EMBRACING AI AND FORMALIZATION: EXPERIMENTING WITH TOMORROW'S MATHEMATICAL TOOLS

JAROD ALPER

Abstract. You've heard the buzz: AI and formalization will revolutionize mathematics. Computers will soon surpass humans in solving olympiad-style problems. Humans will transition from proving research-level theorems on their own to guiding computers. And maybe you even believe the hype. Other than pulling out your hair waiting for the day when computers take your job, what can you do? If you are like most career mathematicians, you are already overwhelmed with too many academic responsibilities, reserving any precious spare work hours for research. Where are you going to find the time to learn Lean or machine learning techniques?

While I don't have the answers, I can share how I have embraced the potential of AI and formalization in mathematics through the *eXperimental Lean Lab* (XLL) at the University of Washington. I hope to inspire you to also get involved and play an active role in guiding the transformation of our field.

1. Introduction

> Computers are useless. They can only give you answers.
>
> Pablo Picasso (1968)

I am not an expert in formalized mathematics nor in machine learning. Quite frankly, I generally dislike computers. I do, however, recognize their potential to transform mathematics, and over the past several years I have slowly become proficient in the interactive theorem prover Lean and knowledgeable about the applications of artificial intelligence to mathematics.

Around four years ago I became intrigued about how the Lean programming language can formalize mathematics and its potential to assist in theorem proving. I wanted to get involved, but between my research, a book project, teaching, advising, and administrative duties, where would I find the time? After playing around on my own with Lean, I quickly became disheartened. I found the Lean language extremely finicky, difficult even to install, immensely time-consuming, and frustrating to debug. But it was also exhilarating on the (rare) occasions a proof compiled. I realized that I wasn't going to learn Lean on my own, and tried instead to build a community where we could learn together, commiserate over its challenges, and at the same time educate the next generation with this technology. Since the Spring of 2022, I began running undergraduate research projects as part of the *eXperimental Lean Lab (XLL)*, where students learn Lean and subsequently formalize basic results in undergraduate mathematics.

Date: March 29, 2025.

2020 *Mathematics Subject Classification.* Primary 03F03, 68T01.

Key words and phrases. mathematical formalization, artificial intelligence.

The author was supported in part by NSF Grant 2100088.

The last several years have witnessed a dramatic rise in applications of artificial intelligence (AI) and machine learning (ML) techniques.[1] Large language models such as *ChatGPT* have captured the public's attention and have become an everyday tool for hundreds of millions of people. Reinforced learning algorithms such as AlphaZero, after only being taught the rules, have mastered games like chess and Go overnight, becoming far superior than any human player [SHS+18]. In mathematics, these technologies have been applied to solve olympiad-style problems at the level of an IMO Silver Medal [DM24] and will no doubt surpass us soon. We have only scratched the surface in applications of machine learning to research, but through the generation of massive mathematical data sets, it has already proved effective at generating counterexamples and discovering new relationships. It is an exciting (and scary) time!

We begin with a survey of the state of the art of AI and formalization in mathematics, a risky endeavor as it will surely be outdated by the time it is published. After discussing how to learn Lean and the frustrations that come with it, we explain what we have done in the *eXperimental Lean Lab* at the University of Washington. We end with further suggestions with the hope of inspiring you to also get involved.

Whether or not you think it's in our best interest, changes are coming to our field. Mathematics is the purest form of logical reasoning, and with the rise of proof assistants like Lean providing a check against the hallucinations of large language models, it is perhaps no surprise that mathematics has become a benchmark for the effectivity of AI and that the mathematics profession may see earlier and more drastic changes than other academic disciplines and careers. Computer scientists are already developing these technologies, and it is imperative that mathematicians get involved *now* so that we (rather than computer scientists, corporations, deans, or funding agencies) guide the transformation of our field, design the technologies for mathematical research, and decide on the role that computers will play in our profession.

2. What is Math AI?

> Investing in applied machine learning without understanding the mathematical foundations is like investing in health care without understanding biology.
>
> Rebecca Willett [Wil23]

I break down math AI into five interrelated areas:

- the mathematics behind AI,
- mathematical formalization,
- autoformalization: using machine learning to automate formalization,
- machine learning to assist mathematical research, and
- the meaning of mathematics in the age of AI.

