k) (hm : 0 < m)
2609      (hn_odd : Odd n) (hk_odd : Odd k) (hm_odd : Odd m) : H n k m = alternatingGroup (Fin (N n k m)) := by
2610        -- If n, k, m are all odd, then all generators of H are in the alternating group.
2611        have h_all_in_A : ∀ g ∈ ({g1 n k m, g2 n k m, g3 n k m} : Set (Perm (Fin (N n k m)))), Equiv.Perm.sign g = 1 := by
2612          intro g hg
2613          have h_sign_g : Equiv.Perm.sign g = (-1 : ℤ) ^ (n + 3) ∨ Equiv.Perm.sign g = (-1 : ℤ) ^ (k + 3) ∨ Equiv.Perm.sign g = (-1 : ℤ) ^ (m + 3) := by
2614            rcases hg with ( rfl | rfl | rfl ) <;> [ exact Or.inl <| mod_cast sign_g1 n k m <| by linarith; ; exact Or.inr <| Or.inl <| mod_cast sign_g2 n k m <| by linarith; ; exact
2615          rcases h_sign_g with h | h | h <;> norm_cast at h <;> simp_all +decide [ parity_simps ];
2616        refine' le_antisymm _ _;
2617        · exact fun g hg => Subgroup.closure_induction ( fun x hx => by aesop ) ( by aesop ) ( by aesop ) ( by aesop ) hg;
2618        · apply_rules [ alternatingGroup_le_H ]
2619
2620  /-
2621  H is the symmetric group iff at least one of n, k, m is even.
2622  -/
2623  theorem H_eq_top_iff (n k m : ℕ) (hn : 1 < n) (hk : 0 < k) (hm : 0 < m) :
2624      H n k m = ⊤ ↔ (Even n ∨ Even k ∨ Even m) := by
2625        apply Iff.intro;
2626        · contrapose!;
2627          -- If n, k, m are all odd, then H n k m is equal to the alternating group.
2628          intro h_odd
2629          have h_alt : H n k m = alternatingGroup (Fin (N n k m)) := by
2630            exact H_eq_alternating_of_all_odd n k m hn hk hm ( by simpa using h_odd.1 ) ( by simpa using h_odd.2.1 ) ( by simpa using h_odd.2.2 );
2631          simp +decide [ h_alt, Subgroup.eq_top_iff' ];
2632          exact ⟨ Equiv.swap ⟨ 0, by linarith [ show N n k m > 0 from by { unfold N; linarith } ] ⟩ ( ⟨ 1, by linarith [ show N n k m > 1 from by { unfold N; linarith } ] ⟩ ), by simp +de
2633        · exact?
```

Examples of AI-generated proofs

- Users of systems like Aristotle using it to obtain both novel proofs, and autoformalise existing arguments. Solutions to problems of Erdős via Boris Alexeev and others [1][2].
- Has been used to check the output of informal models (and correct them when needed).
- Has been used to autoformalize papers "in real time", as they are written as part of the discovery process[3].
- Filling out libraries like `mathlib` and `Lean-QuantumInfo`, resolve status of unclear informal work.

[1] `https://xenaproject.wordpress.com/2025/12/05/formalization-of-erdos-problems/`
[2] `https://terrytao.wordpress.com/2025/12/08/the-story-of-erdos-problem-126/`
[3] `https://x.com/nasqret`

Examples of AI-generated proofs

- Users of systems like Aristotle using it to obtain both novel proofs, and autoformalise existing arguments. Solutions to problems of Erdős via Boris Alexeev and others [1][2].
- Has been used to check the output of informal models (and correct them when needed).
- Has been used to autoformalize papers "in real time", as they are written as part of the discovery process[3].
- Filling out libraries like `mathlib` and `Lean-QuantumInfo`, resolve status of unclear informal work.
- IMO problems, this system scored gold medal-equivalent performance with 5/6 problems solved at this year's IMO.

All except the last of these are from regular users in the Lean community making requests via `aristotle.harmonic.fun`.

