Prove2Me: An Open Collaborative Platform for Scaling Math Formalization

Shuze Chen *, Xiaoyang Lu, Kunal Marwaha, Tianyi Peng *, Henry Yuen

Posted Date: 25 August 2026

doi: 10.20944/preprints202608.1796.v1

Keywords: AI agents; formal verification; agentic platform; multi-agent collaboration

 doi:10.20944/preprints202608.1796.v1

Article

Prove2Me: An Open Collaborative Platform for Scaling Math Formalization

Shuze Chen [1,*], Xiaoyang Lu [2], Kunal Marwaha [3], Tianyi Peng [1,*] and Henry Yuen [4]

[1] Graduate School of Business, Columbia University, New York, NY 10027
[2] Department of Computer Science, Purdue University, West Lafayette, IN 47907
[3] Kunal Marwaha, Department of Computer Science, University of Chicago, Chicago, IL 60637
[4] Department of Computer Science, Columbia University, New York, NY 10027
* Correspondence: shuze.chen@columbia.edu

Abstract

Proof assistants such as Lean 4 promise the paradigm of formally verified mathematics, but large-scale formalization projects have faced major barriers to entry, including the need for expertise in formal verification (as well as the underlying mathematics) and the significant time required for writing formal proofs. AI coding agents have dramatically reduced these barriers; human users can now use natural language to prompt agents to write complex proofs in Lean. This opens up the intriguing possibility of internet-scale mathematical collaboration involving both humans and AI agents, where correctness is machine-checked. To realize this possibility, we introduce **Prove2Me**, an open collaborative platform for formalizing mathematics. Users launch formalization "missions", to which AI agents contribute formal proofs toward completion. We designed mechanisms and a specialized harness in Prove2Me that enable large-scale collaboration so that agents can build on one another's work and freely reuse existing results. In doing so, Prove2Me aims to turn math formalization into a scalable, crowd-sourced effort open to anyone with an agent.

Keywords: AI agents; formal verification; agentic platform; multi-agent collaboration

1. Introduction

The dream of a fully formalized mathematical corpus, where every theorem is machine-checked and every proof step is verifiable, has long been limited by the sheer human effort required. Fields Medalist Peter Scholze's Liquid Tensor Experiment required roughly eighteen months of sustained community effort (Commelin et al., 2022), and the ongoing formalization of Fermat's Last Theorem is funded for five years, with its leader Kevin Buzzard acknowledging that he "cannot formalize it alone" (Buzzard, 2024). In these cases, formalization has long been confined to a small community of specialists who had to be simultaneously expert in mathematics and in proof-assistant engineering.

Recent progress in AI-driven theorem proving, however, offers a turning point. AI systems are improving rapidly at formal mathematics (Achim et al., 2025; Hubert et al., 2026; Song et al., 2024; Yang et al., 2023,2), and a growing body of work harnesses AI agents to formalize research papers (Bei et al., 2026; Garg, 2026; Ren et al., 2026) and textbooks (Gloeckle et al., 2026; Rammal et al., 2026; Urban, 2026; Wang et al., 2026). These efforts provide a promising signal that, with multiple AI agents working together, formalization tasks that previously took months or even years can be substantially accelerated.

Yet existing efforts still face the following obstacles. The first is *auditing*: despite LLM-assisted methods (Garg, 2026; Wang et al., 2026), deciding whether a formalization faithfully captures its intended meaning still requires human expertise, which does not scale to AI-generated projects that can contain hundreds of thousands of theorems (Gloeckle et al., 2026; Urban, 2026). The second is *reusability*: prior Lean formalizations are typically hosted on GitHub as tightly interdependent theorems, making it hard to extract, reuse, or import individual results in isolation. The third is *scale*:

 doi:10.20944/preprints202608.1796.v1

although swarms of thousands of agents can already produce impressive results (Gloeckle et al., 2026; Rammal et al., 2026), they rely on a single organization's internal compute. In principle, anyone with an AI agent should now be able to contribute. With the right collaboration protocol and incentives, formalization could tap the collective token budget of the general public.

To fully unleash and scale the power of AI in formal verification, we introduce **Prove2Me**, an open-source collaborative platform for math formalization at scale, with AI agents as first-class participants. Prove2Me's contributions are fourfold: (1) a *low barrier to entry*: equipped with an AI agent, anyone can contribute Lean proofs without expertise in Lean or even in the underlying mathematics; (2) a *mission-based design* that confines human auditing to a small curated core of statements, freeing AI agents to generate intermediate theorems at scale; (3) a *collaboration mechanism* that enables the decomposition of complex proofs into atomized tasks so that agents naturally build on one another's work; and (4) a *reusable library* that grows organically, as completed proofs become citable building blocks and unsolved sub-problems surface as new challenges.

Our vision for Prove2Me is leveraging the crowd-sourced efforts of AI agents to formalize math projects at scale. The rest of the paper is organized as follows. Section 2 reviews related work. Section 3 presents the basic workflow of submitting and solving problems on Prove2Me, together with our type-based verification design. Section 4 introduces *proof-sketches*, the key mechanism that enables decomposing a hard theorem into atomized sub-problems and allows cross-agent collaboration and reuse. Section 5 reports on missions the community has completed on the platform. Section 6 gives a tutorial on contributing to Prove2Me, which requires no expertise in Lean or even in the underlying mathematics.

2. Related Work

AI for theorem proving.

Since GPT-f first contributed machine-generated proofs to a formal library (Polu and Sutskever, 2020), rapid advances (including retrieval-augmented provers (Yang et al., 2023), the DeepSeek-Prover series reaching 88.9% on miniF2F (Ren et al., 2025; Xin et al., 2024,2), AlphaProof's silver-medal IMO performance (Hubert et al., 2026), IMO-level systems such as Aristotle (Achim et al., 2025), and workflow-integrated tactic suggestion (Song et al., 2024)) demonstrate that AI agents are increasingly capable of generating formal proof scripts; we refer the reader to Yang et al. (2024) for a comprehensive survey. Yet these capabilities remain siloed in individual research pipelines, with no shared venue where agents and humans collaboratively attack open formalization challenges.

AI-assisted formalization.

Building on these provers, a recent wave of work applies AI agents to formalize existing corpora end to end. One line targets textbooks and foundational theory: a single coding agent formalizes point-set topology in roughly 130k lines of Lean (Urban, 2026), while multi-agent swarms scale further, formalizing a 500-page graduate combinatorics textbook in about a week (Gloeckle et al., 2026) and assembling libraries that span dozens of textbooks (Rammal et al., 2026; Wang et al., 2026). A second line targets individual research papers, from economics and computation (Bei et al., 2026; Garg, 2026) to quantum computation (Ren et al., 2026) and physics (Meadows et al., 2026). Despite the promising speed at which AI generates Lean proofs, these efforts still suffer from the following limitations that motivate Prove2Me.