This is not an exhaustive list. It doesn't address for instance "teaching in the age of AI", that is, using AI to accelerate student learning and how to adapt the curriculum to the age of AI. Given that within a year or two, computers will be able to do the homework and exam problems for every undergraduate (and likely graduate) mathematics course, it is essential to rethink homework and evaluation

[1] Technically, AI is a broader term than ML encompassing other approaches to computer intelligence such as symbolic AI, but they are now often used synonymously.

metrics. This is a pressing concern affecting not only mathematics but more broadly education, I have limited the scope to the aforementioned five topics.

2.1. **The mathematics behind AI.** The trillion dollar question is: why are neural networks as effective as they are? While the traditional approach of symbolic AI—which dominated AI research up until the mid-1990s—relied on explicit representations of information using rule-driven logic with explainable results, it is the use of neural networks, based on implicit representations of information using statistical techniques on large data sets with black-box results, that has led to the remarkable applications as of late.

The concept of a neural network is not new and also not complicated. They originated from Rosenblatt's work on the *perceptron* in 1958. Loosely modelled after the human brain, a *neural network* is simply the composition

$$\underbrace{\mathbb{R}^{d_1}}_{\text{input}} \xrightarrow{L_1} \underbrace{\mathbb{R}^{d_2} \xrightarrow{A} \mathbb{R}^{d_2} \xrightarrow{L_2} \mathbb{R}^{d_3} \xrightarrow{A} \cdots \to \mathbb{R}^{d_{n-1}} \xrightarrow{L_{n-1}} \mathbb{R}^{d_n}}_{\text{hidden layers}} \xrightarrow{A} \underbrace{\mathbb{R}^{d_n}}_{\text{output}}$$

of linear functions L_i and a non-linear function A defined coordinate-wise $A(x_1, \ldots, x_n) = (a(x_1), \ldots, a(x_n))$ by an *activation function* $a\colon \mathbb{R} \to \mathbb{R}$, with the ReLU *function* function $a(x) = \max(0, x)$ being a popular choice. By *training* a neural network, we mean choosing the coefficients of the matrices $L_1, \ldots, L_{n-1}$ to optimize the input/output relationship based on *training data* consisting of input/output pairs. This is achieved with what is known as *back propagation*, a sophisticated version of gradient descent performed one hidden layer at a time. For further background, see the video series *3Blue1Brown* [San17] on neural networks for as clear an explanation as you will find.

Depending on the specific purpose, there are various modifications to the basic model of a neural network—convolution, recurrent, adversarial, ... For generating text and images, *transformers* have been widely successful and underlie the success of large language models like ChatGPT. Despite their apparent simplicity, for effective outcomes, their design is more of an art than a science, requiring deliberate decisions about the overall architecture, the number of hidden layers and their dimensions, an effective representation of the data, and the curation of high quality data sets. It is only after decades of trial and error together with the massive scaling of data and computing power that ML researchers have obtained the impressive results we see today.

What are the key principles that make neural networks so effective? For what class of functions are neural networks particularly well suited? What is the tradeoff between the size of the network and the amount of training data needed? How do we avoid the *curse of dimensionality*, the phenomenon when models overfit the data by memorizing the input/output data rather than learning the underlying relationship? How do humans process knowledge, and how can we exploit our advancing understanding of the biology of the brain in the design of models? These questions are attracting more attention. See [PF24] for an overview based on a class of *compositionally sparse* functions, with precise theorems on how effectively they can be learned. See [Mum20], [Kut23], and [BM24] for other surveys.

2.2. **Mathematical formalization.** *Mathematical formalization* is the process of translating mathematical proofs into a formal language that can be checked by a computer. This is accomplished using a *proof assistant*, an interactive program that

facilitates the translation of a proof into a sequence of logical deductions from the axioms. There are many proof assistants—Lean, Agda, Coq, Mizar, HOL Light, Metamath, and Isabelle—each with their own advantages. Given their interactive nature, it's much easier to explain how these assistants work with video demonstrations rather than words, e.g., see [Buz20] and [Mor20] (for popular demos using Lean3).

Early successes include the formalization of the real numbers as a complete ordered field in Automath [Jut77], the prime number theorem in Isabelle [ADGR08], the four color theorem in Coq [Gon08], and the Jordan Curve theorem in Isabelle [Hal07].

