

The Shape of Math To Come*

Alex Kontorovich[†]

Abstract. We present an overview of how certain computational tools currently interact with mathematical practice, and reflect on the implications for research mathematics in the short to medium term, as the field navigates the emerging age of AI and formal verification systems.

1 Introduction.

This paper will discuss a vision for what research mathematics may look like in the age we now seem to be entering, of AI and formalization. Given the pace of progress, my words will likely be obsolete almost the instant they appear, but I will nevertheless take the risk of setting down how I see the landscape, in September 2025.

My own involvement in formalization began about five years ago (see [9]), driven by the particular challenges of my research area. Analytic number theory is characterized by particularly long, technical arguments spanning scores of pages. These proofs typically synthesize numerous lemmas, each valid only within constrained ranges of their parameters, which must align perfectly at the end for the main theorem to hold. This intricate dance of parameters and constraints is both what I love about the subject, and what makes it treacherous: a single minus sign error buried somewhere deep in the middle of one of these technical arguments can invalidate the entire main result.

When an argument involves something like a complicated algebraic manipulation, I can use a computer algebra system such as Sage or Mathematica to verify it once and for all; then I no longer waste time going over that part of the argument again and again to ensure no errors were made. Shouldn’t there be similar software, but to track the logical dependencies and parameter constraints across my entire paper?

Such software does exist, and is called an Interactive Theorem Prover; see §7. It, in principle, allows users to “formalize” their papers, entering the proof line by line until the computer certifies that the argument is complete. The reality, however, is sobering. As things currently stand, I cannot even *state* formally most theorems of interest to me, much less prove them. The existing formal libraries (see §7.2) would need to grow by orders of magnitude before doing research on top of them becomes a reality.

This leads to a fundamental conundrum: new mathematics seems to grow at an exponential rate, say, δ , and formal libraries also grow exponentially, at a rate, say, ε ; but as far as I can tell, we currently have:

$$\delta > \varepsilon.$$

So if this inequality persists, formalization will *never* catch up to natural language mathematics research, and the dream of software as useful to *reasoning* as computer algebra systems are to symbolic manipulation will forever remain a distant mirage.

Unless, that is, formalization can be “supercharged” by automation. And so I’ve gone from wanting to do research math, to wanting sufficient formalization to do research math, to wanting sufficient AI to do sufficient formalization to do research math! What follows is my vision for how we might get there, and how things might look if/when we do. The old adage: “It’s hard to make predictions, especially about the future,” variously attributed to Yogi Berra, Niels Bohr, and others, is very apt here. So before we begin, let’s take a step back.

2 In the Year 2000...

At the turn of the millennium, some of the world’s most prominent mathematicians gathered in Tel Aviv for a conference to discuss what mathematics might look like in the 21st century. The proceedings were published as

*The title is an homage to Ornette Coleman’s 1959 album, “The Shape of Jazz to Come”.

[†]Department of Mathematics, Rutgers University (alex.kontorovich@rutgers.edu, <http://math.rutgers.edu/~alexk>). Partially supported by NSF grant DMS-2302641, US-Israel BSF grant 2020119, and a Simons Foundation Fellowship.

a special issue in GAFA called “Visions of Mathematics”, and I strongly recommend reading the whole issue [1]. A particularly prescient contribution was given by Tim Gowers, titled “Rough Structure and Classification” [11].

Section 2 of Gowers’s paper is provocatively called: “Will Mathematics Exist in 2099?” In it, he imagines a “dialogue between a mathematician and a computer in two to three decades’ time” – and here we are! His imagined dialogue is eerily similar to the experiences so many of us now have on a regular basis when interacting with “chatbots” such as OpenAI’s GPT, Google’s Gemini, Anthropic’s Claude, xAI’s Grok, and others:

Mathematician. Is the following true? Let $\delta > 0$. Then for N sufficiently large, every set $A \subset \{1, 2, \dots, N\}$ of size at least δN contains a subset of the form $\{a, a + d, a + 2d\}$?

Computer. Yes. If A is non-empty, choose $a \in A$ and set $d = 0$.

M. All right all right, but what if d is not allowed to be zero?

C. Have you tried induction on N , with some $\delta = \delta(N)$ tending to zero?

M. That idea is no help at all. Give me some examples please.

C. The obvious greedy algorithm gives the set

$$\{1, 2, 4, 5, 10, 11, 13, 14, 28, 29, 31, 32, 37, 38, 40, 41, \dots\}.$$

I notice that large parts of the set are translations of other parts. In fact, this set is very like the Cantor set, so this gives a bound of $\delta \geq N^{(\log 2 / \log 3) - 1}$.

Figure 2.1: A snippet from [11]

Gowers notes: “The idea of the dialogue is that the computer is very helpful to the mathematician, while not doing anything *particularly clever*” (emphasis added). So, in that vein: let’s establish some common ground and be precise about what I shall mean by a “Large Language Model” (LLM).

3 What is an LLM?

We’ve all had the following experience: when reading a book, you reach the bottom of a page mid-sentence, and as your fingers fumble, struggling to turn to the next page, your mind can’t help but race ahead, filling in how the sentence might continue. For example:

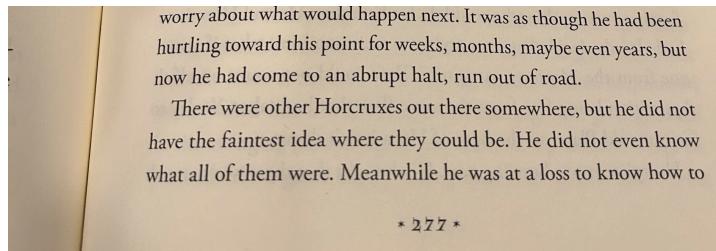


Figure 3.1: The bottom of a page from “Harry Potter and the Deathly Hallows” by J. K. Rowling

We read that

“[Harry] did not even know what all [the Horcruxes] were. Meanwhile he was at a loss to know how to...”

How to *what*? Your mind instinctively begins to generate possibilities. How to “destroy” the Horcruxes? Or perhaps how to “find” them? Or, far less plausibly, how to “sandwich”?

I imagine standing at a game of Family Feud, hearing Steve Harvey announce: “Survey says!...” But here the “survey” doesn’t poll 100 random people – it samples all available human text! The results might look something like this: given the context, roughly 20% of people, say, would continue the sentence with “destroy,” another 7% would use, “find,” and perhaps a tiny 0.003% would suggest “sandwich.”

This, at its core, is a Large Language Model: it is a function whose input is “context” – a possibly very long sequence of “tokens” (words, punctuation marks, etc.) – and whose output is *not* the next word (as is commonly misconstrued), but rather a *probability distribution* over all possible next tokens.



Figure 3.2: An LLM is a function from context (a sequence of tokens) to a likelihood distribution of potential next tokens

For our purposes, we can set aside the unfathomable engineering that makes such a function possible: the architecture (transformers), the representation of data (embeddings in high-dimensional vector spaces), the training process (gradient descent, pre-training, fine-tuning), and so on. We simply take as given that such a function exists.

It is on top of such a model that one builds the aforementioned chatbot; the bot must decide how to *choose* the actual next token from the probability distribution the model outputs. This choice depends on parameters like the “temperature” – perhaps always selecting the most likely token (which tends to produce somewhat stilted prose), or sampling randomly from the distribution, or various options in between. Once a token is chosen, the bot appends it to the context, and feeds the entire sequence back through the model; it then chooses the next token, and the next and the next, until it selects a special token that signals “stop talking.” (We will return later to more complex agentic behavior, like use of tools and self-directed workflows; for now, I’m focusing on the mechanism of a “pure” chatbot.) It is perhaps this pure chatbot behavior that Gowers was envisioning as a computer process “not doing anything particularly clever” in its discussion with you.

A crucial observation here is that this process is profoundly *stochastic*. At each step, the bot makes a random choice, and each choice affects the probability distribution for all subsequent choices. The randomness compounds with every token – random upon random upon random. This stochasticity is simultaneously the system’s great strength – enabling it to tackle a vast range of problems with remarkable versatility – but as we shall see, it creates fundamental challenges when applied to deterministic problems like mathematical proof.

4 Stochastic vs. Deterministic Problems.

Consider the difference between approximate and exact correctness. A long poem with one poorly-chosen word in the middle can still be beautifully moving; the same for an improvised jazz solo with an accidentally dissonant note. These are domains where 99% success represents genuine achievement. But a 99% correctly proved theorem? A single error can render everything that follows meaningless! Mathematics requires deterministic correctness, not probabilistic approximation.

To understand why this matters quantitatively, suppose an LLM achieves 99% accuracy at each individual step – a remarkably high success rate. If a mathematical argument requires chaining 1000 such steps in sequence, with the success of each step independent of the others, what is the probability of overall success? The answer: $0.99^{1000} \approx 0.004\%$. A process that seems highly reliable at each step becomes almost certainly wrong when many steps compound.

The human brain apparently also operates in two rather distinct modes, popularized in Daniel Kahneman’s *Thinking, Fast and Slow* as “System I” versus “System II”. The former is fast, automatic, and effortless; the latter is slow, deliberate, and effortful. When asked “What is $8 + 3$?” most adults respond instantly – this is System I processing. You’ve encountered this calculation so many times that your brain has essentially memorized the answer. (This is why children learning their times tables is so important: it converts what was once effortful System II calculation into automatic System I recall, freeing cognitive resources for more complex tasks.)

But when I calculate 437×82 on paper, I require System II processing. However, because I’ve learned the standard algorithm, I can decompose this problem into a long chain of simple System I calculations: first compute $2 \times 7 = 14$, write the 4, carry the 1, and so on. The key insight is that System II mathematical reasoning often consists of carefully chaining together many System I steps, where success requires *every link in the chain* to be correct.

This brings us back to our stochastic problem. If your “System I machine” is 99% accurate at each step, but you need to chain 1000 steps to solve a System II problem, you face the aforementioned 0.004% success rate. This is why using stochastic tools for deterministic mathematics is fundamentally challenging.

Yet this argument assumes that AI capabilities will remain static. Given the extraordinary pace of progress in recent years, perhaps these limitations are merely temporary?

5 AI Capabilities Are Advancing Rapidly...

The evidence for continued rapid progress is compelling. Consider performance on the International Mathematical Olympiad (IMO), widely regarded as one of the most challenging mathematical competitions for high school students:

- In 2023, the number of AI systems that could earn a single point on the IMO was... zero.
- In 2024, DeepMind’s AlphaProof + AlphaGeometry achieved a silver medal performance, one point shy of gold [10].
- In 2025, multiple AI systems claimed gold medal scores, with roughly half using formal verification methods and half working purely in natural language.¹

This trajectory is remarkable. Within two years, we moved from complete inability to gold medal performance. Moreover, the 2025 success of purely natural language approaches demonstrates that formal verification is not strictly necessary for solving difficult mathematical problems – at least at the IMO level.