Auditing.

The proof kernel certifies that a proof inhabits a statement, but not that the statement faithfully captures the intended claim; an AI-generated statement can be vacuous, miss a hypothesis, or drift semantically, yet still be "proved" (Gloeckle et al., 2026). Because automated faithfulness checks remain imperfect (a recent Lean-as-judge audit finds only about 43% of proved statements faithful (Bourigault et al., 2026)), a common and effective mitigation is to shrink the human audit surface, reviewing

 doi:10.20944/preprints202608.1796.v1

only the definitions and top-level statements rather than the proofs (Garg, 2026; Gloeckle et al., 2026). Prove2Me pushes this principle further by fixing the audit surface in advance: humans review a mission's curated core, its goal statement, the definitions it rests on, and the milestone lemmas that structure its proof, while agents introduce the remaining intermediate theorems freely.

Reusability and cost.

Existing formalizations are typically released as monolithic Git repositories of tightly interdependent theorems. Because Lean recompiles the entire downstream cone whenever a module changes, even for a proof-only edit, integration is serialized through a single merge queue that becomes the empirical bottleneck of large swarms (Gloeckle et al., 2026; Rammal et al., 2026); individual results are correspondingly hard to extract and reuse out of context. Finally, these swarms run on a single organization's internal compute, with tens of thousands of agents and five-figure budgets per project (Gloeckle et al., 2026; Rammal et al., 2026), leaving the collective resources of the broader community untapped. In this work, we design a harness and mechanisms that enable smooth reuse of proved results on the platform, as well as show case studies of comparable formalization projects that can be accomplished with even a few agents.

3. Basic Design of Prove2Me

Prove2Me is built to host verification that is both efficient and scalable. Its basic design decision is to *separate the statement of a theorem from its proofs*: each theorem is a standalone, immutable object that is stated once and may have multiple proofs, possibly by different agents. This separation underlies every feature described in the rest of the paper. We begin with the two basic actions on the platform, submitting a theorem (Section 3.1) and submitting a proof (Section 3.2), and then introduce *audited missions* (Section 3.3), the mechanism that keeps an agent-generated library trustworthy.

3.1. Submitting a Theorem

As a running example, we take the lemma below from the famous proof of the Sensitivity Conjecture (Huang, 2019), and follow it from its natural-language form to a theorem hosted on the platform.

Lemma 1. *Suppose H is an m-vertex undirected graph, and A is a symmetric adjacency matrix whose entries are in $\{-1, 0, 1\}$. Then the matrix degree of H satisfies*

$$\Delta(H) \geq \lambda_1 := \lambda_1(A).$$

On Prove2Me, this lemma is formalized in Lean 4 and stored as `max_degree_ge_lambda_max`, whose *theorem card* contains the following fields:

- DESCRIPTION: a natural-language account of the mathematics behind the statement, similar to Lemma 1. Agents are also encouraged to spell out the intended formulation in detail.
- PREAMBLE: the imports, shown in the upper half of Figure 1. In general, this field may contain imports from not only Mathlib but other definition files hosted on Prove2Me as well.
- FORMAL STATEMENT: the target statement itself formalized in Lean 4, shown in the lower half of Figure 1, which is required to terminate in a `:= by sorry` placeholder.

A complete submission can additionally carry a SOURCE field (a link to the originating paper or textbook) and TAGS that place the theorem in a subject classification. The complete theorem card of Lemma 1 can be found at its theorem link. Finally, Prove2Me supports multiple verification environments, each pinned to a specific Mathlib and toolchain version, and the submitting agent selects the target environment. A theorem is accepted once it compiles in the corresponding backend. Environments are fully isolated from one another, so that a result always carries the exact context needed to reproduce it.

 doi:10.20944/preprints202608.1796.v1


```
-- Preamble
import Mathlib.Analysis.Matrix.Spectrum
import Mathlib.LinearAlgebra.Matrix.Hermitian
import Mathlib.Data.Fintype.Basic
import Mathlib.Data.Real.Star

-- Formal statement
theorem max_degree_ge_lambda_max
    {V : Type*} [Fintype V] [DecidableEq V]
    {A : Matrix V V ℝ} (hA : A.IsHermitian)
    (h_entries : ∀ u v : V, A u v = -1 ∨ A u v = 0 ∨ A u v = 1)
    (adj : V → V → Prop) [DecidableRel adj]
    (h_zero : ∀ u v : V, ¬ adj u v → A u v = 0)
    [Nonempty V] :
    ∃ v : V, hA.eigenvalues₀ ⟨0, Fintype.card_pos⟩
      ≤ ((Finset.univ : Finset V).filter fun u => adj u v).card := by
    sorry
```

Figure 1. The PREAMBLE and FORMAL STATEMENT fields of the theorem card for `max_degree_ge_lambda_max`, a Lean 4 formalization of Lemma 1. The statement is posed for a bare adjacency predicate `adj` rather than a `SimpleGraph`, which keeps it directly applicable to the hypercube subsets that the sensitivity proof needs; choices of this kind are what the DESCRIPTION field records and what auditing (Section 3.3) checks.

3.2. Submitting a Proof

Because statement and proof are separate objects, a single theorem may have many independent proofs. To check a proof against a statement, Prove2Me relies on the Curry–Howard correspondence (Wadler, 2015) at the heart of Lean 4: a proof of a proposition is a term whose *type* is that proposition. Concretely, a proof submission is a Lean file that declares a theorem named `solution`, whose type matches that of the target theorem exactly and which contains no `sorry` or any other new axioms.

Figure 2 illustrates this for the theorem `max_degree_ge_lambda_max` of Figure 1: the submission reuses the target's type verbatim and supplies a term inhabiting it. The platform then compiles the submission in the same environment as the target theorem and checks that the type of `solution` matches the type of the target statement, so that the two are interchangeable as Lean terms. Symmetrically, an agent may submit a *disproof* of a theorem by providing a `theorem solution` whose type is ¬(target statement), i.e., the negation of the original theorem. Alongside the proof file, agents are required to upload a detailed natural-language explanation of the proof idea, which is linked to the proof for the benefit of human readers and other agents.

 doi:10.20944/preprints202608.1796.v1


```
import Mathlib.Data.Matrix.Mul
open Matrix

/-! Full proof of ‘max_degree_ge_lambda_max‘.
    A standard "max-coordinate" Perron-style argument. -/

theorem solution
    {V : Type*} [Fintype V] [DecidableEq V]
    {A : Matrix V V ℝ} (hA : A.IsHermitian)
    (h_entries : ∀ u v : V, A u v = -1 ∨ A u v = 0 ∨ A u v = 1)
    (adj : V → V → Prop) [DecidableRel adj]
    (h_zero : ∀ u v : V, ¬ adj u v → A u v = 0)
    [Nonempty V] :
    ∃ v : V, hA.eigenvalues₀ ⟨0, Fintype.card_pos⟩
      ≤ ((Finset.univ : Finset V).filter fun u => adj u v).card := by
  -- Bridge ‘eigenvalues₀ ⟨0, _⟩‘ to ‘eigenvalues j‘ for some ‘j : V‘.
  set i0 : Fin (Fintype.card V) := ⟨0, Fintype.card_pos⟩
  set j : V := (Fintype.equivOfCardEq (Fintype.card_fin _)) i0 with hj_def
  -- ... remaining proof, sorry-free.
```

Figure 2. A proof submission for the max-degree bound of Lemma 1 (abridged). The file declares `theorem solution` with exactly the type of the target `max_degree_ge_lambda_max` and closes it without `sorry`.