Lean has become the dominant proof assistant in the mathematical community. Just like the inevitable rise of one social network over others, this is not because it is fundamentally better, but because for some reason it attracted more users at first. In the case of Lean, its adoption by influential figures such as Kevin Buzzard have contributed to its widespread use today by research mathematicians. The Lean community has built a large database of mathematical definitions and theorems called `Mathlib` covering almost all of undergraduate mathematics [LC24a]. It is rapidly developing, and it already contains some sophisticated objects such as manifolds, schemes, and triangulated categories. Even the definition of a perfectoid space has also been formalized (but it is not yet in `Mathlib`).

In December of 2020, Peter Scholze challenged the formalization community with the *Liquid Tensor Experiment* with the goal of formalizing a difficult foundational theorem in analytic geometry [Sch22]. A team led by Johan Commelin and Adam Topaz completed the challenge within six months. Terrence Tao led a team to formalize one of his recent theorems establishing a variant of the Maclaurin inequality [Tao23]. Out of a (rather arbitrary) list of 100 theorems in mathematics, 79 have been formalized in Lean [LC24b] including the Brouwer fixed point theorem, the Law of Large Numbers, Minkowski's Fundamental Theorem, and Gödel's Incompleteness Theorem (interestingly, 90 out of the 100 have been formalized in Isabelle [Wie24]). There are now large formalization projects with ambitious goals of classifying semisimple Lie algebras, proving Fermat's Last Theorem, building the foundations of ∞-categories, and proving Carleson's theorem using a recently established generalization [BDJ+24]. For deeper surveys, see [Buz24b] and [CT24].

Proof assistants such as Lean have many built-in *tactics*, i.e., high-level commands that work out many of the proof details for you. Tactics such as `exact?`, `apply?`, and `simp` in Lean use pattern matching algorithms to prove or refine a theorem. There are also more sophisticated tactics such as `llmstep` [WS23] and Lean Copilot's `suggest_tactics` and `search_proof` [SYA24] that use large language models to guess the next step of a proof. Proprietary programming tools such as *Github copilot* (powered by OpenAI) and *Cursor* (powered by Anthropic) offer suggestions for code completion and have chat bots within a VS Code environment that can answer coding questions and assist in error debugging. Many of these tools are not yet quite as effective as one would hope, for instance still confusing Lean4 code with the prior version of Lean3. But there is growing interest in such tools and many bright minds developing them, and it is only a matter of time before they become more helpful.

2.3. **Autoformalization.** Autoformalization refers generally to automating the translation of informal mathematics to formal mathematics. Large language models

and reinforced learning have the potential to vastly accelerate formalization, and there is a wide spectrum of possibilities for the division of labor between the human and computer.

The potential of autoformalization has already been evidenced by Numina's winning performance [Num24] in the AIMO progress prize [AIMO24] and Deepmind's performance on the 2024 IMO deserving of a Silver Medal. Deepmind earned a score of 28/42 by solving four out of six problems correctly and missing only the combinatorial problems and a Gold Medal by a single point [DM24]. On the most challenging problem, which was solved by only five high school students, they took three days and nights of computation (something not allowed, or even possible, for human participants!) and, in the end, came up with a clever proof along with its formalization. For the 2025 IMO, the question is no longer if an AI will win a Gold Medal, but by how many teams and by what rules, i.e., limitations on compute, input/output formats, and whether the model is open source.

In the winning solution to the AIMO prize, the Numina team in a collaboration with the company HuggingFace used a sophisticated mix of techniques. For questions ranging from high school to competition-level mathematics, they created a data set of problem/solution pairs, with the solutions written in a *Chain of Thought* format [WWS+22] with *tool-integrated reasoning* abilities [GSG+23]. Chain of Thought refers to breaking the solution into several logical steps written in a combination of natural language and pseudocode, while tool-integrated reasoning refers to integrating natural language with computer code, in this case in the language Python. In fact, OpenAI's latest large language models o1 [OAI24] and soon-to-be-released o3 use similar techniques. The Numina team then used this data set to fine-tune the large language model DeepSeekMath to act as a reasoning agent so that the computer not only learns the solution, but also on the reasoning process and the code needed to determine the answer. Finally, they created several validation data sets to guide the model selection and to avoid overfitting.