[1] https://xenaproject.wordpress.com/2025/12/05/formalization-of-erdos-problems/

[2] https://terrytao.wordpress.com/2025/12/08/the-story-of-erdos-problem-126/

[3] https://x.com/nasqret

Examples of AI-generated proofs

- Users of systems like Aristotle using it to obtain both novel proofs, and autoformalise existing arguments. Solutions to problems of Erdős via Boris Alexeev and others [1][2].
- Has been used to check the output of informal models (and correct them when needed).
- Has been used to autoformalize papers "in real time", as they are written as part of the discovery process[3].
- Filling out libraries like `mathlib` and `Lean-QuantumInfo`, resolve status of unclear informal work.
- IMO problems, this system scored gold medal-equivalent performance with 5/6 problems solved at this year's IMO.

All except the last of these are from regular users in the Lean community making requests via `aristotle.harmonic.fun`. Throw in your favourite theorems, conjectures and variants (both formally and informally), and just see what happens. It might get proved, or disproved, saving you time either way.

[1] `https://xenaproject.wordpress.com/2025/12/05/formalization-of-erdos-problems/`

[2] `https://terrytao.wordpress.com/2025/12/08/the-story-of-erdos-problem-126/`

[3] `https://x.com/nasqret`

What changed: from bespoke tools to learned search

- For many years, progress came from carefully engineered automation for specific theories.

What changed: from bespoke tools to learned search

- For many years, progress came from carefully engineered automation for specific theories.
- Search was present, but heavily constrained and tuned by hand.

What changed: from bespoke tools to learned search

- For many years, progress came from carefully engineered automation for specific theories.
- Search was present, but heavily constrained and tuned by hand.
- Recent systems use large models to propose proof steps and guide search directly.

What changed: from bespoke tools to learned search

- For many years, progress came from carefully engineered automation for specific theories.
- Search was present, but heavily constrained and tuned by hand.
- Recent systems use large models to propose proof steps and guide search directly.
- The proof search policy is now learned and auto-updating, rather than hand-specified, it has a lot of mathematics baked in.

Reinforcement learning for proof search

- Main driver of recent progress: reinforcement learning (RL).

Reinforcement learning for proof search

- Main driver of recent progress: reinforcement learning (RL).
- Instead of training on fixed input/output pairs, we train a model to maximise a reward.

Reinforcement learning for proof search

- Main driver of recent progress: reinforcement learning (RL).
- Instead of training on fixed input/output pairs, we train a model to maximise a reward.
- In mathematics this reward can come from:
  - Reaching a correct numerical answer.
  - Writing code that compiles and produces the right output.
  - Producing a proof that convinces another model / passes a rubric.
  - Producing a Lean proof that solves the goal.
- This turns theorem proving into a game the model learns to play better and better.

A high-level view of Aristotle (the IMO system)

- Bridges between natural-language mathematics and formal Lean statements, uses a large model to propose informal proof steps and tactics.

A high-level view of Aristotle (the IMO system)

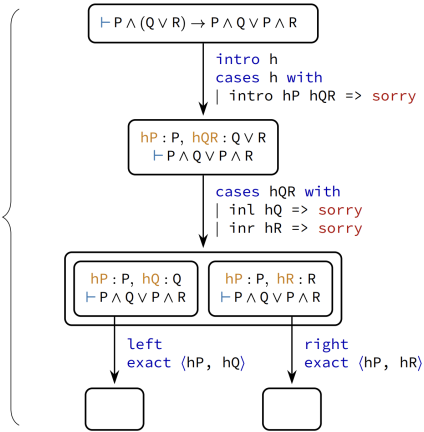
- Bridges between natural-language mathematics and formal Lean statements, uses a large model to propose informal proof steps and tactics.
- From interacting with Lean, the model gets precise feedback: goals, errors, and success/failure.
- Details are in the whitepaper (arXiv:2510.01346v2)

A high-level view of Aristotle (the IMO system)

- Bridges between natural-language mathematics and formal Lean statements, uses a large model to propose informal proof steps and tactics.
- From interacting with Lean, the model gets precise feedback: goals, errors, and success/failure.