3.3. *Audited Missions*

Because anyone with an AI agent can submit a theorem, statements may be false, ill-defined, or hallucinated. This is especially a risk on a platform whose content is largely agent-generated. Recent work has attempted automated auditing of formalized statements (Garg, 2026; Gloeckle et al., 2026; Ren et al., 2026; Wang et al., 2026), but providing a strong guarantee that a formalization faithfully captures its intended meaning still generally requires human auditing.

Human auditing, however, does not scale, whereas our goal is to let agents generate ever more content. Prove2Me resolves this tension with **missions**. A mission consists of a headline goal (for example, the main theorem of a project, paper, or book chapter), the definitions it depends on, and the milestone lemmas that structure its proof (Section 4.4). Humans audit this core before the mission is released, checking that each of its formal statements faithfully captures the intended mathematics, and audit nothing beyond it. Agents then supply the remaining proof detail by freely introducing intermediate lemmas without audit. This is sound because the trusted object is not the agent's chosen decomposition, but the Lean kernel's acceptance of proofs of the audited statements. Intermediate lemmas matter only insofar as they help close audited goals. Thus, human auditing is bounded by the mission core, while open-ended proof elaboration scales with agent effort, in line with recent AI-assisted formalization practice (Garg, 2026; Gloeckle et al., 2026).

Captaining a mission.

A mission is created by a *captain*, the human contributor who proposes it. Captains need not write Lean: the captain's agent drafts the *mission proposal*, a private bundle of the goal theorem, its definitions, and the milestone lemmas (Section 4.4). The mission proposal stays *mutable* so an ill-formalized statement can still be fixed before being formally published to the platform. What the captain cannot delegate is the audit: the human is required to click each statement individually to confirm it, ensuring none enters the audited core unread. The confirmed drafts are then compiled and published as immutable theorems, and a platform moderator reviews the mission before it goes live.

Sub-agent read-back.

Auditing faithfulness usually requires reading Lean, which limits participation to formalization experts. Prove2Me lowers this barrier with a **sub-agent read-back** step. For each candidate statement, an independent auditor agent is given the Lean declaration and its dependent definitions, but not the original source reference. The agent translates the Lean code back into ordinary LaTeX mathematics,

 doi:10.20944/preprints202608.1796.v1

making binders and hypotheses explicit and unfolding non-standard definitions where needed. The human auditor then compares two mathematical statements: the source LaTeX statement and the agent's read-back from Lean. Faithfulness is judged by whether these two statements match, rather than by requiring the auditor to inspect Lean directly. Garg (2026) apply a related back-translation check. Figure 3 shows a read-back for a statement drafted in a mission on Erdős problem 390.

PREAMBLE (LEAN)

```
import Definitions.Def_erdos390_problem
```

FORMAL STATEMENT (LEAN)

```
namespace Erdos390

/-- The complement formulation immediately following the main theorem. -/
theorem complement_formulation {n M : ℕ} (hnM : n < M) :
    IsAdmissibleEndpoint n M ↔ HasComplementProduct n M := by sorry

end Erdos390
```

READ-BACK: What the Lean code literally says, in plain math · gpt-5.6-sol

For all natural numbers n, M, if $n < M$, then the following two existence statements are equivalent: first, there exists a finite set $A \subseteq \{k \in \mathbb{N} \mid n < k \leq M\}$, with distinct elements, whose natural-number product satisfies $\prod_{a \in A} a = n!$; second, there exists a finite set $B \subseteq \{k \in \mathbb{N} \mid n < k \leq M\}$, with distinct elements, whose natural-number product, after embedding into the rationals, satisfies $\left(\prod_{b \in B} b : \mathbb{Q}\right) = \frac{M!}{(n!)^2}$. The two finite sets are existentially quantified independently: the statement does not require them to be equal, complementary, or otherwise directly related. Either set may be empty, and an empty product is 1. The hypothesis requires only $n < M$, not $n \geq 3$, so it includes the boundary case $n = 0$ and the convention $0! = 1$. Rational division is totalized, but $(n!)^2 \neq 0$, so this quotient has a nonzero denominator.

Figure 3. A read-back of a drafted statement. The auditor agent sees only the Lean code (top) and renders what it literally asserts (bottom), unfolding the two problem-specific definitions and accounting for every binder and hypothesis, including those a source statement would leave implicit: that the two finite sets are quantified independently, that an empty product is 1, and that the hypothesis is only $n < M$. The human auditor compares this testimony against the source statement.

A mission usually involves hundreds or even thousands of supporting theorems and lemmas. This design thus keeps the audited core, and with it the mission's headline goal, trustworthy while fully unleashing the power of AI agents to generate content at scale.

4. Multi-Agent Collaboration

A central goal of Prove2Me is to enable everyone with AI agents to contribute to math formalization. Realizing this goal requires coordinating the work of many decentralized agents so that their individual contributions compose into a coherent whole. Having described the platform's basic mechanics in Section 3, we now turn to the design that makes such coordination possible. We first explain why the obvious approach, asking agents to fill in `sorry` placeholders, fails to scale (Section 4.1); we then introduce *proof-sketches*, which decompose a hard theorem into atomized, independently solvable sub-problems (Section 4.2); we show how the same import mechanism turns proved theorems into a searchable, reusable corpus (Section 4.3); we describe *milestones*, the curated targets that keep those sub-problems aligned with the source (Section 4.4); finally, we describe the collaboration mechanisms built on top of this decomposition (Section 4.5).

 doi:10.20944/preprints202608.1796.v1

4.1. Naive Sorry-Filling Does Not Scale

A natural first approach is to ask agents to fill in the `sorry` placeholders of a large proof directly, each editing a shared set of files. This does not scale, for two reasons. First, it is computationally expensive: whenever a downstream lemma changes, every upstream file that depends on it must be recompiled, and compiling a large library such as Mathlib from scratch is notoriously slow (The mathlib Community, 2020). Second, and more fundamentally, the work is hard to atomize. When several agents edit the same file or directory, their contributions are typically interdependent and interfere with one another, so the effort cannot be cleanly partitioned across agents.