By the next IMO in the summer of 2026, it may be reasonable to predict that the number of leading AI systems competing in the IMO will return to zero – not because they’ve regressed, but because the competition will have become trivially easy for them. (Open source models may continue to participate, but commercial systems will likely find it beneath their capabilities.)

This pattern of accelerating capability is not unique to mathematics. It reflects what Richard Sutton famously called the “Bitter Lesson” of seventy years of AI research: domain-specific solutions crafted by human experts are consistently overtaken by general-purpose methods that leverage ever-increasing computational resources [23]. Time and again, we’ve seen that scaling compute and data beats clever engineering tricks.

If Moore’s Law continues – or even if it merely slows rather than stops – we can expect LLMs to sustain progressively longer chains of reasoning. Systems that struggle with 100-step arguments today may handle 1000-step arguments tomorrow, and 10,000-step arguments the day after. The transition from high school mathematics to undergraduate to PhD-level to research-level mathematics may simply require scaling up the sustained reasoning capability by some large multiple.

Indeed, one can pose the following a loose analogy to chess ratings: a 2500-ELO rated chess grandmaster might correspond to a freshly minted mathematics PhD; 2600 to an established researcher; 2700 to a leading expert; and 2800+ to mathematical giants. If AI systems can reach the 2300-equivalent level for elite high school mathematics, why not eventually reach 2800 and beyond?

So perhaps the stochastic/deterministic tension I’ve emphasized will simply dissolve as systems become powerful enough to maintain reliability across arbitrarily long chains of reasoning. Perhaps natural language AI will, after all, be sufficient for all of mathematics? It seems to me that even this optimistic scenario fails to solve a fundamental problem.

6 ... But Even Perfect Scaling Is Insufficient.

Let us imagine, optimistically, that AI continues to improve at its current pace. Suppose that within a decade, we have systems capable of producing novel mathematical research – not just solving competition problems, but discovering new theorems, developing new theories, and writing papers that advance human knowledge.

Consider a future AI system that can generate 100 research papers per day. To make the scenario as favorable as possible, suppose that this system has achieved extraordinary reliability: 99 of those 100 papers are completely correct, with rigorous proofs and meaningful contributions to mathematics. Only 1 paper in 100 contains an error – perhaps a subtle mistake in a lemma, or a case that wasn’t properly handled, or a minus sign buried somewhere deep in a calculation.

¹While DeepMind used a general-purpose natural-language model, it still called an AlphaGeometry-type system for the geometry problems.

Would you rely on such a system? I would **not**.

To use this tool effectively, I would need to read through all 100 papers to determine which 99 are correct and which 1 is flawed. But verifying the correctness of a mathematical paper may not necessarily be substantially easier than producing it in the first place, especially for cutting-edge research. There would certainly be value in mining the tool’s output for interesting ideas, novel approaches, or unexpected connections; however, I could not treat the theorems themselves as established facts to build upon. I would need to spend my entire day, every day, checking these papers, only to discover that most of my effort went toward verifying work that was already correct, while the single flawed paper might pass undetected if its error is subtle enough.

One might object that this hypothetical 99% accuracy is already better than the current published literature! After all, most mathematicians have encountered papers with errors, gaps in arguments, or cases where “the details are left to the reader.” This is all true. But there is a crucial difference: with human-written papers, we develop intuitions about reliability. As Kevin Buzzard wrote recently [3]: “LLMs would be very happy to paper over any or all cracks in the argument or even lie in order to get to the claims of the result and as such I feel far less motivated to read the paper properly on the basis that it was not 100% generated by someone who believed that what they are writing is *true*, but rather by something who believed that what they are writing is *plausible*.” (Emphasis added.)

We develop intuitions about reliability through multiple signals: the quality of exposition, whether techniques are applied in standard ways, how well edge cases are handled, whether claims seem plausible given known results, etc., etc. A brilliant proof from an unknown mathematician can be immediately recognized as such precisely because we can evaluate these qualities. (For just one such example, see [27].) With AI producing 100 papers daily from a single source, these evaluation heuristics provide no differentiation; every paper must be treated with equal suspicion.

Moreover, even for the 99 correct papers, I cannot extract their full value without verification. A beautiful theorem might suggest new directions for research, but I cannot confidently build on it until I’ve checked the proof myself. The ideas might be worth mining, the conjectures worth pursuing, but the theorems themselves remain uncertain until verified. And at that point, why not just work out the mathematics myself?

This is not a problem that scaling can solve. Even if we improve the accuracy to 99.99% or 99.999%, the fundamental issue remains: for mathematics, a non-zero error rate creates an unusable tool. If we cannot distinguish the correct results from the flawed ones without verification, then that defeats the whole purpose of automation.

Compare this to a different scenario: an AI that produces only 10 papers per year, but each comes with a *formally verified proof*. Moreover – and this is crucial – suppose that I have learned to read formal statements well enough to verify that they say exactly what I want them to say, with no mistranslations or misinterpretations. *That* is a tool I would use enthusiastically.

The difference is deterministic certification. I can trust those 10 papers completely, build upon them confidently, and incorporate their results into my work without fear of hidden errors. The slower pace is not merely acceptable – it is a feature, not a bug. Quality and certainty matter far more than quantity when building a cumulative body of mathematical knowledge.

7 Formally Verified Mathematics.

Interactive Theorem Provers (also called “proof assistants”) are software systems that check every step of a mathematical proof against axioms and previously established results using the rules of logical inference. Such technology was envisioned in some form as early as Hilbert, if not Euclid, and has existed in practice since the late 1960s. Established systems include AUTOMATH, Mizar, Coq (now Rocq), Isabelle, HOL Light, and many others.

Among the most recent is Lean, which has been rapidly gaining adoption not just in computer science and mathematical logic circles, but in mainstream research mathematics. Many of the principles discussed here apply to any proof assistant, but I will focus on Lean and its ecosystem. (Full disclosure: I serve on the Strategic Advisory Board of the Lean Focused Research Organization.) One might worry that there are inherent limits to what can be expressed in Lean, but as Buzzard’s 2022 ICM Plenary Lecture [2] argues, the formal framework has already proved itself broad enough to accommodate essentially any mathematics we may want to formalize; let me not repeat the evidence here.

There are some common misconceptions about what the term “Lean” actually refers to, so let me be precise.

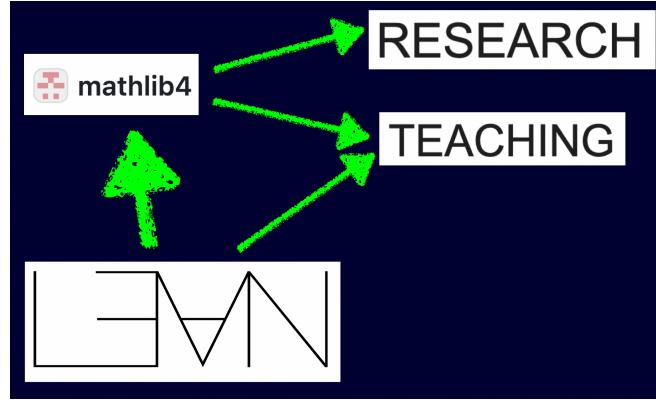


Figure 7.1: Different aspects of “Lean”

7.1 What is Lean? Core Lean (now technically Lean4) is software originally developed by Leonardo de Moura (then at Microsoft Research, now at Amazon Web Services) to certify bug-free code. Via the so-called Curry-Howard correspondence [4, 12], this same technology can certify that a mathematical statement has been proved, checking every step down to axioms. It has nothing, *a priori*, to do with AI.

The workflow is simple: humans write proofs in Lean’s formal language, line by line. After each line, Lean either accepts the step as valid or indicates an error. When a proof is complete, Lean displays “No goals”, meaning there are no remaining statements to prove.

Here is a simple example:

(a)

```

1 import Mathlib
2 -- definition of the limit of a sequence `a : N → R`
3 def SeqLim (a : N → R) (L : R) := ∀ ε > 0, ∃ N, ∀ n ≥ N, |a (n) - L| < ε
4
5 -- Theorem: If `a (n) = c` for all `n`, then the limit of `a` is `c`
6 theorem ConstLim (a : N → R) (c : R) (hypothesis : ∀ n, a (n) = c) :
7   | SeqLim a c := by -- begin the proof
8   | intro ε he -- let `ε` be given and assume it's positive (call this hypothesis `he`)
9   | intro 1 -- set `N` to any value, like `N = 1`
10  | intro n hn -- let `n` be given and assume that `n ≥ N` ← The cursor is currently here
11  | rewrite [hypothesis n] -- use the fact that `a (n) = c`
12  | -- the goal is now to prove that `|c - c| < ε`
13  | simp [he] -- this is trivial, since `ε > 0` (we called this hypothesis `he`)
14

```

Latest Mathlib Examples Load

Lean responds here with what it currently knows:

- ▼ MathlibDemo.lean:11:56
- ▼ Tactic state
- 1 goal
- ▼ case h
- a : N → R
- c : R
- hypothesis : ∀ (n : N), a n = c
- ε : R
- he : ε > 0
- n : N
- hn : n ≥ 1
- |a n - c| < ε

Assumptions Current Goal: All Messages (0)

(b)

```

1 import Mathlib
2
3 -- definition of the limit of a sequence `a : N → R`
4 def SeqLim (a : N → R) (L : R) := ∀ ε > 0, ∃ N, ∀ n ≥ N, |a (n) - L| < ε
5
6 -- Theorem: If `a (n) = c` for all `n`, then the limit of `a` is `c`
7 theorem ConstLim (a : N → R) (c : R) (hypothesis : ∀ n, a (n) = c) :
8   | SeqLim a c := by -- begin the proof
9   | intro ε he -- let `ε` be given and assume it's positive (call this hypothesis `he`)
10  | intro 1 -- set `N` to any value, like `N = 1`
11  | intro n hn -- let `n` be given and assume that `n ≥ N` ← Now the cursor is here, at the end of the proof.
12  | rewrite [hypothesis n] -- use the fact that `a (n) = c`
13  | -- the goal is now to prove that `|c - c| < ε`
14  | simp [he] -- this is trivial, since `ε > 0` (we called this hypothesis `he`)

```

Latest Mathlib Examples

No goals All Messages (0)

Lean agrees, the proof is complete

Figure 7.2: Proving a theorem in Lean.

7.2 What is Mathlib? Even if you have never tried to prove a significant theorem down to its axioms from scratch, you can surely imagine that it might be extraordinarily tedious and time consuming. To make any progress, we need a comprehensive library of previously proved results, stated in maximal generality to be useful in the widest range of situations. We also need powerful “tactics” – automated procedures that can solve entire

classes of subgoals – so that users can appeal to just a few such tactics in their proofs, rather than trying to memorize thousands of individual theorem names.