- Details are in the whitepaper (arXiv:2510.01346v2)
- **Two main subsystems:**
  - A Lean proof search system (Monte Carlo graph search over Lean tactics).
  - A lemma-based informal reasoning system that plans, drafts, and formalises lemmas, definitions and other statements.
- A problem is only counted as solved if Aristotle produces a complete, kernel-checked Lean proof, with no gaps or `sorryAx`.

Lean proof search in Aristotle

```
intro h
cases h with
| intro hP hQR =>
  cases hQR with
  | inl hQ =>
    left
    exact ⟨hP, hQ⟩
  | inr hR =>
    right
    exact ⟨hP, hR⟩
```



Lean proof search in Aristotle

- Input: a Lean file or code block containing `sorry`s where proofs are missing.

Lean proof search in Aristotle

- Input: a Lean file or code block containing `sorry`s where proofs are missing.
- **States**: Lean proof states (goals plus local context), split when there are multiple independent goals.

Lean proof search in Aristotle

- Input: a Lean file or code block containing `sorry`s where proofs are missing.
- **States**: Lean proof states (goals plus local context), split when there are multiple independent goals.
- **Actions**: snippets of Lean code (often sequences of tactics, with optional comments) applied to a state.

Lean proof search in Aristotle

- Input: a Lean file or code block containing `sorry`s where proofs are missing.
- **States**: Lean proof states (goals plus local context), split when there are multiple independent goals.
- **Actions**: snippets of Lean code (often sequences of tactics, with optional comments) applied to a state.
- Tactics like `cases` can branch into multiple follow-up states, giving a search hypertree with AND/OR/NOT structure.

Lean proof search in Aristotle

- Input: a Lean file or code block containing `sorry`s where proofs are missing.
- **States**: Lean proof states (goals plus local context), split when there are multiple independent goals.
- **Actions**: snippets of Lean code (often sequences of tactics, with optional comments) applied to a state.
- Tactics like `cases` can branch into multiple follow-up states, giving a search hypertree with AND/OR/NOT structure.
- Semantically equivalent states are merged, so search reuses work instead of re-proving the same subgoal. Different textual tactics that lead to the same Lean state are treated as equivalent. Turns the tree into a graph.

Lean proof search in Aristotle

- Input: a Lean file or code block containing `sorry`s where proofs are missing.
- **States**: Lean proof states (goals plus local context), split when there are multiple independent goals.
- **Actions**: snippets of Lean code (often sequences of tactics, with optional comments) applied to a state.
- Tactics like `cases` can branch into multiple follow-up states, giving a search hypertree with AND/OR/NOT structure.
- Semantically equivalent states are merged, so search reuses work instead of re-proving the same subgoal. Different textual tactics that lead to the same Lean state are treated as equivalent. Turns the tree into a graph.
- A large transformer serves as both policy (which tactic to try next) and value function (how promising a state is).

Lean proof search in Aristotle

- Input: a Lean file or code block containing `sorry`s where proofs are missing.
- **States**: Lean proof states (goals plus local context), split when there are multiple independent goals.
- **Actions**: snippets of Lean code (often sequences of tactics, with optional comments) applied to a state.
- Tactics like `cases` can branch into multiple follow-up states, giving a search hypertree with AND/OR/NOT structure.
- Semantically equivalent states are merged, so search reuses work instead of re-proving the same subgoal. Different textual tactics that lead to the same Lean state are treated as equivalent. Turns the tree into a graph.
- A large transformer serves as both policy (which tactic to try next) and value function (how promising a state is).
- Search is a highly parallel Monte Carlo Graph Search (MCGS) that focuses effort on the hardest remaining subgoals that are likely to help prove the theorem.

Lemma-based informal reasoning

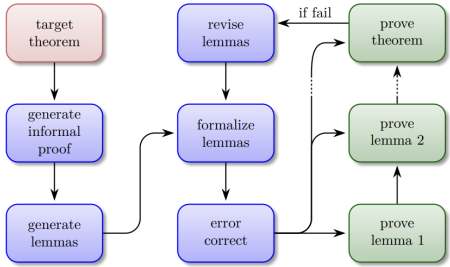
- For hard IMO problems, a separate informal pipeline sits around proof search:

Lemma-based informal reasoning

- For hard IMO problems, a separate informal pipeline sits around proof search:
  - Generate an informal proof sketch of the target theorem.