On the other hand, Deepmind's Silver Medal performance uses a specialized algorithm AlphaGeometry2 to solve geometry problems, and for other problems, they appear[2] to use a combination of the above strategies together with a reinforced learning algorithm styled after AlphaZero [SHS+18]. AlphaZero is a remarkable algorithm that mastered games such as chess, Go, and shogi through self play with only the knowledge of the rules. Classic algorithms for games used domain-specific strategies (i.e., opening and closing strategies) and a tree search with alpha-beta pruning to eliminate branches unlikely to contribute to a best (or worst) value of a board position and with an evaluation function at the leaves of the tree based on human conceptions of good board positions. Given the large state of possible moves (especially in Go), these tree searches had limited depth. AlphaZero, building on the success of a prior model AlphaGo Zero [SHM+16], uses a probabilistic method to traverse the tree called *Monte Carlo Tree Search (MCTS)* and a deep neural network trained on self play. Rather than simply training the network to assign a *value* to each board state, a number between -1 and 1 where 1 should correspond to a definite win, AlphaZero cleverly assigns a value and a *policy*, a probability distribution for the next move. Through repeated self play using the policy to guide the MCTS search, the computer learns to assign higher probabilities to more effective moves and in the process improves its value assignment. By replacing

[2]Deepmind has not yet released the specific details of their model.

moves with logical steps and board states with proof states, these same techniques apply to proof generation.

These ideas will undoubtedly be applied soon to formalize theorems at the frontier of mathematical research. In some mathematical fields, however, it will take time to build up the foundations first and develop high quality data sets of advanced mathematics to train the AI models.

Unfortunately, at the moment, most of ML research in autoformalization seems narrowly focused on achieving the best results on artificial benchmarks such as miniF2F or working on proof-of-concept results. Their papers may get published in prestigious journals like Nature, but rarely have their algorithms and tools become available to the working mathematician.

Just as the general community should be concerned with the large scale deployment of AI by corporations, the mathematical community should be wary about the long-term intentions of corporations like *Google* and *OpenAI*[3] in developing mathematical reasoning tools. Many of the breakthroughs in autoformalization have come recently from industry, and if we do not act prudently, we could be witnessing the capitalization and corporatization of mathematics, something that our discipline, unlike many others, has so far avoided.

2.4. **Machine learning to assist mathematical research.** When applied to mathematics, modern machine learning algorithms have already proved effective at:

- using large data sets to discover new relationships: computers have predicted the rank of an elliptic curve leading to the discovery of a new relationship with the average value of Frobenius traces over primes, now called *elliptic curve murmurations* [HLOP24] (see [Chi24] for a popular account), predicted Hodge numbers of Calabi–Yau threefolds [He21], discovered relationships between knot invariants [DVB+21], and unravelled relationships between the Kazhdan–Lusztig polynomial and Bruhat graphs leading to a generalization of Lusztig and Dyer's combinatorial invariance conjecture [BBD+22],
- generating counterexamples: conjectures in graph theory have been disproved [Wag21], and
- producing efficient formulas: efficient tensor decompositions have been discovered leading to improvements in matrix multiplication [FBH+22].

In many ways, this is no different than how the computations of Felkel and Vega in 1770s of factorization tables up to 408,000 inspired Legendre and Gauss to conjecture what is now known as the prime number theorem. Or how Birch and Swinnerton-Dyer's use of a primitive computer in the 1960s to count the number of solutions to elliptic curves over finite fields led to their famous conjecture. The machine learning of today offers a supercharged version of this. See [Hal14], [Buz24b], [Dav24], and [Wil24] for surveys.

2.5. **The meaning of mathematics in the age of AI.** What do we do when computers outperform even the brightest high school students in the Olympiad? What about when they become better at proving theorems? What if they discover a five page elementary proof of Fermat's Last Theorem? What if they become better at generating conjectures and synthesizing mathematics across very different fields?

[3]Shouldn't they change their name to *ClosedAI* already?

We can draw some lessons from the chess community, which faced a similar reckoning with machines after IBM Deep Blue's defeat of Garry Kasparov in 1997. Chess is now more popular than ever. Players have benefited from studying the strategies of AIs, and the community has emphasized the human element of chess. But mathematics is also very different from chess. It's not a game; it's our profession. Perhaps competition math may not change much and, who knows, may even get more popular (but I doubt millions will be regularly streaming videos of students taking math exams), but research mathematics will fundamentally change.

My view is that mathematics too is a human endeavor. AI will change the way we do research, the way we write, and even the way we think, but it will be up to us to determine what mathematical statements we value and to develop a human understanding. It is imperative that the mathematical community think about these questions now and that we deliberately adapt our profession to the emerging technology. See [Ven24] and [Och24] for a broader discussion.