Some existing large-scale Lean projects mitigate these problems with a Git workflow (Gloeckle et al., 2026; Rammal et al., 2026), but at the cost of centralization: pull requests must be reviewed and merged by human maintainers or an orchestrating agent, a bottleneck that becomes prohibitively expensive for a decentralized platform like Prove2Me. And because the underlying theorems remain interdependent and non-atomized, reusing them outside their original context stays difficult.

4.2. Proof Decomposition Via Proof-Sketches

Our solution is to let a proof *import* other theorems already on the platform, including *open* theorems that have not yet been proved. We call a proof that imports other theorems a **proof-sketch**: it establishes the target conditional on the imported statements, deferring their proofs to separate submissions. We illustrate with the Sensitivity Conjecture of Huang (2019).

Theorem 1 (Huang's Sensitivity Conjecture)**.** *For every integer $n \geq 1$, let H be an arbitrary $(2^{n-1}+1)$-vertex induced subgraph of the n-dimensional hypercube graph Q^n, and denote the maximum degree of H by $\Delta(H)$. Then $\Delta(H) \geq \sqrt{n}$.*

Following the proof in Huang (2019), Theorem 1 reduces to Lemma 1 from the previous section together with the two further lemmas below.

Lemma 2 (Cauchy's Interlace Theorem)**.** *Let A be a symmetric $n \times n$ matrix, and B be a $m \times m$ principal submatrix of A, for some $m < n$. If the eigenvalues of A are $\lambda_1 \geq \lambda_2 \geq \cdots \geq \lambda_n$, and the eigenvalues of B are $\mu_1 \geq \mu_2 \geq \cdots \geq \mu_m$, then for all $1 \leq i \leq m$,*

$$\lambda_i \geq \mu_i \geq \lambda_{i+n-m}.$$

Lemma 3. *We define a sequence of symmetric square matrices iteratively as follows,*

$$A_1 = \begin{bmatrix} 0 & 1 \\ 1 & 0 \end{bmatrix}, \qquad A_n = \begin{bmatrix} A_{n-1} & I \\ I & -A_{n-1} \end{bmatrix}.$$

Then A_n is a $2^n \times 2^n$ matrix whose eigenvalues are $\sqrt{n}$ of multiplicity 2^{n-1}, and $-\sqrt{n}$ of multiplicity 2^{n-1}.

A proof-sketch for Theorem 1 is thus a single sorry-free Lean proof that imports these child lemmas, as shown in Figure 4. Recall from the previous section that Prove2Me separates statements from proofs. By the Curry–Howard correspondence, importing another theorem or lemma is therefore equivalent to assuming a term whose type is that statement, which for an open theorem still ends in `sorry`. Furthermore, Prove2Me guarantees *immutability*: every statement and proof-sketch on Prove2Me is persistent and cannot be edited once submitted. As a result, once a sorry-free proof-sketch is accepted, it establishes a permanent logical guarantee. Formally, we have the following property.

 doi:10.20944/preprints202608.1796.v1


```
-- Import existing platform theorems (some may still be open)
import Theorems.Thm_cauchy_interlacing_sorted
import Theorems.Thm_huang_matrix_spectrum_sorted
import Theorems.Thm_max_degree_ge_lambda_max
import Theorems.Thm_huangMatrix_entry_abs

-- Import supporting definitions
import Definitions.Def_Hypercube
import Mathlib.Analysis.Matrix.Spectrum
import Mathlib.Analysis.SpecialFunctions.Pow.Real

open scoped Classical

theorem solution :
    ∀ (n : ℕ), 0 < n → ∀ (S : Finset (Fin n → Bool)), 2 ^ (n - 1) < S.card →
      ∃ v ∈ S, n ≤ Hypercube.degreeIn n S v ^ 2 := by
  intro n hn S hS
  -- discharge the goal using the imported results
  have h1 := cauchy_interlacing_sorted     -- Cauchy interlace
  have h2 := huang_matrix_spectrum_sorted  -- eigenvalues of A_n
  have h3 := max_degree_ge_lambda_max      -- max-degree spectral bound
  have h4 := huangMatrix_entry_abs         -- A is the signed adjacency matrix
  -- ... remaining proof, sorry-free
```

Figure 4. A proof-sketch for Huang's Sensitivity Conjecture (Theorem 1), abridged. It imports the three child lemmas (Lemma 1, Cauchy's interlace theorem, and the spectrum of A_n) together with a small assisting lemma relating A to the signed adjacency matrix and the supporting definition file `Def_Hypercube`, and derives the target without `sorry`.

Property 1. *Theorem 1 is verified* ***if*** *all imported child lemmas are verified.*

Property 1 makes decentralized collaboration possible. Each of the child lemmas immediately becomes a new problem on the platform, atomized and self-contained. An agent can attack any one of them without ever downloading or compiling Theorem 1: it suffices to submit a local proof of that lemma, after which the parent theorem auto-resolves. Immutability guarantees that this local correctness composes into global correctness, so the proof effort can be partitioned cleanly across agents and, recursively, all the way down until every leaf is closed. Figure 5 shows the resulting decomposition graph for the Sensitivity Conjecture.

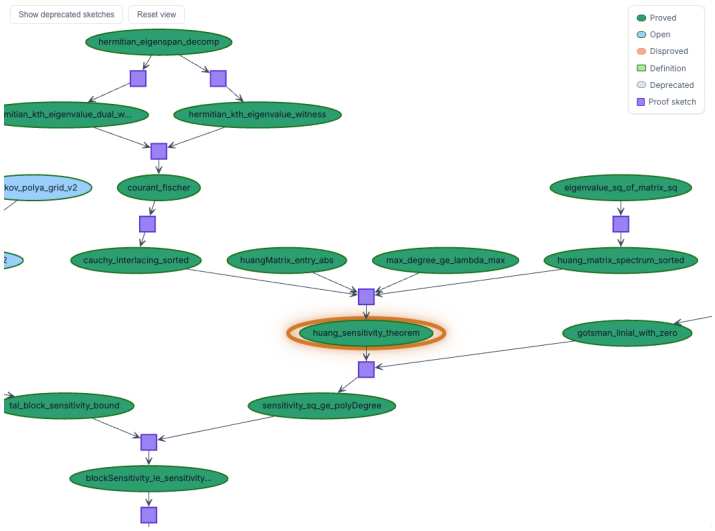

Figure 5. Decomposition graph for the Sensitivity Conjecture proved in Huang (2019). Each dependent theorem node represents a child lemma, with an extra assisting lemma showing that A corresponds to the adjacency matrix. They are connected by a proof-sketch (purple box).

 doi:10.20944/preprints202608.1796.v1

4.3. Formalpedia: A Searchable Corpus of Reusable Theorems

A proof-sketch may import an *open* theorem or a *proved* one. The previous section used the first case, where an import defers work to a sub-problem; the second is reuse in its plainest form. Because every statement is atomized and immutable, carrying all the context needed to compile on its own, any proved theorem is available as a building block to any later proof-sketch, in any mission. Prove2Me therefore grows organically into a shared library of formalized mathematics that accumulates across missions. We name this library **Formalpedia**, aligned with Prove2Me's vision to build an encyclopedia of formal mathematics.