  - Break it into a sequence of short lemmas with simple proofs.
  - Autoformalise the lemma statements into Lean and fix type errors.
- The search system then tries to prove the lemmas first, reusing them when tackling the main theorem.

Lemma-based informal reasoning

- For hard IMO problems, a separate informal pipeline sits around proof search:
  - Generate an informal proof sketch of the target theorem.
  - Break it into a sequence of short lemmas with simple proofs.
  - Autoformalise the lemma statements into Lean and fix type errors.
- The search system then tries to prove the lemmas first, reusing them when tackling the main theorem.
- If the attempt fails, Aristotle revises the lemma list using formal feedback (which lemmas did/did not prove) and iterates.



Interleaving formal and informal reasoning

- Each proof step is produced by a model that thinks informally and then outputs Lean tactics plus comments.

Interleaving formal and informal reasoning

- Each proof step is produced by a model that thinks informally and then outputs Lean tactics plus comments.
- Hidden chain-of-thought, inline comments, and formal actions are trained together via RL on Lean feedback.

Interleaving formal and informal reasoning

- Each proof step is produced by a model that thinks informally and then outputs Lean tactics plus comments.
- Hidden chain-of-thought, inline comments, and formal actions are trained together via RL on Lean feedback.
- Comments are visible later in the proof and help the model maintain a coherent high-level strategy.

Interleaving formal and informal reasoning

- Each proof step is produced by a model that thinks informally and then outputs Lean tactics plus comments.
- Hidden chain-of-thought, inline comments, and formal actions are trained together via RL on Lean feedback.
- Comments are visible later in the proof and help the model maintain a coherent high-level strategy.
- From interacting with the Lean REPL, the models get: goal states, errors, messages, and success/failure of each tactic.

Interleaving formal and informal reasoning

- Each proof step is produced by a model that thinks informally and then outputs Lean tactics plus comments.
- Hidden chain-of-thought, inline comments, and formal actions are trained together via RL on Lean feedback.
- Comments are visible later in the proof and help the model maintain a coherent high-level strategy.
- From interacting with the Lean REPL, the models get: goal states, errors, messages, and success/failure of each tactic.
- This loop lets informal reasoning explore ideas, while formal verification keeps only the sound ones.

Reinforcement learning over proofs

- The core of Aristotle is a model that is trained to produce Lean proofs with reinforcement learning.

Reinforcement learning over proofs

- The core of Aristotle is a model that is trained to produce Lean proofs with reinforcement learning.
- Reward comes from search traces that successfully prove goals in Lean (and from hard failed states).

Reinforcement learning over proofs

- The core of Aristotle is a model that is trained to produce Lean proofs with reinforcement learning.
- Reward comes from search traces that successfully prove goals in Lean (and from hard failed states).
- The same model serves as a generative policy (tactics to try) and as a value function over states.

Reinforcement learning over proofs

- The core of Aristotle is a model that is trained to produce Lean proofs with reinforcement learning.
- Reward comes from search traces that successfully prove goals in Lean (and from hard failed states).
- The same model serves as a generative policy (tactics to try) and as a value function over states.
- Over time it learns which informal plans and tactic patterns tend to lead to short, successful proofs.

Aristotle at the IMO and beyond

- At the 2025 IMO, Aristotle produced formal Lean solutions for 5 of the 6 problems: gold-medal-equivalent performance.

Aristotle at the IMO and beyond

- At the 2025 IMO, Aristotle produced formal Lean solutions for 5 of the 6 problems: gold-medal-equivalent performance.
- Scaling was crucial: a $>$ 200B-parameter model, many parallel lemma pipelines, and multiple refinement iterations per problem.

Aristotle at the IMO and beyond

- At the 2025 IMO, Aristotle produced formal Lean solutions for 5 of the 6 problems: gold-medal-equivalent performance.