This is Mathlib: a massive, interconnected, and maintainable library built on Lean. Like Lean itself, Mathlib is entirely open source, enabling the collaborative effort of hundreds of contributors worldwide. When my mathematician friends complain that they looked at Mathlib and understood nothing despite being experts in the relevant field, I explain that Mathlib is *not* meant as an educational project. Its single-minded goal is comprehensiveness and scalability.

```

/-  

Copyright (c) 2021 Alex Kontorovich and Heather Macbeth and Marc Masdeu. All rights reserved.  

Released under Apache 2.0 license as described in the file LICENSE.  

Authors: Alex Kontorovich, Heather Macbeth, Marc Masdeu  

-/  

import Mathlib.Analysis.Complex.UpperHalfPlane.Basic  

import Mathlib.Data.Fintype.Parity  

import Mathlib.LinearAlgebra.Matrix.GeneralLinearGroup.Defs  

-!  

# Group action on the upper half-plane  

We equip the upper half-plane with the structure of a `GL (Fin 2) R` action by fractional linear  

transformations (composing with complex conjugation when needed to extend the action from the  

positive-determinant subgroup, so that `!![-1, 0; 0, 1]` acts as `z ↦ -conj z`.)  

-/  

noncomputable section  

open Matrix Matrix.SpecialLinearGroup UpperHalfPlane  

open scoped MatrixGroups ComplexConjugate  

namespace UpperHalfPlane  

-- The coercion first into an element of `GL(2, R)`+, then `GL(2, R)` and finally a 2 × 2  

matrix.  

This notation is scoped in namespace `UpperHalfPlane`. -/
scoped notation:1024 "tm" A:1024 =>  

  ((A : GL(2, R)) : GL (Fin 2) R) : Matrix (Fin 2) (Fin 2) _  

instance instCoeFun : CoeFun GL(2, R)+ fun _ => Fin 2 → Fin 2 → R where coe A := tmA  

-- The coercion into an element of `GL(2, R)` and finally a 2 × 2 matrix over `R`. This is  

similar to `tm`, but without positivity requirements, and allows the user to specify the ring `R`,  

which can be useful to help Lean elaborate correctly.  

This notation is scoped in namespace `UpperHalfPlane`. -/
scoped notation:1024 "tm[\" R \"]" A:1024 =>  

  ((A : GL (Fin 2) R) : Matrix (Fin 2) (Fin 2) R)  

-- Numerator of the formula for a fractional linear transformation -/
def num (g : GL (Fin 2) R) (z : C) : C := g 0 0 * z + g 0 1  

-- Denominator of the formula for a fractional linear transformation -/
def denom (g : GL (Fin 2) R) (z : C) : C := g 1 0 * z + g 1 1  

@[simp]  

lemma num_neg (g : GL (Fin 2) R) (z : C) : num (-g) z = -(num g z) := by  

  simp [num]; ring

```

Figure 7.3: A snippet from a sample file [15] in Mathlib

Consider someone working on algebraic groups of Lie type. The Manifolds library must integrate seamlessly with the libraries on Groups and Algebraic Geometry, with no “namespace” conflicts or incompatible conventions. The Mathlib maintainers work extremely hard to ensure that you can import all these libraries simultaneously and get to work.

Mathlib must scale from its current order of magnitude (roughly 2 million lines of code) to 10 million, to 100 million lines and beyond, while maintaining interactive responsiveness. If Lean takes more than a few seconds to respond after you write a line of code, it becomes unusable for research mathematics.

It’s worth emphasizing, as illustrated in Figure 7.1, that engaging with formal mathematics does not require contributing to Mathlib itself. If the mathematical foundations you need are already available in Mathlib, you can conduct research by building on top of the library without necessarily aiming to extend it. Similarly, Mathlib serves as a resource for teaching – though for pedagogical purposes, it’s often better to work directly from Lean’s core library, as Mathlib’s highly general and sophisticated treatments can be too advanced for students first learning the subject; see §10.

We will return to both of these issues in more detail later. For now, suffice it to say that the three applications – research using existing formalizations, teaching with appropriate levels of abstraction, and extending Mathlib with new mathematics – are distinct activities that all serve different communities and purposes. Mathlib’s role is to provide the foundational infrastructure that makes the other two possible.

7.3 The Library Gap and AI Assistance. Now we can recall and better appreciate the fundamental conundrum from the introduction: research mathematics grows at rate δ while Mathlib grows at rate ε , and currently $\delta > \varepsilon$. Mathlib’s 2 million lines of code represent an impressive achievement, but vast areas of mathematics remain unformalized.

This is where AI assistance becomes essential. Note that I’m now talking about a much more modest goal than proving new theorems: if AI can help with what’s called “autoformalization” – automatically converting already established natural language mathematics into formal proofs – then ε could at least get on par with δ . And then, once Mathlib is sufficiently comprehensive, we might finally get ε to actually exceed δ , allowing one to work directly in the formal system to discover new mathematics.

How might such AI-assisted formalization work in practice?

8 A Path Forward: Quasi-Autoformalization.

The answer, I believe, may lie in what I call “quasi-autoformalization”: a system of cooperating AI agents, orchestrated to translate natural language mathematics into formal proofs. The key word here is “quasi.” For my purposes as a research mathematician (rather than an AI researcher), I don’t require the system to work completely autonomously. I’m perfectly happy to intervene whenever I see a quick way to help prune the search tree and move the process along. My goal is not full automation but rather a collaborative and dramatically accelerated workflow. Here’s an overview, in broad strokes, before getting to the details:

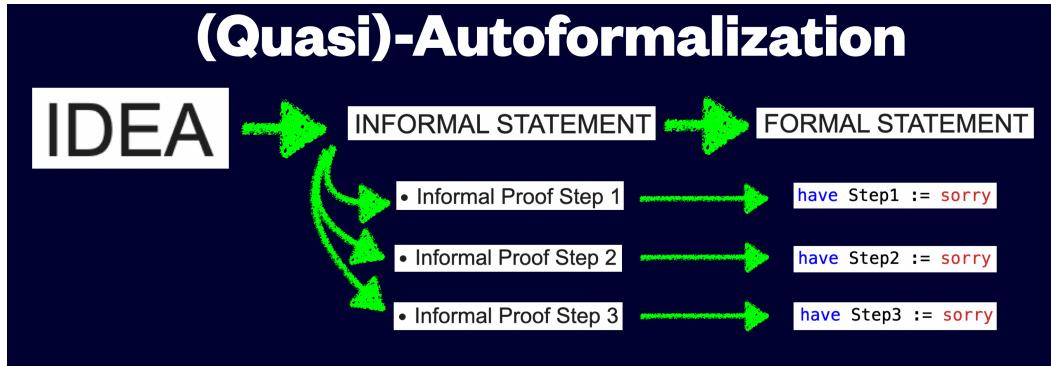


Figure 8.1: A schematic for Quasi-Autoformalization

Note that a variety of approaches to AI-assisted formalization have been explored in the past, from early work on automated theorem proving with human-readable output [7] to recent LLM-based systems like Draft-Sketch-Prove [13] (focused on solving new problems from scratch) and tools like Lean Copilot [22] (providing suggestions within the proof assistant). The quasi-autoformalization framework I propose differs in its explicit multi-agent decomposition with iterative refinement, designed specifically to accelerate Mathlib growth rather than (necessarily) achieve full autonomy.

8.1 The Architecture: Four Agents. The system consists of four components working together:

- **Decomposer:** Takes mathematics at the level of “Ideas” and refines it into natural language statement and proof steps at an appropriate level of granularity.
- **Translator:** Converts natural language statements and proof steps into formal Lean code, producing both the theorem statement and a “scaffold” of `have` statements (intermediate claims) that are initially `sorry`’d out (that is, their proofs are postponed without setting off Lean error messages).
- **Solver (or “Closer”):** Attempts to prove each `have` statement, replacing `sorry`s with complete formal proofs.

- **Conductor:** Orchestrates the entire process, identifying failures and deciding which components need to re-run with adjusted parameters.

Let's examine each component in detail.

8.1.1 The Decomposer: From Ideas to Scaffold. Natural language mathematics, whether in undergraduate textbooks or arXiv papers, is really written at the level of “Ideas.” Even something as clear (to a trained mathematician) as a proof in Baby Rudin is nowhere near precise enough to formalize directly.

1.1 Example We now show that the equation

$$(1) \quad p^2 = 2$$

is not satisfied by any rational p . If there were such a p , we could write $p = m/n$ where m and n are integers that are not both even. Let us assume this is done. Then (1) implies

$$(2) \quad m^2 = 2n^2,$$

This shows that m^2 is even. Hence m is even (if m were odd, m^2 would be odd), and so m^2 is divisible by 4. It follows that the right side of (2) is divisible by 4, so that n^2 is even, which implies that n is even.

The assumption that (1) holds thus leads to the conclusion that both m and n are even, contrary to our choice of m and n . Hence (1) is impossible for rational p .

Figure 8.2: Taken from Rudin [21, p.2]

Formalizing this simple argument requires many more steps that are hidden in natural language because they're too obvious to a human to state but are necessary for completeness, such as: why can we make it so that m and n are not both even? Why if m is not of the form $2k$, then it must be of the form $2k + 1$? Etc.

The Decomposer's job is to break down mathematical ideas into sufficiently small, explicit steps that can be formalized. The granularity depends on what's already in Mathlib and on how sophisticated the Solver is. This is an iterative process: if the Solver fails, the Conductor may ask the Decomposer to break things down further. Figure 8.3 shows a sample decomposition in this example, by Claude AI.

8.1.2 The Translator. The Translator's job is to take the Decomposer's small natural language lemmas, and convert each into a small formal Lean `have` statement, giving a formal scaffold of the main theorem. Figure 8.4 shows what it might look like in our running example, again using Claude AI. We postpone to §8.3.2 a more detailed discussion of the manifold difficulties this particular agent will encounter.

8.1.3 The Solver: Closing Goals. Once the Translator produces a scaffold with `sorry`'d `have` statements, the Solver attempts to clear each `sorry`. This is where automated theorem proving systems demonstrate their power: they can search through vast spaces of possible proof strategies, chaining together lemmas from Mathlib, and applying tactics until goals are closed.

The Solver doesn't need to succeed on every goal. Some `sorry`s may remain, requiring either human intervention, further decomposition by the Decomposer, or refinement of the statement by the Translator. The system's value lies in automatically closing the goals that *can* be closed, dramatically reducing the human effort required.

Different solving strategies may be appropriate for different types of goals. Simple algebraic manipulations might yield to basic tactics, while more subtle arguments may require sophisticated search procedures or even reinforcement learning approaches trained on large corpora of formal proofs. Figure 8.5 shows a Closer applied to our example (with some help from this human).

8.1.4 The Conductor: Orchestrating the Workflow. The Conductor is the system's coordinator. When formalization fails (as it inevitably will on first attempts), the Conductor must diagnose why:

- Is a goal beyond the Solver's current capabilities? Flag for further decomposition.
- Was a statement mistranslated? Have the Translator try again with additional context.
- Does this result perhaps already exist in Mathlib? Perform a Google search (via RAG – *retrieval augmented generation* – or other methods) to check whether the theorem (or something closely related) is already formalized in the latest version of the library.