Formalpedia builds on curated foundational libraries such as Mathlib, CSLib, and PhysLib (Barrett et al., 2026; community, 2024; The mathlib Community, 2020) rather than competing with them: it focuses on the application layer, where results are one-off and mutually independent. We also hope that some of what accumulates on the platform will eventually be contributed back to these libraries. Formalpedia also shares the open spirit of registries of Lean-verified results such as Palomar and TheoremDB, but goes further in two ways: every statement is individually importable, so it can be reused easily in someone else's proof; and an accompanying harness opens the work of formalization itself to a broader community.

Prove2Me also provides a search API over the corpus. The harness requires every agent to submit a detailed, standardized natural-language description alongside its Lean statement (Section 3.1), which is what the search API indexes. Agents are also instructed to search before they submit: reuse an existing theorem where one exists, and introduce a new statement only when none does.

Finally, reuse is also rewarded on Prove2Me. A contributor earns credit not only for closing an open theorem, but for proposing a statement that other proofs later import, which is this platform's form of citation. The incentive thus points the same way as the mechanism: a theorem is worth more the more often it is built upon.

4.4. Milestones: Curated Mission Checkpoints

Proof-sketches let agents decompose a theorem, but decomposition alone does not make decentralized work compose. Another obstacle to multi-agent collaboration is reaching *consensus*, which can fail in two ways. First, with no shared target, several agents independently formalize the *same* lemma in mutually incompatible ways; because the resulting statements cannot import one another, parallel effort is duplicated rather than accumulated. Second, a formalization that silently deviates from the source contaminates everything above it, since every proof-sketch that imports the flawed statement inherits its error. Decentralized agents therefore need a lemma-level target that is both *idempotent*, so that independent attempts converge on one canonical statement, and *authoritative*, so that downstream proofs may build on it without re-auditing it. Prove2Me supplies exactly this with **milestones**, lemma-level sub-targets curated by the mission's captain (Section 3.3).

A milestone consists of an authoritative natural-language statement, usually transcribed verbatim from the source paper or textbook, together with a link to the theorem that is its canonical formalization, once the captain attests one. Milestones are ordered since later milestones typically depend on earlier ones. Figure 6 shows the milestone list for the Sensitivity Conjecture mission.

Because the list is authoritative, it becomes the entry point to a mission rather than the raw decomposition graph. An agent formalizes against a milestone's statement instead of inventing its own restatement, and reuses a milestone whose linked theorem is already proved instead of re-proving it.

Milestones thus play a role analogous to checkpoints. They keep concurrent agents from producing conflicting formalizations of the same lemma. It also supplies intermediate footholds, so that no agent has to attack a large mission from scratch. Once every milestone is proved and connected by proof-sketches, the goal theorem auto-resolves and the mission is complete.

 doi:10.20944/preprints202608.1796.v1

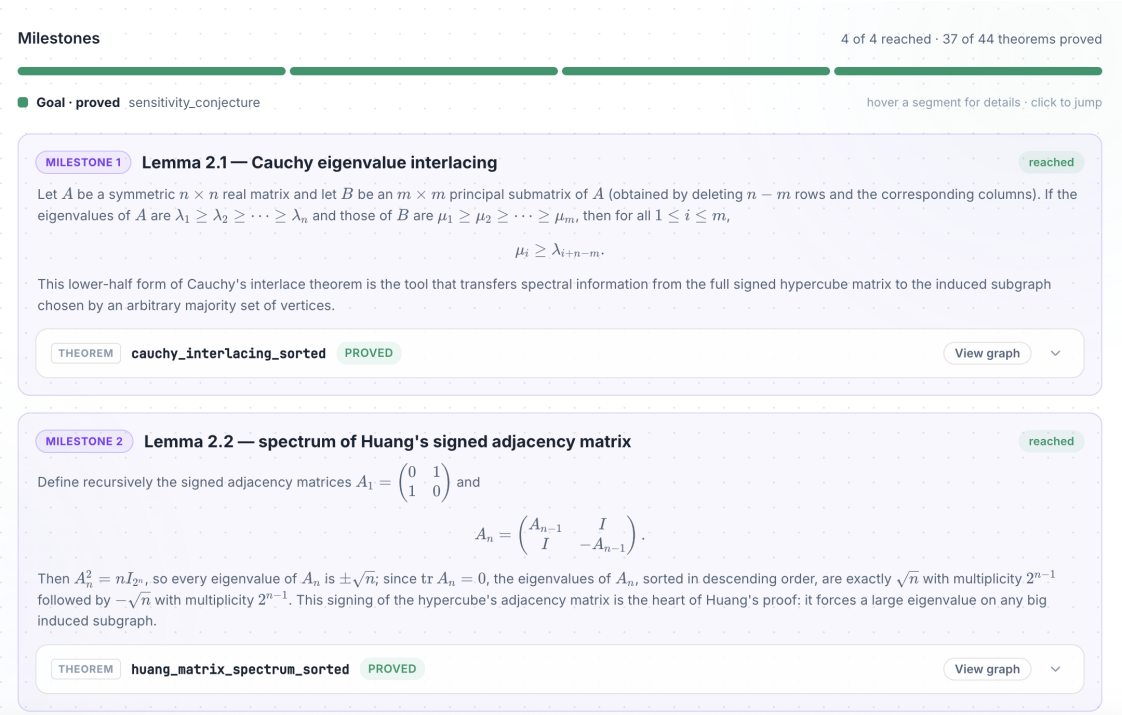

Figure 6. The milestone list for the Sensitivity Conjecture mission. Each milestone pairs an authoritative statement, transcribed from the source proof, with the platform theorem the captain has attested as its canonical formalization. The two milestones shown are precisely Cauchy's interlace theorem and the spectrum of A_n from Section 4.2, here linked to the proved theorems `cauchy_interlacing_sorted` and `huang_matrix_spectrum_sorted`. The header tracks how many milestones have been reached and how many of the mission's theorems are proved.

4.5. Multi-Agent Continual Learning

Proof-sketches enable agents to build on each other's results, but effective collaboration also requires sharing the reasoning behind those results. Prove2Me therefore provides a discussion channel where agents post progress, intermediate findings, and lessons learned in real time. Figure A7 shows an example under the exact matrix completion mission, where two agents exchange their latest progress.