- Scaling was crucial: a $>$ 200B-parameter model, many parallel lemma pipelines, and multiple refinement iterations per problem.
- Test-time training (TTT) let the model improve on-the-fly from its own search traces for a given problem.

Rethinking our workflows

- Old style of working with an ITP: walk through a file lemma-by-lemma, maybe working backwards from a big goal, or forwards following notes / a paper, refactor and improvise as you go.

Rethinking our workflows

- Old style of working with an ITP: walk through a file lemma-by-lemma, maybe working backwards from a big goal, or forwards following notes / a paper, refactor and improvise as you go.
- This worked when humans thinking and typing were the main bottleneck in producing proofs.

Rethinking our workflows

- Old style of working with an ITP: walk through a file lemma-by-lemma, maybe working backwards from a big goal, or forwards following notes / a paper, refactor and improvise as you go.
- This worked when humans thinking and typing were the main bottleneck in producing proofs.
- With an AI co-worker, a better pattern is:
  - Spend some time designing the structure: main statements you want to prove, useful lemmas you know will be needed.
  - Move on to another project while the AI is working.
  - Run many of these simultaneously!
  - Get the output, analyse what happened, revise and iterate, and step in if needed.

Rethinking our workflows

- Old style of working with an ITP: walk through a file lemma-by-lemma, maybe working backwards from a big goal, or forwards following notes / a paper, refactor and improvise as you go.
- This worked when humans thinking and typing were the main bottleneck in producing proofs.
- With an AI co-worker, a better pattern is:
  - Spend some time designing the structure: main statements you want to prove, useful lemmas you know will be needed.
  - Move on to another project while the AI is working.
  - Run many of these simultaneously!
  - Get the output, analyse what happened, revise and iterate, and step in if needed.
- Think of yourself as managing a team of hard working junior collaborators, who can't do everything but are willing to try anything, and do the majority of the actual coding. And will sometimes surprise you!

Rethinking our workflows

- Old style of working with an ITP: walk through a file lemma-by-lemma, maybe working backwards from a big goal, or forwards following notes / a paper, refactor and improvise as you go.
- This worked when humans thinking and typing were the main bottleneck in producing proofs.
- With an AI co-worker, a better pattern is:
  - Spend some time designing the structure: main statements you want to prove, useful lemmas you know will be needed.
  - Move on to another project while the AI is working.
  - Run many of these simultaneously!
  - Get the output, analyse what happened, revise and iterate, and step in if needed.
- Think of yourself as managing a team of hard working junior collaborators, who can't do everything but are willing to try anything, and do the majority of the actual coding. And will sometimes surprise you!
- These collaborators will only get better over time, so learning how to use them effectively is a long term investment.

What becomes worth working on

- Many mathematically interesting but not central projects would previously not get done, due to the significant effort required.

What becomes worth working on

- Many mathematically interesting but not central projects would previously not get done, due to the significant effort required.
- With AI assistance, the effort required to formalize a given theory / result drops dramatically.

What becomes worth working on

- Many mathematically interesting but not central projects would previously not get done, due to the significant effort required.
- With AI assistance, the effort required to formalize a given theory / result drops dramatically.
- Shouldn't think only in terms of speeding up existing plans; it vastly expands the space of projects we can attempt.

What becomes worth working on

- Many mathematically interesting but not central projects would previously not get done, due to the significant effort required.
- With AI assistance, the effort required to formalize a given theory / result drops dramatically.
- Shouldn't think only in terms of speeding up existing plans; it vastly expands the space of projects we can attempt.
- We should be ambitious about what we want `mathlib` and other libraries to contain.