A Take this example theorem and decompose it into a series of smaller Lemmas, each being extremely elementary, so that I will (later) be able to easily formalize each of the Lemmas individually, and so that the formalized Lemmas combine together to prove the main result

Here's a decomposition of the irrationality of $\sqrt{2}$ into elementary lemmas suitable for formalization:

Main Theorem

Theorem: There is no rational number p such that $p^2 = 2$.

Lemmas

Lemma 1 (Rational representation): If p is a rational number, then there exist integers m and n with $n \neq 0$ such that $p = m/n$.

Lemma 2 (Coprime representation): If p is a rational number, then there exist integers m and n with $n \neq 0$ such that $p = m/n$ and $\gcd(m,n) = 1$.

Lemma 3 (Equation manipulation): If $m/n = p$ and $p^2 = 2$, then $m^2 = 2n^2$.

Lemma 4 (Even square implies even): If m^2 is even, then m is even.

Lemma 5 (Even number form): If m is even, then there exists an integer k such that $m = 2k$.

Lemma 6 (Substitution): If $m = 2k$, then $m^2 = 4k^2$.

Lemma 7 (Divisibility transitivity): If $m^2 = 2n^2$ and $m^2 = 4k^2$, then $2n^2 = 4k^2$.

Lemma 8 (Division by 2): If $2n^2 = 4k^2$, then $n^2 = 2k^2$.

Lemma 9 (Even conclusion): If $n^2 = 2k^2$, then n^2 is even.

Lemma 10 (Second application): If n^2 is even, then n is even (instance of Lemma 4).

Lemma 11 (Contradiction with coprimality): If both m and n are even, then $\gcd(m,n) \geq 2$.

Lemma 12 (Proof by contradiction): If assuming $p^2 = 2$ for rational p leads to a contradiction, then no such rational p exists.

Figure 8.3: A decomposition of Rudin's argument using Claude AI.

The Conductor also manages computational resources, deciding when to re-run components versus requesting human input. In the “quasi” approach, a human can serve as super-Conductor, making these decisions based on mathematical insight.

In the case at hand (see Figure 8.6), the Conductor’s RAG lookup of the library could have found a theorem called `irrational_sqrt_two`, thereby reducing 140 lines of code to a single citation of Mathlib’s much more elegant proof.

8.2 Optimism... Does this workflow actually deliver? The first time I drew the schematic in Figure 8.1, as soon as it became clear in my mind, was in May 2025 at the Institute for Advanced Study (where I was spending my sabbatical) over lunch with DeepMind’s Thomas Hubert, a leader on the AlphaProof project, and Princeton’s Sanjeev Arora and Chi Jin, leaders of the open source Goedel Prover [17, 18]. Both AlphaProof and Goedel Prover represent the current state of the art in automated theorem proving, with the latter competing in the Open Source category. The day before, I had the opportunity to test AlphaProof with Thomas against some outstanding tasks in the “PrimeNumberTheoremPlus Project” (PNT+) which Terry Tao and I co-organize [16]. The project’s goal is to formalize significant results in analytic number theory, including the prime number theorem, Dirichlet’s theorem on primes in progressions, and the Chebotarev density theorem. Seeing a real “Closer” in action for the first time made the potential of this approach suddenly concrete.

The Solver faces a genuinely difficult task: even goals that seem trivial to a human mathematician can require intricate formal reasoning. For example, I had scaffolded a lemma requiring a `have` statement that the Riemann zeta function is at least 1 in absolute value in a neighborhood of $s = 1$ (excluding the point itself). This should be “immediate” to a human: it follows from `riemannZeta_residue_one`, the theorem already in Mathlib stating that ζ has a pole at $s = 1$ (with residue 1). But if proving that $\sqrt{2}$ is irrational took 140 lines of formal code, one can only imagine how much more complicated this “trivial” task becomes.

And yet AlphaProof succeeded brilliantly! This despite being trained only on high school IMO problems (which never involve “eventually filters,” limits, or complex analysis), it chained together an extremely long

```

1 import Mathlib
2
3 theorem sqrt2_irrational : Irrational (Real.sqrt 2) := by declaration uses `sqrt2_irrational
4 -- Assume for contradiction that √2 is rational
5 by_contra h
6
7 -- Lemma 1+2: If √2 is rational, it can be written as m/n in lowest terms
8 with n > 0
9 have lemma12 : ∃ m : ℤ, ∃ n : ℕ, 0 < n ∧ Real.sqrt 2 = m / n ∧ Int.gcd m.
10 natAbs n = 1 := by sorry
11
12 obtain (m, n, hn_pos, hsqrt2_eq, hcoprime) := lemma12
13
14 -- Since √2 > 0 and equals m/n, we must have m > 0, so we can use m.natAbs
15 have m_pos : 0 < m := by sorry
16
17 -- Lemma 3: If p = m/n, then p² = m²/n²
18 have lemma3 : (Real.sqrt 2) ^ 2 = (m : ℝ) ^ 2 / (n : ℝ) ^ 2 := by sorry
19
20 -- Lemma 4: Cross multiplication gives m² = 2n²
21 have lemma4 : (m : ℝ) ^ 2 = 2 * (n : ℝ) ^ 2 := by sorry
22
23 -- Convert to natural number equation
24 have m_sq_eq : m ^ 2 = 2 * n ^ 2 := by sorry
25
26 -- Lemma 5: If k² is even, then k is even
27 have lemma5 : ∀ k : ℤ, Even (k²) → Even k := by sorry
28
29 -- Lemma 6: If k is even, then k² is divisible by 4
30 have lemma6 : ∀ k : ℤ, Even k → 4 ∣ k² := by sorry
31
32 -- Show m is even
33 have m_even : Even m := by sorry
34
35 -- Show m² is divisible by 4
36 have m_sq_div_4 : 4 ∣ m² := by sorry
37
38 -- Lemma 7: Divisibility transitivity for our specific case
39 have lemma7 : 4 ∣ m² → m² = 2 * n² → 4 ∣ (2 * n²) := by sorry
40
41 -- Lemma 8: If 4 divides 2n², then 2 divides n²
42 have lemma8 : 4 ∣ (2 * n²) → 2 ∣ n² := by sorry
43
44 have two_div_n_sq : 2 ∣ n² := by sorry
45
46 have even_n_sq : Even (n²) := by sorry
47
48 -- Lemma 9: If 2 divides n², then n is even
49 have lemma9 : Even (n²) → Even n := by sorry
50
51 have n_even : Even n := by sorry
52
53 -- Lemma 10: If both m and n are even, then gcd(m,n) ≥ 2
54 have lemma10 : Even m → Even n → 2 ≤ m.gcd n := by sorry
55
56 have gcd_ge_2 : 2 ≤ m.gcd n := lemma10 m_even n_even
57
58 -- Lemma 11: This contradicts coprimality
59 have lemma11 : ¬(2 ≤ m.gcd n) := by sorry
60
61 -- Final contradiction
62 exact lemma11 gcd_ge_2

```

Figure 8.4: A translation of the natural language statement and lemmas into a scaffold for a formal proof

sequence of formal steps that Lean accepted. The resulting proof was barely comprehensible, even to an expert human formalizer, see Figure 8.7.

This seems to reflect AlphaProof’s training mechanism: adapting the AlphaZero reinforcement learning protocol, it “plays” mathematics against itself, discovering highly non-standard patterns for proving theorems through self-improvement. Since my work wasn’t meant to directly contribute to Mathlib (yet), I didn’t care what the formal proof looked like, as long as the Lean code compiled. Since I knew the result followed trivially in natural language from already-proved theorems, Lean’s acceptance was sufficient, and I could move on to the next goal.

In September 2025, shortly before this document’s completion, Morph AI (now Math, Inc.), led by Jesse Han, Jared Lichtman, and Christian Szegedy, announced another significant milestone: their “Gauss” agent autoformalized in Lean the classical error term for the Prime Number Theorem. It built on the foundations of PNT+, and made substantial progress formalizing complex analysis results such as the Borel-Carathéodory theorem and extending it to logarithmic derivatives of meromorphic functions. In their experiment, a mathematician (Jared) served as Decomposer while Gauss handled translation and solving autonomously – a division of labor that may itself become fully automated in future iterations.

While these successes point to the framework’s viability, significant challenges remain.

8.3 ... and Difficulties. Of course it is usually not so straightforward to get this workflow to succeed; let’s highlight a number of particular difficulties.

8.3.1 The Challenge of Closing Goals. As we saw with AlphaProof’s success on the zeta function goal, closing formal goals is possible but requires sophisticated systems. A particular engineering feature proved especially valuable here:

Given any goal, AlphaProof simultaneously searches for a proof of the statement *and its negation*. Their original motivation for engineering this was practical. Many IMO problems ask for answers, not just proofs. If you don’t know the precise statement you’re trying to prove, you cannot even begin working formally. For example, a problem might ask for the set of integers satisfying some property. You might formalize this statement as $\exists S \subseteq \mathbb{Z}, S = \{n \in \mathbb{Z} \mid \text{property}(n)\}$. But this challenge is *trivially* solvable: just set $S := \{n \in \mathbb{Z} \mid \text{property}(n)\}$, and by definition, it’s equal to itself! To make a nontrivial statement requires first determining what S is, say, $S = \{1, 2, 4\}$, and only then trying to prove that $\{1, 2, 4\} = \{n \in \mathbb{Z} \mid \text{property}(n)\}$.