There are many ethical issues and risks that arise with the use of AI in mathematics. First, there is the enormous cost—both monetary and environmentally—in the deployment of large-scale models. Second, there are risks that we are already confronting: infringement of intellectual property rights and the potential for bias. Looking forward, we should be very concerned of the dangers of AI in surveillance and control by the government, military, or corporations (or even powerful individuals). Lastly, there is existential risk: mathematics is one of the most complex and intellectually challenging human endeavors, and if we are actually able to succeed at the challenge of training computers, are we not fairly close to superhuman intelligence? Toby Orb, in his book *The Precipice* [Ord20], calculates that humanity's overall existential risk over the next hundred years—meaning either extinction or an unrecoverable collapse—is 17% (one out of six) and that the existential risk of unaligned AI is 10%. You can be be skeptical of the numbers, but it would be foolish to ignore the risk.

3. Evolution of mathematics

> Rigor has ceased to be thought of as a cumbersome style of formal dress that one has to wear on state occasions and discards with a sigh of relief as soon as one comes home. We do not ask any more whether a theorem has been rigorously proved but whether it has been proved.
>
> André Weil [Wei56, p. 550]

From Euclid's postulates of geometry in 300 BC to Hilbert's axiomization of geometry in 1899 to the introduction of Zermelo–Fraenkel set theory in the early 1900s, the mathematical process has evolved through time. In the 20th century, the abstraction of mathematical concepts and the introduction of new notation changed the way people wrote and thought about mathematics.

The Bourbaki School, a collection of mathematicians initiated in 1930s by Claude Chevalley and André Weil, produced volumes of mathematics that were at the time the highest standards of rigor. A formalized proof in Lean (or another proof assistant) is the gold standard of today but we would be naive to think that it will not evolve further. For interesting surveys on the evolution of proof, see [Gra24],[DT24], and [McL24].

To see this progression in an example, let us consider *Hilbert's Basis Theorem*, one of my favorite theorems in mathematics and, in fact, the first result our *eXperimental Lean Lab* tried to formalize.

> David Hilbert (1890): *Ist irgend eine nicht abbrechende Reihe von Formen der n Veränderlichen $x_1, x_2, \ldots, x_n$ gegeben, etwa F_1, F_2, F_3, ..., so giebt es stets eine Zahl m von der Art, dass eine jede Form jener Reihe in die Gestalt*
>
> $$F = A_1F_1 + A_2F_2 + \cdots + A_mF_m$$
>
> *bringen lässt, wo $A_1, A_2, \ldots, A_m$ geeignete Formen der nämlichen n Veränderlichen sind* [Hil90, Thm. I, p. 174].
>
> Translation: *If any non-terminating sequence of forms of the n variables $x_1, x_2, \ldots, x_n$ is given, for instance $F_1, F_2, F_3, \ldots$, then there always exists a number m of such a kind that every form of that sequence can be written as*
>
> $$F = A_1F_1 + A_2F_2 + \cdots + A_mF_m,$$
>
> *where $A_1, A_2, \ldots, A_m$ are suitable forms of the same n variables.*

A *form in n variables* is what we would call today a homogeneous polynomial in n variables with complex coefficients. Hilbert established his basis theorem in order to prove that the ring of algebraic functions defined on the space of homogeneous polynomials in n variables of degree d that are invariant under coordinate change—which we might write today as $\left(\mathrm{Sym}^*(\mathrm{Sym}^d \mathbb{C}^n)^\vee\right)^{\mathrm{SL}_n}$—is finitely generated for all d and n. Determining these rings was of upmost importance in the 19th century, and Hilbert's proof was not constructive, shocking the mathematical community. The self-proclaimed 'King of Invariant Theory' Paul Gordan declared: "Das ist nicht Mathematik, das ist Theologie." It was only after Hilbert offered a constructive proof three years later—and in the process established the Syzygy Theorem, the Nullstellensatz, a version of Noether normalization, and a version of the Hilbert–Mumford criterion—that Gordan retorted: "Ich habe mich überzeugt, dass auch die Theologie ihre Vorzüge hat." (Translation: I've been convinced that even theology has its merits.)

Inspired by ideas of David Hilbert and Évariste Galois, Emmy Noether led the abstraction of algebra in the early 19th century with the development of the theory of rings and fields. Noether studied a class of rings—now called *noetherian rings*—with the property that every ideal is finitely generated. This led to a formulation of Hilbert's Basis Theorem as presented today in an undergraduate course.