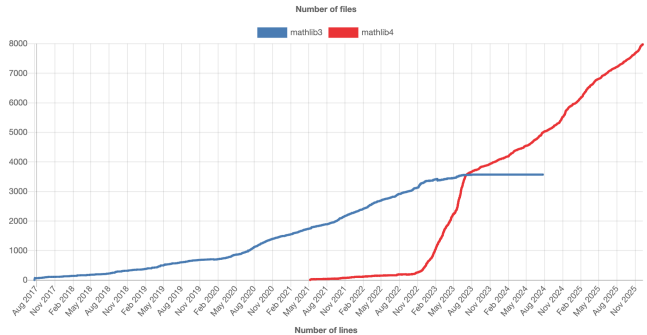
mathlib growth

Mathlib statistics

Counts

Definitions	Theorems	Contributors
121181	245960	653

Code growth



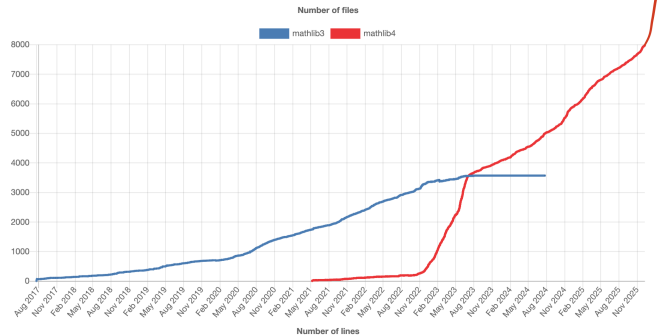
mathlib growth

Mathlib statistics

Counts

Definitions	Theorems	Contributors
121181	245960	653

Code growth



From proofs to quality control

- "To improve, we have to measure": we need clear criteria for good formal code.

From proofs to quality control

- "To improve, we have to measure": we need clear criteria for good formal code.
- Current systems already generate correct proofs; the next frontiers are speed and quality, for a reusable library we care about more than corectness.

From proofs to quality control

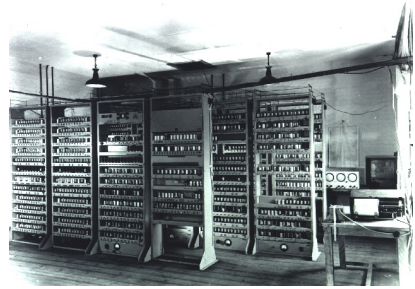
- "To improve, we have to measure": we need clear criteria for good formal code.
- Current systems already generate correct proofs; the next frontiers are speed and quality, for a reusable library we care about more than corectness.
- We should automate the most common reviewer complaints about the quality of AI-written code as much as possible, to avoid this becoming a bottleneck:
  - Style issues (forgetting dot notation, ...)
  - Unnecessary complexity
  - Missing lemmas, that should be extracted from proofs
  - Poor naming
- A great intermediate project for this week: metaprogramming and linting tools that enforce some of these rules automatically.

Computers as calculators

- The first major impact of machines on mathematical practice came from large computers as calculators.
- Example: Birch–Swinnerton-Dyer's conjecture was informed by extensive computations on an early mainframe in Cambridge.

Computers as calculators

- The first major impact of machines on mathematical practice came from large computers as calculators.
- Example: Birch–Swinnerton-Dyer's conjecture was informed by extensive computations on an early mainframe in Cambridge.
- These experiments changed what could be conjectured and tested numerically, but access was scarce: mainframes were expensive, shared, and required institutional infrastructure.
- This was a transformative tool, but not something every working mathematician could use, a very slow rollout. Not every field of mathematics was affected by this, and not every working mathematician uses computer calculation.



EDSAC at Cambridge.

The internet, arXiv, and TEX

- The second wave of automation was the combination of TEX, electronic archives, and the internet.
- Communication and dissemination sped up dramatically:
  - A paper can be written one week and read by a study group on the other side of the world the next.
  - Preprints, mailing lists, and online seminars reshaped how results spread.
  - You can email the author, ask on MathOverflow, watch talks by experts
  - Google search, Google scholar, are now essential tools for finding relevant research.
- This changed workflows: faster feedback loops, more collaboration, and broader access to cutting-edge work.
- It also normalised machine-assisted typesetting and distribution as part of everyday mathematics.

How might the future look?

- AI is a general tool, that will affect all areas of mathematics.

How might the future look?

- AI is a general tool, that will affect all areas of mathematics.