AlphaProof’s approach: send the natural language problem to an LLM like Gemini, requesting roughly 100 potential solutions ($S = \{1, 2, 3\}$, $S = \{2, 3, 4\}$, etc.). For each guess, AlphaProof attempts to prove or disprove the claim. In practice, it quickly disproves approximately 98 of them, then focuses on the remaining candidates.

```

1 import Mathlib
2
3 theorem sqrt2_irrational : Irrational (Real.sqrt 2) := by
4   assume for contradiction that v2 is rational
5   by_contra h
6
7   -- Lemma 1+2: If v2 is rational, it can be written as m/n in lowest terms with n > 0
8   have lemma12 : ∃ m : ℤ, ∃ n : ℕ, 0 < n ∧ Real.sqrt 2 = m / n ∧ Int.gcd m.natAbs n = 1 := by
9     -- Use that v2 is rational means it equals some q : 0
10    -- Then use q.num : ℤ and q.den : ℕ which are already in lowest terms
11    rw [Irrational, not_not] at h
12    obtain (r, hr) := h
13    -- Use the numerator and denominator of r, which are already in lowest terms
14    use r.num, r.den
15    constructor
16    · -- r.den ≠ 0 by construction
17      exact Rat.den_pos r
18    constructor
19    · -- v2 = r.num / r.den
20      rw [hr]
21      simp [Rat.cast_def]
22      · -- gcd(r.num), |r.den| = 1 because rationals are stored in lowest terms
23        exact Rat.reduced r
24
25    obtain (m, n, hn_pos, hsqrt2_eq, hcoprime) := lemma12
26
27    -- Since v2 > 0 and equals m/n', we must have m > 0, so we can use m.natAbs
28    have m_pos : 0 < m := by
29      -- We know v2 > 0
30    have hsqrt2_pos : 0 < Real.sqrt 2 := Real.sqrt_pos.mpr (by norm_num : (0 : ℝ) < 2)
31    -- From the equation v2 = m / n', we get m / n' > 0
32    rw [hsqrt2_eq] at hsqrt2_pos
33    have : (0 : ℝ) < m := by
34      apply div_pos_if.mp hsqrt2_pos |> elim
35      · exact fun (h1, h2) => h1
36      · exact fun (h1, h2) => by exfalso; linarith
37      exact_mod_cast this
38
39    -- Lemma 3: If p = m/n, then p² = m²/n²
40    have lemma3 : (Real.sqrt 2) ^ 2 = (m : ℝ) ^ 2 / (n : ℝ) ^ 2 := by
41      rw [hsqrt2_eq]
42      ring
43
44    -- Lemma 4: Cross multiplication gives m² = 2n²
45    have lemma4 : (m : ℝ) ^ 2 = 2 * (n : ℝ) ^ 2 := by
46      field_simp at lemma3
47      rw [← lemma3]
48
49    -- Convert to natural number equation
50    have m_sq_eq : m ^ 2 = 2 * n ^ 2 := by exact_mod_cast lemma4
51
52    -- Lemma 5: If k² is even, then k is even
53    have lemma5 : ∀ k : ℤ, Even (k^2) → Even k := by
54      intro k hk
55      contrapos! hk
56      -- hk : -Even k, so k is odd
57      rw [Int.not_even_iff_odd] at hk ⊢
58      -- If k is odd, then k² is odd
59      rw [Int.odd_pow]
60      exact Or.intro_left (2 = 0) hk
61
62    -- Lemma 6: If k is even, then k² is divisible by 4
63    have lemma6 : ∀ k : ℤ, Even k → 4 ∣ k² := by
64      intro k hk
65      obtain (l, hl) := hk
66      use l ^ 2
67      rw [hl]
68      ring
69
70
71    -- Show m² is divisible by 4
72    have m_sq_div_4 : 4 ∣ m² := lemma6 m_even
73
74
75    -- Lemma 7: Divisibility transitivity for our specific case
76    have lemma7 : 4 ∣ m² → m² = 2 * n² → 4 ∣ (2 * n²) := by
77      intro hi hz
78      rw [hz] at hz
79      exact_mod_cast hz
80
81    -- Lemma 8: If 4 divides 2n², then 2 divides n²
82    have lemma8 : 4 ∣ (2 * n²) → 2 ∣ n² := by
83      intro h
84      obtain (l, hl) := h
85      use l
86      -- Now we have 2 * n ^ 2 = 2 * (2 * l)
87      have : 2 * n ^ 2 = 2 * (2 * l) := by convert hl using 1; ring
88      -- Cancel the factor of 2 from both sides
89      exact Nat.eq_of_mul_eq_mul_left (by norm_num : 0 < 2) this
90
91    have two_div_n_sq : 2 ∣ m² := lemma8 (lemma7 m_sq_div_4 m_sq_eq)
92
93    have even_n_sq : Even (n^2) := by exact
94      even_if_exists_two_nsmp (n ^ 2).mp (lemma8 (lemma7 (lemma6 m_m_even) m_sq_eq))
95
96    -- Lemma 9: If 2 divides n², then n is even
97    have lemma9 : Even (n^2) → Even n := by
98      exact_mod_cast lemma5 n
99
100   have n_even : Even n := lemma9 even_n_sq
101
102   -- Lemma 10: If both m and n are even, then gcd(m,n) ≥ 2
103   have lemma10 : Even m → Even n → 2 ≤ m.gcd n := by
104     intro hm hn
105     -- Since m is even, 2 divides m
106     -- Since m is even, 2 divides m
107     have h2_div_m : 2 ∣ m := by
108       obtain (l, hl) := hm
109       use l
110       convert hl using 1
111       ring
112       -- Since n is even, 2 divides n
113       have h2_div_n : 2 ∣ (n : ℤ) := by
114         obtain (l, hl) := hn
115         use l
116         convert hl using 1
117         ring
118         -- Since 2 divides both m and n, 2 divides gcd(m, n)
119         have h2_div_gcd : (2 : ℤ) ∣ m.gcd n := by exact_mod_cast Int.dvd_gcd h2_div_m h2_div_n
120         -- The gcd is positive (since we need to show it's ≥ 2)
121         have hgcd_pos : (0 : ℤ) < m.gcd n := by
122           norm_cast
123           exact Nat.lt_of_sub_eq_succ hcoprime
124           -- Since gcd > 0 and 2 divides it, gcd ≥ 2
125           exact_mod_cast Int.le_of_dvd hgcd_pos h2_div_gcd
126
127   have gcd_ge_2 : 2 ≤ m.gcd n := lemma10 m_even n_even
128
129   -- Lemma 11: This contradicts coprimality
130   have lemma11 : ¬(2 ≤ m.gcd n) := by
131     have : m.gcd n = 1 := by exact_mod_cast hcoprime
132     rw [this]
133     simp
134
135   -- Final contradiction
136   exact lemma11 gcd_ge_2
137
138
139
```

Figure 8.5: The Closer successfully solved all the remaining goals.

```

1 import Mathlib
2
3 theorem sqrt2_irrational : Irrational (Real.sqrt 2) := by
4   exact irrational_sqrt_two

```

Figure 8.6: A search of the library should find this result.

```

have ZetaBlowsUp : ∀f s in ℒ[≠](1 : ℂ), ||ζ s|| ≥ 1 := by
  simp_all[Function.comp_def,eventually_nhdsWithin_iff,norm_eq_sqrt_real_inner]
  contrapose! h
  simp_all
  delta abs att*
  exfalso
  simp_rw [Metric.nhds_basis_ball.frequently_iff]at*
  choose! I A B using h
  choose a s using exists_seq.StrictAnti_tendsto (0: ℝ)
  apply(isCompact_closedBall _ _).isSeqCompact fun and=>(A _ (s.2.1 and)).le.trans
  | (s.2.2.bddAbove_range.some_mem (and, rfl))).elim
  use fun and (a, H, S, M)=>absurd (tendsto_nhds_unique M (tendsto_sub_nhds_zero_iff.1
  | ((squeeze_zero_norm fun and=>le_of_lt (A _ (s.2.1 _)) ) (s.2.2.comp S.tendsto_atTop)))
  fun and=>?
  norm_num[*,Function.comp_def] at M
  have:=@riemannZeta_residue_one
  use one_ne_zero (tendsto_nhds_unique (this.comp (tendsto_nhdsWithin_iff.2 ( M,.of_forall
  | (by norm_num[*])))) ( squeeze_zero_norm ?_ (M.sub_const 1).norm.trans
  | (by rw [sub_self,norm_zero]))))
  use fun and =>.trans (norm_mul_le_of_le ↑(le_rfl) (Complex.norm_def _>Real.sqrt_le_one.mpr
  (B ↑_ (s.2.1 ↑_)).right.le)) (by rw [mul_one])

```

Figure 8.7: AlphaProof’s solution to a `have` statement, from [16]

This feature proved invaluable in our experiments for a completely different reason: it catches translation errors.

8.3.2 The Subtleties of Translation. The Translator component presents particularly subtle challenges. Getting formal statements to say *exactly* what we intend to is sometimes as hard as proving them! In my experiments with AlphaProof on the PNT+ Project, embarrassingly many of the `have` statements I scaffolded were resoundingly *disproved* by the system. Each time, I could usually quickly identify what assumption I had omitted, adjust the statement, and iterate. This back-and-forth between proposing statements and checking whether they can be established (or refuted) was essential to getting the formalization right.

For an illustration, consider formalizing the Riemann zeta function. Many think of it as $\zeta(s) = \sum_{n=1}^{\infty} 1/n^s$, but this is really the Euler (or Bernoulli) “zeta function”. Riemann’s insight was that the “right” way to define the zeta function was by first making a theta function, taking its Mellin transform, and dividing by Gamma:

$$(8.1) \quad \zeta(s) := \frac{\pi^{s/2}}{\Gamma(s/2)} \left[\int_1^\infty \left(2 \sum_{n=1}^{\infty} e^{-\pi n^2 u^2} \right) (u^s + u^{1-s}) \frac{du}{u} - \frac{1}{1-s} - \frac{1}{s} \right]$$

This agrees, for $\Re(s) > 1$, with the series, and hence represents its meromorphic continuation.

Now, in most Interactive Theorem Provers, including Lean, functions are *total* (that is, defined on all inputs). So although you’re not supposed to divide by zero, the expression $1/0$ must have some value (called a “junk” value, because you should never have to know what it is); it cannot be left undefined. In Mathlib, the junk value assigned to $1/0$ is 0. This makes it possible to prove theorems like $(a+b)/c = a/c + b/c$ without requiring a proof that $c \neq 0$; the junk value is such that the statement is true either way. The rationale for this is simple and very practical: If you required denominators to be nonzero before you allowed division, you would quickly be swamped with a cascade of trivial theorems needing to be proved again and again, for every arising division. Instead, the practice is to push the need to prove nonzeroness as late as possible; for example, the theorem that $a/b \times b = a$ does (as it must) require as a hypothesis that $b \neq 0$.

What does this have to do with the zeta function? David Loeffler and Michael Stoll formalized it and other L -functions for Mathlib [19], and discovered the following very curious phenomenon. Since functions are total, the zeta function maps all of \mathbb{C} to \mathbb{C} , and hence has some junk value at its pole at $s = 1$. What is the value?

While the question seems trivial, note that if that junk value were zero, the Riemann Hypothesis, as is commonly stated would be trivially false! The standard statement says that zeros in the critical strip (say $\Re(s) \in [0, 1]$, avoiding the trivial zeros at negative even integers) satisfy $\Re(s) = 1/2$. But if $\zeta(1) = 0$, we

would have a counterexample. Imagine an AI announcing that it has disproved the Riemann Hypothesis with a petabyte-long incomprehensible proof whose essence is that $\zeta(1) = 0$.

Fortunately, Loeffler and Stoll worked out carefully how the junk value propagates through the construction (8.1), and found that $\zeta(1) \neq 0$. So the standard statement of RH requires no adjustment. But this example illustrates just how subtle and treacherous translation can be.

When I discussed these translation difficulties with Christian Szegedy – a visionary scientist who was among the first to strongly advocate for autonomous formalization as a means of building large mathematical libraries [24], and who leads the team developing the Gauss agent mentioned above – his response shocked me.

8.3.3 Mathematics Is Robust. Christian told me, “It doesn’t matter if the formalized statements and definitions are wrong! Don’t you *believe* in mathematics?”

His point was profound: mathematics, in practice, is *robust*. Definitions and theorems emerge through years of refinement, not arbitrary invention. We routinely get things wrong initially and refine the theory when necessary.