> *Pour tout anneau commutatif nœthérien C, l'anneau de polynômes $C[x]$ is nœthérien* [Bou61, III §2.10 Cor. 1].
>
> Translation: *For every commutative noetherian ring C, the ring of polynomials $C[x]$ is noetherian.*

I have enjoyed dabbling in the classical invariant theory literature of the 19th century, but it has been difficult at times to follow their conventions. However, I do believe that if I were better versed in the notations and standards of the time, I would find it preferable to the writing in the decades following the Bourbaki School. We all know of some textbooks, that despite their success in synthesizing a large body of knowledge, are an absolute pain to read. It sometimes seems that those books were not written with a human reader in mind, but rather to train computers:

while reading a proof on page 423, who else is going to remember that symbol that was introduced once on page 58 and never recalled.

This is how Hilbert's Basis Theorem is currently stated in Lean's `Mathlib`:

```
protected theorem Polynomial.isNoetherianRing[4] [inst :
IsNoetherianRing R] : IsNoetherianRing R[X]
```
[Lau19].

Given that Hilbert's original formulation is arguably the clearest in conveying the mathematical meaning, are we even progressing?

4. Why formalize?

> A word of warning — and apology. There are several thousand formulas in this paper which allow one or more 'sign-like ambiguities': i.e., alternate and symmetric but non-equivalent reformulations. These occur in definitions and theorems. I have made a superhuman effort to achieve consistency and even to make correct statements: but I still cannot guarantee the result.
>
> David Mumford [Mum66, p. 288]

The most obvious reason to formalize a proof is to provide a verifiable check on its correctness. We no longer need to endlessly question whether we verified every last detail, giving us the confidence that twenty years from now a mathematician will not point out a crucial error.[5] We can also avoid mathematical disputes that arise when human egos come into play. In some areas of mathematics such as homotopy type theory involving sophisticated objects and subtle reasoning, formalization has already served as an effective guide making sure errors aren't introduced along the way [Shu24].

I work in a technical subfield of algebraic geometry—algebraic stacks and moduli spaces—where most papers (including my own) inevitably have errors, most being rather minor and patchable with slight tweaks. There are sometimes serious errors, but ultimately humans are wise and they are eventually located and corrected. Other than peace of mind and not having to worry about the all-too-common pesky sign error, the biggest advantages I see in formalizing mathematics are in improving our understanding, training computers, and in mathematical exposition.

4.1. **Improving understanding.**

In my experience with students, I have observed that formalizing a statement often exposes a gap in their understanding. This gap is necessarily filled during the process, preventing the gap from widening into a chasm. From a pedagogical perspective, proof assistants let you visually walk through the steps of an argument, explaining exactly where hypotheses are used.

At the research level, formalization has also led to a better understanding. In the formalization of the Kepler conjecture, in addition to many small errors being addressed, a serious error was located [HHM+10]. In correcting it, Hales realized he could prove other conjectures such as the strong dodecahedral theorem [Hal12].

[4] Did you notice the difference between `isNoetherianRing` and `IsNoetherianRing`? The former is the name of the theorem, while the latter is the property of a ring being noetherian. A better name would of course be `hilberts_basis_theorem` but `Mathlib` has a convention to avoid proper names and thereby conflicts with how mathematicians universally refer to them. Other conventions lead to the *five lemma* in homological algebra being named `is_iso_of_is_iso_of_is_iso_of_is_iso_of_is_iso` rather than `five_lemma`.

[5] There is a limit of course to this degree of confidence: why should you trust the Lean kernel? And how can you be certain that the statement of the theorem in Lean is equivalent to the statement in a paper or book? See [Pol98] for an interesting perspective on these questions.

Similarly, in the Liquid Tensor Experiment, Commelin and Scholze realized that the proof could be substantially simplified by using MacLane's Q'-construction (which they rediscovered in the process). Scholze wrote that "during the formalization, a significant amount of complex geometry had to be formalized... and this made me realize that actually the key thing happening is a reduction from a non-convex problem over the reals to a convex problem over the integers" [Sch21].