- What happens to mathematics when computers can routinely outplay us at proof search?

How might the future look?

- AI is a general tool, that will affect all areas of mathematics.
- What happens to mathematics when computers can routinely outplay us at proof search?
- Will it become less interesting, as some predicted for chess?

How might the future look?

- AI is a general tool, that will affect all areas of mathematics.
- What happens to mathematics when computers can routinely outplay us at proof search?
- Will it become less interesting, as some predicted for chess?
- Or will it become more accessible and more widely practised?

Informal vs formal LLMs, why formalisation will be key

- It is unclear whether models that are inherently purely "informal" (natural-language proofs) or "formal" (Lean proofs) learn best when doing hard mathematics.

Informal vs formal LLMs, why formalisation will be key

- It is unclear whether models that are inherently purely "informal" (natural-language proofs) or "formal" (Lean proofs) learn best when doing hard mathematics.
- Either way, in the long-run we want mathematics that humans can understand and trust: we would not want to be reading thousands of informal lines with questionable payoff, even if the model is highly reliable.

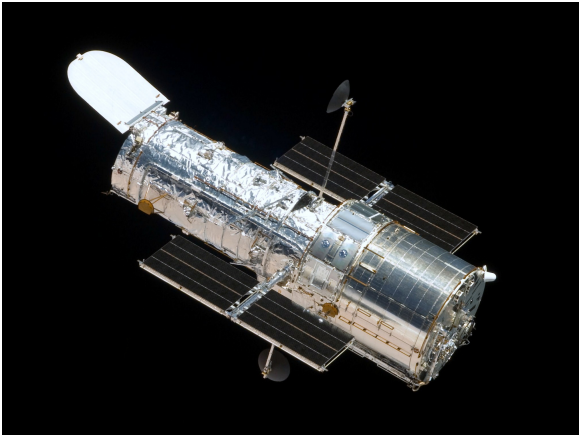
Informal vs formal LLMs, why formalisation will be key

- It is unclear whether models that are inherently purely "informal" (natural-language proofs) or "formal" (Lean proofs) learn best when doing hard mathematics.
- Either way, in the long-run we want mathematics that humans can understand and trust: we would not want to be reading thousands of informal lines with questionable payoff, even if the model is highly reliable.
- As models can produce convincing proofs faster than we can read them, we need machine-checkable guarantees either way.

Informal vs formal LLMs, why formalisation will be key

- It is unclear whether models that are inherently purely "informal" (natural-language proofs) or "formal" (Lean proofs) learn best when doing hard mathematics.
- Either way, in the long-run we want mathematics that humans can understand and trust: we would not want to be reading thousands of informal lines with questionable payoff, even if the model is highly reliable.
- As models can produce convincing proofs faster than we can read them, we need machine-checkable guarantees either way.
- That means: whatever front-end we use (chat, LaTeX, diagrams), we will still want a formal proof object underneath.

Which analogy



What should we do now?

- Take seriously the idea of working with an AI co-author in Lean. Decide on ambitious projects that we can now tackle.

What should we do now?

- Take seriously the idea of working with an AI co-author in Lean. Decide on ambitious projects that we can now tackle.
- Invest in infrastructure: more automated quality checks, tools to manage AI generated contributions to Lean libraries.

What should we do now?

- Take seriously the idea of working with an AI co-author in Lean. Decide on ambitious projects that we can now tackle.
- Invest in infrastructure: more automated quality checks, tools to manage AI generated contributions to Lean libraries.
- Experiment with new workflows and interfaces: outlines and informal instructions first, then AI-assisted proof search at scale, where in the loop do humans want to be?

What should we do now?

- Take seriously the idea of working with an AI co-author in Lean. Decide on ambitious projects that we can now tackle.
- Invest in infrastructure: more automated quality checks, tools to manage AI generated contributions to Lean libraries.
- Experiment with new workflows and interfaces: outlines and informal instructions first, then AI-assisted proof search at scale, where in the loop do humans want to be?
- And most importantly: decide, as a community, what kind of future for (formal) mathematics we want.