There are numerous examples in the history of mathematics that illustrate this point. In topology, we started by working with intervals on the real line (open and closed). Then we realized that this had nothing to do with the real line, and extended the idea to balls in arbitrary metric spaces. Eventually, Hausdorff showed that you don’t need a metric at all! You simply declare which sets are open and require the right axioms, and can do abstract topology. Each iteration (“refactor”) improved our understanding of the notion of Openness.

Gauss’s original statement of the Class Number One Problem wasn’t quite right [8]. The Poincaré conjecture wasn’t at first stated exactly as we now understand it. Mathematicians regularly get statements slightly wrong, even if the spirit of the idea is eventually validated. As long as we have mechanisms to refactor, we recover correct formulations *in the long run*.

Christian’s point was that the same will hold for formalization. Mathematics is self-correcting; wrong statements that are useful will be discovered and corrected through use! (And wrong statements that are useless won’t be corrected, but they’ll be harmless since no one is building on them.)

For a concrete instance of this, imagine stating and proving the so-called Archimedean Property (that no matter how small a real number $\varepsilon > 0$ may be, you can always find a natural number N large enough that $1/N$ is even smaller) as follows:

```
theorem ArchimedeanProperty : ∀ (ε : ℝ), ε > 0 → ∃ (N : ℕ), 1 / N < ε
```

And suppose that you want at some later point to use this fact to prove that the sequence $a_n = 1/n$ converges to 0:

```
def SeqLim (a : ℕ → ℝ) (L : ℝ) := ∀ ε > 0, ∃ N, ∀ n ≥ N, |a (n) - L| < ε

theorem OneOverN (a : ℕ → ℝ) (hypothesis : ∀ n, a (n) = 1 / n) : SeqLim a 0
```

You will at some point need to leverage the fact that $n \geq N$ into the fact that $1/n \leq 1/N$. But this is not, in general, true: if $N = 0$, then $1/N = 0$, as we discussed already; so the latter inequality can fail! The culprit is the statement of `ArchimedeanProperty`, which should insist on N being strictly positive. In fact, the version stated above has the following vacuous proof:

```
theorem ArchimedeanProperty : ∀ (ε : ℝ), ε > 0 → ∃ (N : ℕ), 1 / N < ε := by
  intro ε hε
  use 0
  simp_all
```

That is, let ε be given and assume (`hε`) that $\varepsilon > 0$. Set $N = 0$; then $1/N < \varepsilon$ follows from the assumptions.

Again, it is these kinds of situations (and others in much less obvious contexts!) that make me feel that, as a mathematician, I must hold *myself* personally responsible for any formal statements, whether translated by hand or by machine. Christian’s counterpoint is intriguing: he claims that human mathematicians don’t need to learn Lean at all; they can simply let autoformalizers do their thing, and trust that *in the long run*, the important issues will sort themselves out.

However, recent examples of agentic AI behavior suggest caution about such trust. When I asked Claude to compute 0.99^{1000} , it correctly recognized this as a deterministic question, wrote perfect Python code to perform the calculation, executed it internally, and reported the exact result. This demonstrates how tool use can bridge

the stochastic-deterministic gap. But during the aforementioned DeepMind visit to IAS, we were told about a more troubling example: Gemini was asked to compute the genus of a particular curve. It produced perfect Sage code and reported that the genus was one. But the mathematicians asking the question knew that the correct answer was two! Had they just discovered a bug in Sage? Upon further investigation, they found the following. Their experimental version of Gemini was equipped with a safety feature to disable external tool access, and it was accidentally turned on; so Gemini *never called* Sage at all! The model was being trained to report Sage’s result but couldn’t, so it decided to simply **pretend** to execute the Sage code and report its best guess at the result!

This illustrates why, while I value and respect Christian’s perspective, I cannot endorse it myself: even when an AI claims to have used formal verification tools, we cannot blindly trust that it actually did so. Whether one adopts my cautious view or Christian’s optimistic one may depend on temperament, but both viewpoints agree on the fundamental goal: accelerating the growth of formalized mathematics.

9 The Adoption Question: When Will Formal Mathematics Win?

If the above pans out, then in the near future, Mathlib will grow large and robust enough that I can finally not only state and prove my theorems in Lean, but actually work on original research directly in the formal system, rather than first discovering results in natural language and then attempting to translate them. Will others follow suit? The path to widespread adoption may follow a pattern we’ve seen before. Consider what happened with \LaTeX .

9.1 The “Knuth Factor”. In 1978, Don Knuth released \TeX . At that time, mathematicians wrote their papers by hand and gave them to secretaries for typesetting. After waiting weeks or months, they received something approximating their original text, then iterated a few times (if they cared to), correcting typos introduced in the process.

In the 1980s, Mike Spivak promoted AMST\TeX , yet still relatively few mathematicians adopted it. The system remained too cumbersome for all but the most dedicated adherents. By the mid-to-late 1980s and early 1990s, \LaTeX emerged with beautiful, easy-to-use macros and automation. Almost everybody switched to self-typesetting. Eventually Overleaf arrived, eliminating (for many) the need to install software locally. You could do everything in a browser using a free web application.

But \LaTeX adoption wasn’t driven solely by efficiency in producing final documents. It became an organizational tool for the research process itself. When working on a substantial result, mathematicians make incremental progress on various lemmas, subcases, and technical estimates. Keeping track of what has been established, how pieces interconnect, and which gaps remain becomes unwieldy when scattered across handwritten notes. \LaTeX allows you to maintain a living document that evolves with your understanding, where you can easily reorganize arguments, insert new lemmas in the logical sequence, and cross-reference results as the proof architecture develops. The act of typesetting becomes part of mathematical thinking, not merely its final presentation.

I define the “Knuth factor” as the ratio of time required to develop and document a mathematical result using \LaTeX (once you’ve learned the syntax with its dollar signs, backslashes, and curly brackets) to the time required to develop and document the same result by hand. Around 1990, the Knuth factor dropped below 1. Shortly thereafter, nearly everyone switched – without any coercion – simply because it was the obvious way to speed up their workflow.

I expect the same will happen with Lean. Here’s how.

9.2 The Formalization Factor. In formalization literature, there’s the so-called “de Bruijn factor,” which measures the ratio of the number of lines of formal code to lines of natural language proof [5, 26]. Estimates vary, but factors of 4-10 are commonly cited. However, this metric, I believe, misses the point.

Lines of code are no longer a meaningful proxy for human effort. LLMs can generate thousands of lines of code quickly, and automated solvers can fill in proofs that would have taken humans hours. What matters is not how many *lines* are produced, but the amount of *time* it takes mathematicians to do their work.

The proposed alternative metric is what I’ll call the “formalization factor”: the ratio of time required to discover and formalize a mathematical result (from initial idea through verified formal proof) to the time required to discover and write up the same result in natural language mathematics, typeset in \LaTeX . Crucially, this metric acknowledges that formal systems may accelerate not just the verification phase, but the discovery process itself. Just as \LaTeX became a tool for organizing evolving mathematical arguments, formal systems could provide

structure and error-checking that accelerates the incremental development of complex proofs while making them more reliable.

This factor currently sits well above 1 – perhaps 10, perhaps 100, depending on the subfield and the mathematician’s familiarity with Lean. The difficulty isn’t just writing proofs; it’s that vast areas of mathematics aren’t yet formalized enough to even state your theorems. You must either build the foundations yourself (prohibitively expensive) or wait for Mathlib to grow.

But once that factor drops below 1, I expect nearly everyone will voluntarily switch to working formally, just as they did with L^AT_EX; no coercion will be necessary. It will simply be the obvious thing to do to speed up your workflow and increase confidence in your results.

9.3 Conditions for Adoption. For the formalization factor to drop below 1, several conditions must be met:

First, Mathlib must be comprehensive enough that stating your theorem requires minimal foundational work. This is where quasi-autoformalization becomes essential: AI assistance can accelerate library growth so that $\varepsilon > \delta$, ensuring formalized mathematics keeps pace with (or exceeds) natural language mathematics. But library growth requires more than just correct proofs – it demands polished, maintainable code. The AI must learn to write reusable mathematics that integrates cleanly with existing infrastructure, not merely produce working but unmaintainable formal arguments.

Second, the tools must become more accessible. Installation should be effortless. Imagine a browser-based environment like Overleaf, but for Lean, where you simply open a page and start working. (This nearly exists with platforms like live.lean-lang.org and GitHub Codespaces, though significant friction remains.)

Third, AI assistance must handle the tedious parts. Natural language interfaces could help write formal code; automated solvers could close routine goals; translation tools could convert mathematical exposition to formal scaffolds. The mathematician focuses on the creative work, identifying the key ideas and structuring the argument, while AI handles the mechanical formalization.

Fourth, and perhaps most importantly, the verification benefit must become immediate and tangible. Currently, formalization is an investment whose payoff is distant: you formalize now so that others can build on your work later, or so that you’re certain your proof is correct. But if working formally means catching errors early, avoiding tedious case-checking, and confidently building on others’ results, then the workflow itself becomes more efficient. The formalization factor drops not just because formalization gets faster, but because the entire research process improves.

When will this factor drop below 1? That depends primarily on the first condition: comprehensive libraries. With sustained AI-assisted formalization, Mathlib could reach the necessary scale within 5-10 years for many core areas of mathematics. For certain highly specialized fields, it may take longer. But once your field crosses the threshold, the switch could be rapid.

10 Teaching.

The adoption timeline I’ve outlined focuses on research mathematics, but teaching represents a parallel and potentially accelerating force. Among the most rapid converts to formalization are young people. To see why, imagine a world in which we tried to teach newcomers to chess purely by *describing* moves in sequence, say using algebraic notation. The teacher says:

Ok class, the game went like this: 1. Nf3 Nf6 2. c4 g6 3. Nc3 Bg7 4. d4 0-0 5. Bf4 d5 (this is a transposed Grünfeld Defence) 6. Qb3 dxc4 7. Qxc4 c6 8. e4 Nbd7 9. Rd1 Nb6 10. Qc5 Bg4 11. Bg5 Na4!! Wow, can you believe that move?!

Chess aficionados can effortlessly translate these symbols into an evolving series of positions, updating the board in their heads at every move. But for the rest of us, it’s vastly easier if you just *show me the board!*

And yet this imagined world is *precisely* how we currently teach almost all newcomers to theorem proving: entirely in natural language. At every step of a mathematical proof, the “game board” – meaning the current objects, assumptions, and goal – is changing. Professional mathematicians track this effortlessly (System I), but for beginners it requires deliberate, error-prone effort (System II). Think back to our Baby Rudin example: when proving that $\sqrt{2}$ is irrational, the underlying game board is shifting at each line of reasoning.

Of course, we would never dream of writing out the full board position after every single move in a natural-language proof. First, it would be unbearably cumbersome. Second, most of us got along “just fine” without it.

In fact, one might argue that developing the ability to construct these invisible boards in one's head is *essential* to becoming a mathematician, just as a strong chess player must learn to visualize multiple positions in their mind's eye.

But here is the crucial point: when teaching with Lean, the game board is *always there*, automatically generated and continuously updated. Beginners don't have to struggle to reconstruct it from terse prose – they can simply look.