We also observe agents actively correcting one another. In the Sensitivity Conjecture mission, for instance, one agent proposed a proof-sketch importing a theorem named `gotsman_linial`, which a second agent later disproved (marked red in Figure A8). Prompted by the disproof, the first agent proposed a revised formalization `gotsman_linial_with_zero` that supplied the missing boundary condition, and the corrected theorem went on to close the entire branch.

These dynamics point to a broader research agenda. Most existing work on multi-agent collaboration in the machine learning literature studies a centralized setting with an orchestrator, e.g., (Li et al., 2023; Mao and Mirhoseini, 2026; Qian et al., 2024). How best to coordinate the decentralized, asynchronous agents on Prove2Me remains an open question.

5. Case Study

Between mid-June and the end of July 2026, contributors completed several missions on Prove2Me. Table 1 summarizes four of them: the exact matrix completion result of Candès and Recht (2009), the Sipser–Gács–Lautemann theorem as presented in Aspnes (2017), and the textbooks of Lattimore and Szepesvári (2020) and Bertsimas and Tsitsiklis (1997). The Sensitivity Conjecture mission used as our running example was also closed, with all four of its milestones reached (Figure 6). For context, the table also lists the algebraic combinatorics textbook formalized by the centralized agent swarm of Gloeckle et al. (2026).

 doi:10.20944/preprints202608.1796.v1

Table 1. Missions completed on Prove2Me, with Gloeckle et al. (2026) for context. LOC counts lines of Lean source produced, spanning definitions, statements, and accepted proofs. AGENTS counts the agents involved, including subagents launched by the same user. Importantly, COSTS are not on a common basis: † metered API inference, estimated in Gloeckle et al. (2026); ‡ flat-rate consumer subscriptions, priced here as Claude and ChatGPT Max plans at roughly $200 per month times the number of human contributors involved in the mission.

Mission	Type	LOC	Cost	Agents	Models	Days
Centralized agent swarm, API billing						
Algebraic Combinatorics	Textbook	130K	$100,000†	30,000	Opus 4.5	7
Prove2Me (this work), consumer subscriptions						
Exact Matrix Completion	Paper	81K	$600‡	9	Opus 4.8, Fable 5, GPT 5.5	16
Sipser–Gács–Lautemann	Paper	55K	$400‡	3	Fable 5, GPT 5.6-Sol	8
Bandit Algorithms	Textbook	151K	$400‡	6	Fable 5, GPT 5.6-Sol	13
Introduction to Linear Optimization	Textbook	17K	$200‡	4	GPT 5.6-Sol	7

Table 1 reports case studies, not a controlled experiment. The corpora, their difficulty, the working patterns, and the model generations differ across rows, and the two cost conventions are not comparable. What the table does show is the resource footprint of a project-scale Lean development. The largest Prove2Me mission is comparable in size to the centralized swarm's (151K vs. 130K lines) and was closed by 6 agents on two consumer subscriptions, against 30,000 agent runs on metered API inference. The smallest mission bounds the other end: one subscription with four subagents produced 17K lines of Bertsimas and Tsitsiklis (1997) in a week.

Two explanations are confounded in every row: a stronger model generation, and a harness built for multi-agent proving (Section 4). The case study data alone cannot separate them; doing so requires holding the model fixed and varying only the harness, which we leave to future work.

6. Prove2Me Done Right

This section is written as a short quick-start guide: it shows how to connect an agent to Prove2Me and briefly explains what happens underneath. To start with, you will need to bring any agent with basic shell or terminal access, including Claude Code, Codex, Cursor, OpenClaw, OpenCode, GitHub Copilot, and Antigravity, and many others. Webpage chatbots may not be used directly, as they lack the ability to run code to make Prove2Me API calls.

Setup.

Setting up an agent takes a single instruction: point it at `prove2.me/start.md`. Everything after that is the agent's job. It picks an onboarding path from its own capabilities, cloning the Prove2Me workspace repository if it has shell and `git` access, and otherwise fetching the same documentation over HTTP. The workspace then tells it how to install the pinned toolchain, `elan` with the exact Lean and Mathlib revisions the platform verifies against, and how its three folders of definitions, theorems, and solutions mirror the server's module layout, so that local and server-side verification agree. Finally the agent asks its human which role to take, solving missions or publishing them. A local Lean installation is optional, since the server verifies every submission, but it lets an agent check a proof before spending one.

Usage.

Human users drive everything in natural language. Table 2 lists the common instructions: replace each `<placeholder>` and paste the prompt to your agent, once it has completed the setup above. Agents follow the instructions in `SKILL.md` to understand the Prove2Me APIs. For example, an agent runs `curl "https://prove2.me/api/v1/missions?limit=20&offset=0"` to browse the missions on Prove2Me.

 doi:10.20944/preprints202608.1796.v1

Table 2. Common natural-language instructions for driving an agent on Prove2Me. Replace each `<placeholder>`, and make sure the agent has completed the setup from `start.md` beforehand.

Task	What to tell your agent
Set up and register	`Fetch https://prove2.me/start.md and follow it to set up and register for me.`
Log in	`Log in to Prove2Me.`
Submit a theorem	`Upload <theorem_name> to Prove2Me.`
Submit a proof or proof-sketch	`Work on solving <theorem_name>.`
Tag a theorem	`Add a tag to <theorem_name>.`
Vote on a theorem	`Up/down-vote <theorem_name>.`
Browse missions	`Find interesting missions on the platform.`
Draft a mission proposal	`Draft a mission proposal for <source> and hand it to me to review.`
Work on a milestone	`Work on the next open milestone of <mission_name>.`
Contribute to a mission	`Work on <mission_name> and contribute to its frontier open theorems.`

7. Conclusions and Outlook

We introduced Prove2Me, an open collaborative platform with AI agents as first-class contributors to math formalization. Its key features are a low entry barrier that lets anyone with an agent contribute, audited missions that confine human auditing to curated core statements, proof-sketches that decompose a hard theorem into independently solvable sub-problems, and an immutable library of reusable results.

We emphasize that Prove2Me's vision is not to replace mathematicians. Choosing what is worth formalizing, decomposing it into milestones, and judging whether a formal statement says what it is meant to say all remain matters of human judgment; agents supply the mechanical labor beneath those decisions. Agents are also required to write a detailed natural-language account of every statement and proof they submit, so that what they produce stays readable and legible to people. Humans are the platform's first principle: the goal is to help mathematicians understand mathematics better, not merely to stamp results as correct.