4.2. **Training computers.** Large language models have had remarkable success at generating human-like arguments, but as we all know now, they can fabricate details in sometimes very convincing ways. They may become more reliable over time, but given the statistical nature of these algorithms, will we ever be able to trust them? Formalization provides a check: by requiring the computers to give a formalized proof, we can be certain of its correctness. Moreover, this gauge for correctness can be leveraged in reinforcement learning algorithms to train computers at generating proofs in a similar style to AlphaZero. For partly this reason, mathematical problem solving is now one of the hottest topics in the artificial intelligence community. Where there used to be only a handful of researchers in the special sessions in the autoformalization session of the ML conferences NeurIPS and ICLR, there are now hundreds, with thousands of others (including more and more mathematicians) paying attention.

4.3. **Mathematical exposition.** Let's be honest: most writing in mathematics is terrible. Part of the problem is that there is no (or at least very little) training for our PhD students in writing. A larger issue is that there is little incentive for good exposition of research articles. Writing well takes time, and there is far more reward—in terms of how committees evaluate grants, job applicants, and tenure cases—in writing more papers. Bad writing also leads to a form of gatekeeping, where if a paper is technically correct but the key ideas are completely inaccessible, it is only the authors that can continue that line of investigation.

Formalization offers a potentially better paradigm. After a theorem is formalized, the authors can write a lengthy introduction with a high-level exposition of the proof emphasizing the key new insights. The formalized code can be submitted with the paper, and after the referees make sure that the code proves what the paper claims it does (something more subtle than it may seem), they can focus their attention instead on improving the exposition rather than checking countless details.

Another possibility is *automated informalization*: the process of generating human-readable expositions, possibly at various levels of detail, from formal proofs. See [MM24] for a project in this direction.

4.4. **Software verification.** One of the original motivations for Lean was to verify the correctness of software, an imperative need in applications handling critical infrastructure such as financial transactions or military hardware. Software verification is a degree more complicated than theorem proving as a necessary first step is to provide a specification, i.e., a translation of the program into a mathematical framework, which may or may not perfectly reflect real world uses. Proof assistants can then be applied to formalize the theorem. There is now a formally verified OS kernel called seL4 [KEH+09] and C compiler CompCert [BDL06], the latter in use by Airbus France. There are a growing number of applications in industry: the verified cryptographic routines of EverCrypt are now used in Mozilla Firefox, the Linux

kernel, and the verified electronic voting software of ElectionGuard [PPF+20]. Theorems in Lean's `Mathlib` have been used by Amazon Web Services to verify code [Mou24]. The verification of software has led to the discovery of errors just as in mathematics, and has had the added benefit of allowing programmers to tinker with their code to drastically improve performance, with the confidence they aren't introducing bugs in the process.

4.5. **Further reasons.** Formalization also offers the potential for mass collaboration on mathematical projects, where individuals with various levels of expertise can participate in a project. As long as the computer accepts the code, there is no need for every author to check everyone else's contributions (although arguably it is still important to have well-written and documented code). Patrick Massot's Lean BluePrint [Mas24] offers a tool to facilitate this type of collaboration. As a trial run for this new form of collaboration, Terence Tao recently initiated the *Equational Theories Project* [Tao24]. See [Mas21], [Avi24], and [Drö24] for further motiviation.

5. Learning how to use Lean

> We often hear that mathematics consists mainly of proving theorems. Is a writer's job mainly that of writing sentences?
>
> Gian-Carlo Rota [DH80, Foreward]

The degree of success of the Lean community in formalizing theorems is somewhat surprising given how difficult Lean is to learn and use. At this point, in addition to solid mathematical foundations, there are two other necessary characteristics to become an expert in Lean: (1) a programming mindset of the sort where you get deep joy from resolving an incomprehensible error message and (2) endless amounts of time. On the other hand, there are several excellent introductory sources [LC24c] making it possible, even with limited patience and time, to become proficient in Lean.

5.1. **Learning Lean.** If you want a taste of Lean, I recommend starting with the Natural Number Game [BP24]: no setup is required and you are guided through formalizing the basic properties of the natural numbers, but I wouldn't feel pressured to finish the game. In our *eXperimental Lean Lab*, our primary source has been Jeremy Avigad and Patrick Massot's *Mathematics in Lean* [AM22] with Kevin Buzzard's *Formalization Mathematics* [Buz24a] as a secondary source.

It is possible to start playing around with Lean online using the Lean Playground [LC24d], Gitpod, or Github Codespaces. This may save you some time and frustration at first, but if you want to get serious, I strongly recommend installing a VS Code environment on your computer with the Lean extension. Lean4 has become easier to install and if you follow the instructions carefully from [Lea24], it will probably work for you.