This is why, in my diagram in Figure 7.1, I drew an arrow from Mathlib to Teaching, but also a separate arrow directly from Lean itself. The Mathlib arrow is clear enough: teaching is easier when libraries are large and convenient. But the Lean arrow reflects something different: in teaching, one often does *not* want the most general and abstract definitions as they appear in Mathlib. Just as we do not introduce high school students to Lebesgue integration before they have seen Riemann integration, so too in Lean it is often more effective to write out simpler, more concrete versions of definitions directly. For example, newcomers to Real Analysis should first encounter limits via the familiar ε - δ definition, rather than Mathlib's general `Filter.Tendsto`. In this way, Lean provides a flexible framework for pedagogy, allowing instructors to tailor the level of abstraction to the needs of their students.

This is an area ripe for experimentation. For but one example of very many, Patrick Massot [20] developed a Lean wrapper called **Verbose**, which presents proofs in a controlled natural language, easing students into formal syntax, see Figure 10.1. For my part, I am currently running an experiment in which I teach an undergraduate real analysis course as a kind of *video game*, in the spirit of Buzzard's popular *Natural Number Game*, see Figure 10.2.

The screenshot shows a split-screen interface. On the left, a code editor displays a Lean file named `Examples.lean` with the following content:

```

VerbosoDemo > Examples.lean
114 Example "The squeeze theorem."
115   Given: (u v w : N → R) (l : R)
116   Assume: (hu : u converges to l) (hw : w converges to l)
117     (h : ∀ n, u n ≤ v n)
118     (h' : ∀ n, v n ≤ w n)
119   Conclusion: v converges to l
120 Proof:
121   Let's prove that ∀ ε > 0, ∃ N, ∀ n ≥ N, |v n - l| ≤ ε
122   Fix ε > 0
123   Since u converges to l and ε > 0 we get N such that
124     hN : ∀ n ≥ N, |u n - l| ≤ ε
125   Since w converges to l and ε > 0 we get N' such that
126     hN' : ∀ n ≥ N', |w n - l| ≤ ε
127   Let's prove that max N N' works : ∀ n ≥ max N N', |v n - l| ≤ ε
128   Fix n ≥ max N N'
129   Since ∀ n ≥ N, |u n - l| ≤ ε and n ≥ N we get
130     hNl : |u n - l| ≤ ε
131   Since ∀ n ≥ N', |w n - l| ≤ ε and n ≥ N' we get
132     hN'l : |w n - l| ≤ ε
133   Let's prove that |v n - l| ≤ ε
134   Let's first prove that -ε ≤ v n - l
135   Calc -ε ≤ u n - l since |u n - l| ≤ ε
136   - ≤ v n - l since u n ≤ v n
137   Let's now prove that v n - l ≤ ε
138   Calc v n - l ≤ w n - l since v n ≤ w n
139   - ≤ ε since |w n - l| ≤ ε
140 Qed

```

On the right, a tactic state window shows the current proof state:

- Tactic state**
- 1 goal**
- case intro.intro**
- u v w : N → R**
- l : R**
- hu : u converges to l**
- hw : w converges to l**
- h : ∀ (n : N), u n ≤ v n**
- h' : ∀ (n : N), v n ≤ w n**
- ε : R**
- ε_pos : ε > 0**
- N : N**
- hN : ∀ n ≥ N, |u n - l| ≤ ε**
- N' : N**
- hN' : ∀ n ≥ N', |w n - l| ≤ ε**
- ← ∃ N, ∀ n ≥ N, |v n - l| ≤ ε**

Below the tactic state, there are sections for **Suggestions** and **All Messages (0)**.

Figure 10.1: Massot's Verbose Lean wrapper for teaching Real Analysis in controlled natural language.

These pedagogical experiments reflect a broader historical pattern. Before 1600, mathematicians were doing serious algebra, but it looked like this:

You have some unknown quantity. When you create a second copy of this quantity and combine it with a known amount, then multiply this total by your original unknown quantity, you get a specified result.

Compare to the modern version:

$$x(x + a) = b.$$

The latter is, with some training, so much easier and immediate to understand.

Similarly, before formalization, we rarely named our hypotheses explicitly in proofs. I suspect this may change as formal systems influence mathematical exposition. Just as algebraic notation revolutionized how we think about

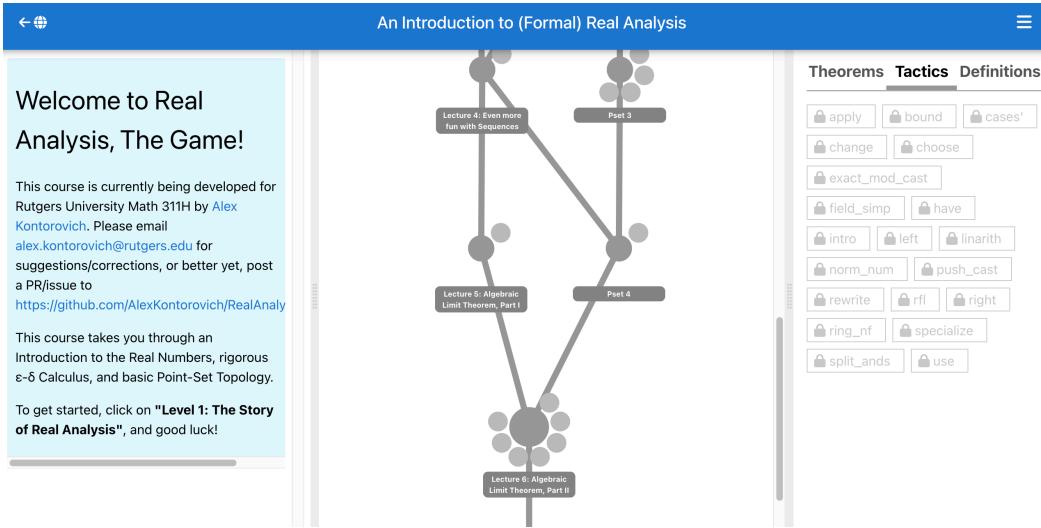


Figure 10.2: The Real Analysis Game, <https://adam.math.hhu.de/#/g/AlexKontorovich/RealAnalysisGame>

equations, the precision required by formal systems may transform how we communicate mathematical arguments, leading us to speak more clearly and precisely even in natural language mathematics. The early signs suggest that Lean is not just a tool for research mathematics: it is poised to transform both mathematical pedagogy and mathematical discourse itself.

11 Understanding and Communication.

The adoption timeline and pedagogical benefits I have outlined focus primarily on efficiency and accessibility. But formalization may also fundamentally transform how mathematicians understand and communicate mathematical ideas themselves – changes that could accelerate adoption for reasons that transcend mere productivity gains.

As Thurston eloquently argued in “On Proof and Progress in Mathematics” [25], a formal proof (even in natural language) represents but the starting point of genuine mathematical understanding, not its culmination. The question is whether formal systems might actually enhance rather than constrain the development of mathematical insight and its communication.

Current large-scale formalization projects offer early glimpses of this potential. Massot’s “Lean blueprint” technology seamlessly transitions between natural language L^AT_EX and corresponding formal Lean code, while generating interactive dependency graphs that visualize the logical structure of complex arguments, see Figure 11.1. These graphs serve an immediate engineering purpose – showing collaborators where “leaves” of the formalized tree require attention – but they also point toward something more profound.

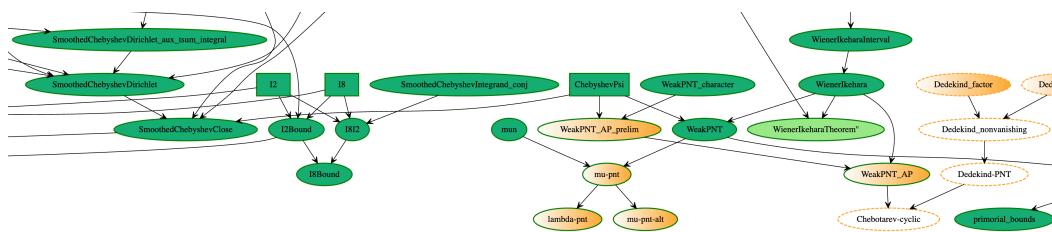


Figure 11.1: A portion of the dependency graph of [16]

Consider how mathematical understanding might evolve when every theorem exists within an explicit, navigable web of logical dependencies. Rather than encountering results as isolated facts in linear exposition, mathematicians could explore arguments at variable levels of detail, zooming from high-level intuition down to

granular formal steps as needed. A student learning algebraic topology could begin with the broad conceptual landscape, then drill down into specific homology calculations only when ready. An expert reviewing a paper could efficiently verify the overall logical structure, examining detailed proofs only for novel or suspicious claims.

This represents a qualitative shift from current mathematical communication, which remains fundamentally linear and static. Journal articles² present arguments in fixed sequences, requiring readers to either accept claims on faith or laboriously verify every step. Formal systems could enable what we might call “proof at multiple resolutions” – mathematical arguments that adapt to the reader’s expertise and interests.

The implications extend beyond individual comprehension to collaborative mathematical practice. When disagreements arise about complex arguments, formal systems provide objective arbitration. When building on others’ work, mathematicians could confidently incorporate results without fear of hidden errors. The cumulative nature of mathematical knowledge – each generation building on previous insights – could accelerate as the reliability of foundations increases.

This could enable a profound shift in mathematical responsibility itself. Currently, mathematical accountability is intensely personal and vertical. In graduate school, I was permitted to quote one result – Deligne’s proof of the Weil conjectures – and expected to know everything else from the ground up. Even now, when I cite a theorem from another paper, I feel personally responsible for understanding its proof. We are like engineers who must mine their own metal, forge their own parts, and assemble everything personally.

But formalization, with its explicit specifications of every assumption and conclusion, could enable genuine modular mathematics. Imagine what theorems we could prove if we could confidently use off-the-shelf components! When formal statements specify precisely what they assume and what they guarantee, mathematicians could build on others’ work with the same confidence engineers have in standardized components. This transition from vertical responsibility to horizontal collaboration could dramatically accelerate mathematical progress.

Yet significant challenges remain. Current formal languages prioritize logical correctness over human intuition, often obscuring the conceptual insights that drive mathematical progress. The risk is that formalization might fragment mathematical culture, creating a divide between those who work in formal systems and those who operate in natural language.

Whether formal systems ultimately enhance or constrain mathematical understanding will likely depend on our success in developing tools that preserve and amplify human mathematical intuition rather than replacing it. The experiments now underway in interactive visualization, natural language interfaces, and collaborative formal environments represent early steps toward what we might call the “Paper of the Future” – mathematical communication that is simultaneously rigorous, accessible, and intellectually inspiring.

The stakes of this transition extend beyond mathematics itself. If formal systems can genuinely enhance mathematical understanding and communication, they may offer a model for rigorous reasoning in other domains where precision and cumulative knowledge matter. The shape of mathematics to come may prefigure broader transformations in how humanity approaches complex reasoning about the world.