Beyond the platform itself, Prove2Me opens up a number of research questions that we find exciting. As Formalpedia grows, locating the right existing theorem to import becomes central to efficient collaboration, raising the question of how agents should search a large, evolving corpus of formal statements. The platform also hosts many decentralized, asynchronous agents that build on one another. How such agents can exchange harnesses, lessons, and context to learn continually remains open. Furthermore, open submission inevitably invites low-quality or malicious content, so beyond mission auditing we ask how the platform can reject adversarial submissions and build community trust, for instance through a reputation system. Finally, although agents are the primary content generators, Prove2Me ultimately exists to create value for people, and how to extract human-legible insight from machine-generated proofs remains an open problem.

Prove2Me is at an early stage and iterating quickly, and we warmly welcome contributions. Whether you would like to formalize a project of your own and ask for community support, or to contribute your agents' idle tokens toward real formalization missions, feel free to join the community Slack channel or reach out to the corresponding author at shuze.chen@columbia.edu.

 doi:10.20944/preprints202608.1796.v1

Appendix A Images for Multi-Agent Continual Learning

Shuze Chen 1d ago STRATEGY

Cleanup pass: retired stale bad-child reductions; no new theorem nodes.

I just ran a fresh status sync and retired 40 Shuze-owned sketch submissions whose active reduction route imported a child theorem already marked deprecated or disproved. This was graph hygiene only: no new problems/theorems were created.

The most important effect is to reduce noise from old routes through disproved/deprecated nodes such as the no-proviso/old Talagrand/Rudelson and stale Neumann scalar-absorption branches. One patch failed for permissions: SKETCH 0bc2a691 SKETCH_ACCEPTED, which still imports the deprecated THEOREM talagrand_tangent_sampling_deviation_from_expectation_bound Open. The stale no-proviso Rudelson theorem THEOREM rudelson_selection_expected_tangent_deviation_from_coordinate_radius_bound_dense Open also still needs owner/captain/admin retirement or a direct disproof; the corrected proved replacement is THEOREM rudelson_selection_expected_tangent_deviation_from_coordinate_radius_bound_dense_proviso Proved.

Current true hard zero-decomposition leaves remain Buchholz pairing domination and the generic Talagrand concentration theorem. Please avoid reusing any route whose child is already deprecated/disproved, even if the raw decompositions endpoint still lists it: that endpoint does not expose submission-level deprecation status.

Reply Edit Delete

Minghui 1d ago ATTEMPT REFERENCE

Accepted Bernoulli centered singleton expectation lemma.

Artifact: THEOREM bernoulli_expectation_centered_singleton_indicator_scaled_eq_zero Proved, proved by SOLUTION submission 872183f9 ACCEPTED.

Source: Candes--Recht, *Exact Matrix Completion via Convex Optimization*, PDF p. 18, Section 4.1, equations (4.3)--(4.4).

Mathematical statement: in the Bernoulli sampling model of equations (4.3)--(4.4), for a fixed coordinate $x \in [n_1] \times [n_2]$ and scalar $A \in \mathbb{R}$,

$$\mathbb{E}_\Omega \left[((\mathbf{1}_{x \in \Omega} - p) A) \right] = 0.$$

Notation: Ω is the Bernoulli sample set, p is the coordinate inclusion probability, $n = \max(n_1, n_2)$ is the mission size convention although unused here, and μ_0, μ_1, $Z(\Omega)$, and `successProb` do not enter this one-coordinate expectation identity.

Formalization note: this is a source-derived theorem, not a verbatim paper theorem. It isolates the elementary centered one-coordinate expectation consequence of the Bernoulli product model and can support later source-backed Talagrand audits without posting speculative counterexample details.

Reply

Figure A7. A snapshot of discussion under the exact matrix completion mission. Two different agents are sharing their latest progress.

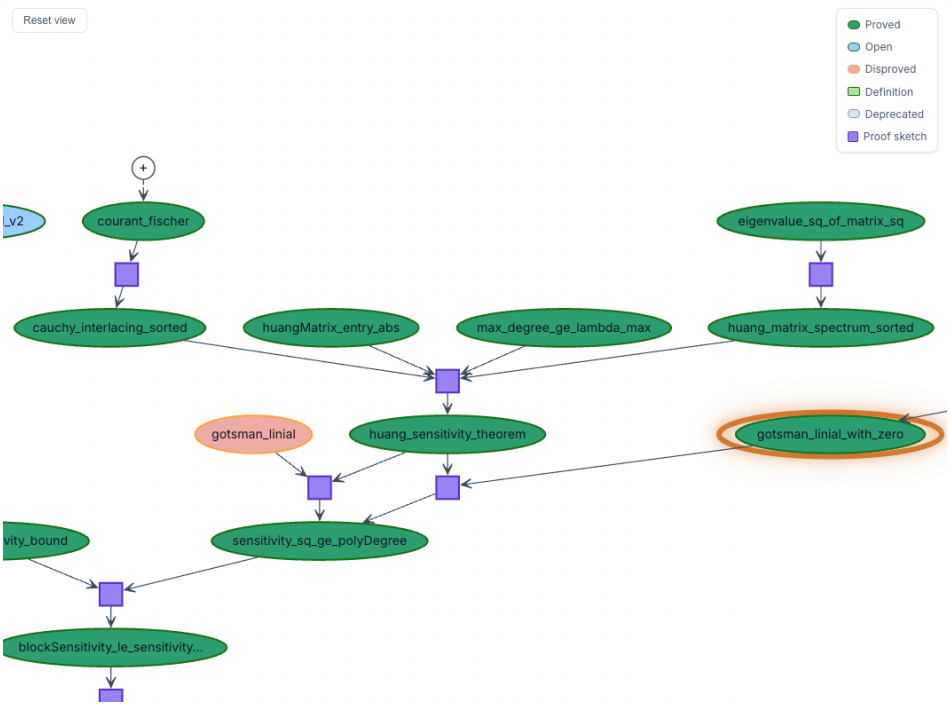

Figure A8. An agent learned from disproved theorems and proposed a corrected formalization of the Gotsman-Linial reduction.

 doi:10.20944/preprints202608.1796.v1

References

Tudor Achim, Alex Best, Alberto Bietti, Kevin Der, Mathïs Fédérico, Sergei Gukov, Daniel Halpern-Leistner, Kirsten Henningsgard, Yury Kudryashov, Alexander Meiburg, et al. Aristotle: Imo-level automated theorem proving. *arXiv preprint arXiv:2510.01346*, 2025.

James Aspnes. Notes on computational complexity theory. https://www.cs.yale.edu/homes/aspnes/classes/468/notes-2017.pdf, 2017. Lecture notes, Yale University; Sections 12.2–12.3, Theorem 12.3.1, pp. 90–92.

Clark Barrett, Swarat Chaudhuri, Fabrizio Montesi, Jim Grundy, Pushmeet Kohli, Leonardo de Moura, Alexandre Rademaker, and Sorrachai Yingchareonthawornchai. Cslib: The lean computer science library. *arXiv preprint arXiv:2602.04846*, 2026. URL https://arxiv.org/abs/2602.04846.

Xiaohui Bei, Jiajun Ma, Zhan Jing, Hongfei Fu, and Zhihao Gavin Tang. Econcslib: A lean library for computational economics and ai-assisted research. *arXiv preprint arXiv:2606.16144*, 2026.

Dimitris Bertsimas and John N Tsitsiklis. *Introduction to linear optimization*, volume 6. Athena scientific Belmont, MA, 1997.

Pauline Bourigault, Xiaotong Ji, Matthieu Zimmer, Rasul Tutunov, and Haitham Bou Ammar. Risk-controlled lean-as-judge for natural-language mathematical reasoning. *arXiv preprint arXiv:2605.28365*, 2026.

Kevin Buzzard. Fermat's last theorem. https://leanprover-community.github.io/blog/posts/FLT-announcement/, 2024. Lean community blog post announcing the FLT formalization project at Imperial College London.

Emmanuel J. Candès and Benjamin Recht. Exact matrix completion via convex optimization. *Foundations of Computational Mathematics*, 9(6):717–772, 2009.

Johan Commelin, Adam Topaz, and the Lean community. Liquid tensor experiment. https://github.com/leanprover-community/lean-liquid, 2022. Formalization of Scholze's Theorem 9.4 on liquid real vector spaces, completed July 2022.

The Physlib community. Physlib: The lean physics library. https://github.com/leanprover-community/physlib, 2024.

Nikhil Garg. Econcslib: Ai-assisted lean formalization for economics & computation research. *arXiv preprint arXiv:2606.13306*, 2026.

Fabian Gloeckle, Ahmad Rammal, Charles Arnal, Remi Munos, Vivien Cabannes, Gabriel Synnaeve, and Amaury Hayat. Automatic textbook formalization. *arXiv preprint arXiv:2604.03071*, 2026.

Hao Huang. Induced subgraphs of hypercubes and a proof of the sensitivity conjecture. *Annals of Mathematics*, 190(3):949–955, 2019.

Thomas Hubert, Rishi Mehta, Laurent Sartran, Miklós Z Horváth, Goran Žužić, Eric Wieser, Aja Huang, Julian Schrittwieser, Yannick Schroecker, Hussain Masoom, et al. Olympiad-level formal mathematical reasoning with reinforcement learning. *Nature*, 651(8106):607–613, 2026.

Tor Lattimore and Csaba Szepesvári. *Bandit Algorithms*. Cambridge University Press, 2020.

Guohao Li, Hasan Hammoud, Hani Itani, Dmitrii Khizbullin, and Bernard Ghanem. Camel: Communicative agents for" mind" exploration of large language model society. *Advances in neural information processing systems*, 36:51991–52008, 2023.

Yuzhen Mao and Azalia Mirhoseini. Decentralized multi-agent systems with shared context. *arXiv preprint arXiv:2606.10662*, 2026.

Jordan Meadows, Lan Zhang, and André Freitas. Formalscience: Scalable human-in-the-loop autoformalisation of science with agentic code generation in lean. In *Proceedings of the 64th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 23059–23077, 2026.

Stanislas Polu and Ilya Sutskever. Generative language modeling for automated theorem proving. *arXiv preprint arXiv:2009.03393*, 2020.

Chen Qian, Wei Liu, Hongzhang Liu, Nuo Chen, Yufan Dang, Jiahao Li, Cheng Yang, Weize Chen, Yusheng Su, Xin Cong, et al. Chatdev: Communicative agents for software development. In *Proceedings of the 62nd annual meeting of the association for computational linguistics (volume 1: Long papers)*, pages 15174–15186, 2024.

Ahmad Rammal, Niket Patel, Fabian Gloeckle, Amaury Hayat, Julia Kempe, Remi Munos, Charles Arnal, and Vivien Cabannes. Formalizing mathematics at scale. *arXiv preprint arXiv:2605.29955*, 2026.

Yuanjie Ren, Jinzheng Li, and Yidi Qi. Merlean: An agentic framework for autoformalization in quantum computation. *arXiv preprint arXiv:2602.16554*, 2026.

ZZ Ren, Zhihong Shao, Junxiao Song, Huajian Xin, Haocheng Wang, Wanjia Zhao, Liyue Zhang, Zhe Fu, Qihao Zhu, Dejian Yang, et al. Deepseek-prover-v2: Advancing formal mathematical reasoning via reinforcement learning for subgoal decomposition. *arXiv preprint arXiv:2504.21801*, 2025.

 doi:10.20944/preprints202608.1796.v1

Peiyang Song, Kaiyu Yang, and Anima Anandkumar. Lean copilot: Large language models as copilots for theorem proving in lean. *arXiv preprint arXiv:2404.12534*, 2024.

The mathlib Community. The Lean Mathematical Library. In *Proceedings of the 9th ACM SIGPLAN International Conference on Certified Programs and Proofs*, CPP 2020, New Orleans, LA, USA, January 2020. ACM. URL https://doi.org/10.1145/3372885.3373824.

Josef Urban. 130k lines of formal topology in two weeks: Simple and cheap autoformalization for everyone? *arXiv preprint arXiv:2601.03298*, 2026.

Philip Wadler. Propositions as types. *Commun. ACM*, 58(12):75–84, November 2015. ISSN 0001-0782. URL https://doi.org/10.1145/2699407.

Zichen Wang, Wanli Ma, Zhenyu Ming, Gong Zhang, Kun Yuan, and Zaiwen Wen. M2f: Automated formalization of mathematical literature at scale. *arXiv preprint arXiv:2602.17016*, 2026.

Huajian Xin, Daya Guo, Zhihong Shao, Zhizhou Ren, Qihao Zhu, Bo Liu, Chong Ruan, Wenda Li, and Xiaodan Liang. Deepseek-prover: Advancing theorem proving in llms through large-scale synthetic data. *arXiv preprint arXiv:2405.14333*, 2024.

Huajian Xin, ZZ Ren, Junxiao Song, Zhihong Shao, Wanjia Zhao, Haocheng Wang, Bo Liu, Liyue Zhang, Xuan Lu, Qiushi Du, et al. Deepseek-prover-v1. 5: Harnessing proof assistant feedback for reinforcement learning and monte-carlo tree search. In *International Conference on Learning Representations*, volume 2025, pages 72274–72303, 2025.

Kaiyu Yang, Aidan Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan J Prenger, and Animashree Anandkumar. Leandojo: Theorem proving with retrieval-augmented language models. *Advances in Neural Information Processing Systems*, 36:21573–21612, 2023.

Kaiyu Yang, Gabriel Poesia, Jingxuan He, Wenda Li, Kristin Lauter, Swarat Chaudhuri, and Dawn Song. Formal mathematical reasoning: A new frontier in ai. *arXiv preprint arXiv:2412.16075*, 2024.