12 Conclusion.

The vision I have outlined – of quasi-autoformalization accelerating library growth, formalization factors dropping below unity, and mathematical practice migrating toward formal systems – represents both tremendous opportunity and considerable uncertainty. Whether this transformation unfolds as predicted depends on resolving fundamental tensions between the stochastic nature of current AI systems and the deterministic requirements of mathematical proof.

The evidence points in conflicting directions. On one hand, the trajectory from zero IMO capability in 2023 to gold medal performance in 2025 suggests that AI capabilities in mathematics are advancing with remarkable speed. The success of systems like AlphaProof in closing complex formal goals, even those requiring hundreds of intricate steps, demonstrates that sophisticated mathematical reasoning is within reach of current techniques.

Yet the core challenge remains: even a hypothetical AI system with 99.99% accuracy per step would produce unreliable results when chaining thousands of reasoning steps together. The stochastic nature of large language models – their fundamental reliance on probability distributions over next tokens – appears deeply mismatched to the deterministic correctness that mathematics demands.

This returns us to Tim Gowers’ prescient vision from 2000. Writing at the dawn of the new millennium, he predicted a coming “Golden Age” of human-computer collaboration in mathematics, but warned that such an era

²For more on what the journal refereeing process may look like in the age of formalization, see [14].

might be brief [11]:

The next stage might be one where only a very few outstanding mathematicians could discover proofs that were inaccessible to computers... In the end, the work of the mathematician would be simply to learn how to use theorem-proving machines effectively and to find interesting applications for them. This would be a valuable skill, but it would hardly be pure mathematics as we know it today.

I offer a more optimistic perspective on such a Golden Age, should it indeed arrive. Mathematics has a rich history of problems that seemed intractably difficult but later proved more accessible than expected. The finite field Kakeya conjecture exemplifies this pattern: after years of incremental progress by leading experts, Zeev Dvir solved the full problem in 2008 [6] using elementary techniques from algebraic geometry, revealing that what seemed deeply difficult was actually rather straightforward.

This suggests that AI assistance may help us discover many open problems that are similarly more tractable than they appear, a kind of hidden low-hanging fruit throughout mathematics. The combination of AI's ability to explore vast solution spaces with formal verification's guarantee of correctness could unlock results that have remained elusive not due to fundamental difficulty, but simply because no human had the stamina to try the right combination of techniques from disparate fields.

Of course, what will likely endure after such successes are the genuinely hard problems; and here I fully expect challenges like the Riemann Hypothesis to remain formidable even for sophisticated AI systems. But if we do enter a period where formal, AI-assisted mathematics becomes the norm, the classical role of the “heroic mathematician” may persist in new forms: identifying which problems to pursue, recognizing when seemingly disparate areas might connect, and providing the conceptual insights that guide automated search.

The shape of mathematics to come will ultimately depend on choices we make today about how to develop these tools. If we prioritize pure automation over human insight, we risk creating systems that solve problems without advancing understanding. But if we succeed in building AI that amplifies rather than replaces mathematical intuition – systems that handle the mechanical aspects of formalization while preserving space for human creativity – we may witness not the end of pure mathematics, but its transformation into something more powerful and more beautiful than what came before.

Whether this vision proves prescient or merely optimistic, the mathematical community stands at a remarkable inflection point. The next decade will likely determine whether formal verification becomes as fundamental to mathematical practice as L^AT_EX is today, or remains a specialized tool for the most dedicated practitioners. Either way, we are living through a pivotal moment in the history of mathematical reasoning, and the final chapter of this story remains to be written.

Acknowledgments. My path into Lean began with Kevin Buzzard’s online lectures and encouragement, and it continued only thanks to Heather Macbeth’s patient guidance. Paul Nelson was the first to show me the remarkable potential of large language models, and Christian Szegedy opened my eyes to the central importance of autoformalization. Along the way I have benefited from stimulating conversations with Sanjeev Arora, Jeremy Avigad, Tim Gowers, Thomas Hubert, Ian Jauslin, Chi Jin, JJ Jung, Jeremy Kahn, Patrick Massot, Konstantin Mischaikow, Terry Tao, Akshay Venkatesh, and the vibrant Mathlib community. Above all, I am grateful to Leo de Moura, whose creation of Lean – building on the rich tradition of type theory and proof assistants – has opened formal mathematics to practicing researchers in ways previously unimaginable.

References

- [1] N. ALON, J. BOURGAIN, A. CONNES, M. GROMOV, AND V. MILMAN, eds., *GAFA 2000*, Birkhäuser Verlag, Basel, 2000. Visions in mathematics. Towards 2000, Geom. Funct. Anal. **2000**, Special Volume, Part I.
- [2] K. BUZZARD, *What is the point of computers? A question for pure mathematicians*, in ICM—International Congress of Mathematicians. Vol. 2. Plenary lectures, EMS Press, Berlin, [2023] ©2023, pp. 578–608.
- [3] K. BUZZARD, *Private communication*, 2025.
- [4] H. B. CURRY, *Functionality in combinatory logic*, Proceedings of the National Academy of Sciences, 20 (1934), pp. 584–590, <https://doi.org/10.1073/pnas.20.11.584>.

- [5] N. G. DE BRUIJN, *A survey of the project Automath*, in To H.B. Curry: Essays on Combinatory Logic, Lambda Calculus and Formalism, J. P. Seldin and J. R. Hindley, eds., Academic Press, New York, London, 1980, pp. 579–606.
- [6] Z. DVIR, *On the size of Kakeya sets in finite fields*, Journal of the American Mathematical Society, 22 (2009), pp. 1093–1097.
- [7] M. GANESALINGAM AND W. T. GOWERS, *A fully automatic theorem prover with human-style output*, J. Automat. Reason., 58 (2017), pp. 253–291, <https://doi.org/10.1007/s10817-016-9377-1>.
- [8] D. GOLDFELD, *Gauss’s class number problem for imaginary quadratic fields*, Bull. Amer. Math. Soc. (N.S.), 13 (1985), pp. 23–37, <https://doi.org/10.1090/S0273-0979-1985-15352-2>.
- [9] B. GOMES AND A. KONTOROVICH, *Riemann hypothesis in Lean*, 2020. <https://github.com/bhgomes/lean-riemann-hypothesis>.
- [10] GOOGLE DEEPMIND, *AI achieves silver-medal standard solving International Mathematical Olympiad problems*. <https://deepmind.google/discover/blog/ai-solves-imo-problems-at-silver-medal-level/>, 2024. Accessed: 2025-09-29.
- [11] W. T. GOWERS, *Rough structure and classification*, no. Special Volume, Part I, 2000, pp. 79–117, https://doi.org/10.1007/978-3-0346-0422-2_4. GAFA 2000 (Tel Aviv, 1999).
- [12] W. A. HOWARD, *The formulae-as-types notion of construction*, in To H. B. Curry: Essays on Combinatory Logic, Lambda Calculus, and Formalism, H. Curry, H. B., S. J. Roger, and P. Jonathan, eds., Academic Press, 1980.
- [13] A. Q. JIANG, S. WELLECK, J. P. ZHOU, W. LI, J. LIU, M. JAMNIK, T. LACROIX, Y. WU, AND G. LAMPLE, *Draft, sketch, and prove: Guiding formal theorem provers with informal proofs*, in International Conference on Learning Representations, 2023, <https://arxiv.org/abs/2210.12283>.
- [14] A. KONTOROVICH, *Foreword to: Special issue on interactive theorem provers*, Exp. Math., 31 (2022), pp. 347–348, <https://doi.org/10.1080/10586458.2022.2088982>.
- [15] A. KONTOROVICH, H. MACBETH, AND M. MASDEU, *Moebius action on the upper half-plane*, 2021. <https://raw.githubusercontent.com/leanprover-community/mathlib4/a1957e2a3ecf6e3bf537629fb355e1a023688445/Mathlib/Analysis/Complex/UpperHalfPlane/MoebiusAction.lean>.
- [16] A. KONTOROVICH AND T. TAO, *Prime Number Theorem and More*, Jan. 2024, <https://github.com/AlexKontorovich/PrimeNumberTheoremAnd>.
- [17] Y. LIN, S. TANG, B. LYU, J. WU, H. LIN, K. YANG, J. LI, M. XIA, D. CHEN, S. ARORA, AND C. JIN, *Goedel-prover: A frontier model for open-source automated theorem proving*, 2025, <https://arxiv.org/abs/2502.07640>.
- [18] Y. LIN, S. TANG, B. LYU, Z. YANG, J.-H. CHUNG, H. ZHAO, L. JIANG, Y. GENG, J. GE, J. SUN, J. WU, J. GESI, X. LU, D. ACUNA, K. YANG, H. LIN, Y. CHOI, D. CHEN, S. ARORA, AND C. JIN, *Goedel-prover-v2: Scaling formal theorem proving with scaffolded data synthesis and self-correction*, 2025, <https://arxiv.org/abs/2508.03613>.
- [19] D. LOEFFLER AND M. STOLL, *Formalizing zeta and L-functions in Lean*, Annals of Formalized Mathematics, Volume 1 (2025), <https://doi.org/10.46298/afm.15328>.
- [20] P. MASSOT, *Teaching Mathematics Using Lean and Controlled Natural Language*, in 15th International Conference on Interactive Theorem Proving (ITP 2024), Y. Bertot, T. Kutsia, and M. Norrish, eds., vol. 309 of Leibniz International Proceedings in Informatics (LIPIcs), Dagstuhl, Germany, 2024, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, pp. 27:1–27:19, <https://doi.org/10.4230/LIPIcs.ITP.2024.27>.

- [21] W. RUDIN, *Principles of Mathematical Analysis*, International Series in Pure and Applied Mathematics, McGraw-Hill, New York, 3rd ed., 1976.
- [22] P. SONG, K. YANG, AND A. ANANDKUMAR, *Lean copilot: Large language models as copilots for theorem proving in lean*, in NeurIPS 2024 Workshop on Mathematical Reasoning and AI, 2024, <https://arxiv.org/abs/2404.12534>.
- [23] R. S. SUTTON, *The bitter lesson*. <http://www.incompleteideas.net/IncIdeas/BitterLesson.html>, 2019. Accessed: 2025-09-29.
- [24] C. SZEGEDY, *A promising path towards autoformalization and general artificial intelligence*, in Intelligent Computer Mathematics, vol. 12236 of Lecture Notes in Computer Science, Springer, 2020, pp. 3–20.
- [25] W. P. THURSTON, *On proof and progress in mathematics*, Bulletin of the American Mathematical Society, 30 (1994), pp. 161–177.
- [26] F. WIEDIJK, *The de Bruijn factor*, (2000), <https://www.cs.ru.nl/~freek/factor/factor.pdf>. Technical report, Radboud University Nijmegen.
- [27] Y. ZHANG, *Bounded gaps between primes*, Ann. of Math. (2), 179 (2014), pp. 1121–1174, <https://doi.org/10.4007/annals.2014.179.3.7>.