Unfortunately, it is still necessary to understand a bit about dependent type theory, which is the underlying mathematical foundations on which Lean is built, along with what is happening underneath the hood of Lean engine. Personally, I've always been more interested in getting on the Lean highway and driving as fast as I can rather than understanding the engine, which probably explains why I crash so often. The book [AMK17] is a good source. However you approach it, learning

Lean takes time; just like any advanced mathematical topic, you should expect to gradually learn it over months and years, not overnight.

5.2. **Beyond tutorials.** One challenge I've found, both for myself and for our students, is the gap between completing exercises in a tutorial, where things are setup to work with the few skills you've learned, and doing a project on your own, especially if it involves even slightly sophisticated concepts like rings and ideals. In some ways, this is like the leap in graduate school from course-oriented mathematics to research.

Unfortunately, there are not many intermediate references to help bridge this gap, such as specialized courses on algebra, topology, or analysis in Lean. The code in `Mathlib` is generally inaccessible, extremely sparsely documented, and *code golfed*[6] to obscurity. However, there is a very welcoming and helpful community on the *Lean Zulip Chat* [LC24g], where you will get answers to most questions—from the most elementary to advanced—within a few hours, if not minutes. With over over 10,000 users, posting a question on the Zulip server may feel as intimidating as posting on *MathOverflow*, but it is a great way to quickly debug an error rather than wasting hours struggling on your own.

6. Mathematicians deserve a more user-friendly Lean

> For Bourbaki, Poincaré was the devil incarnate. For students of chaos and fractals, Poincaré is of course God on Earth.
>
> attributed to Marshall Stone

We need to lower the barrier for students and working mathematicians to learn and use Lean. AI-assisted tools will surely help in the near future, but for AI models to be more effective, they need (at least in my view) more high quality *human-generated* training data. Making Lean more user-friendly would accelerate this cyclical process.

6.1. **Why is software for mathematicians so bad?** There is growing awareness that software and programming languages should be designed to scaffold a user into the complexity. Users should be shielded at first from more complicated tools and provided understandable error messages. From this perspective, our current software options ranging from *LaTeX* to the graphical tools of *Tikz*, *Gimp*, and *Inkscape* to the version control software *git* all fail miserably. There are some exceptions of course—I've found computational software such as *Mathematica*, *SAGE*, and *Macaulay2* easy to learn and use.

It boggles my mind that there is still not a better option than *LaTeX*, a clunky piece of software that was first released in 1984 and never improved. Recently, as I have been trying to finish a book, I spend a good amount of my time writing, and I estimate that I waste on average 30 minutes per week on stupid *LaTeX* issues. I know, I know: *LaTeX* usually tells you where the error is, but all too often there is a unicode, whitespace, or bibtex error, or a package inconsistency that confounds me, and the time does add up. Mathematicians are resourceful and we have managed ways to cope with it, but we do deserve better. We also don't train our graduate students well leaving each individual to learn (or not learn) these skills on their own. We had a graduating PhD student in our department, where it was only after

[6]Code golfing refers to writing code in the most concise way possible.

he complained to fellow students about how time consuming it was to manually update the numbering in his thesis after each minor change that he was informed about *labels*.

And don't get me started on *git*, the version control software that was named (and seemingly designed) after a British slang meaning an 'unpleasant person'. As a non-power user, I only ever want to do one of two things: *download the most recent version* or *upload my revisions*. The graphical interfaces of *Github Desktop* or *Sourcetree*, however, present me with a myriad of options with the mystifying terminology of pull, push, head, origin, master, staging, hunks, ... But my biggest frustration is that it was recently made so secure that I couldn't authenticate for years. Every few months I would try again with renewed courage thinking 'this can't be *that* difficult', but I would only further mess up my ssh keys, which also prevented others from easily helping. Giving up, I would ask my collaborators to place a copy of the files in dropbox (which just works, by the way). That's enough of the rant, sorry.

6.2. **A user-friendly Lean.** In a dream world, installing Lean should take one click and starting a new Lean project should be as simple as creating a folder with a `.lean` file (rather than typing six commands into the terminal). There should be much, much better error debugging and a syntax linter to prevent errors. You should never get a message that '`Imports are out of date and must be rebuilt`' and then have to wait overnight for `Mathlib` to rebuild.[7] Lean and `Mathlib` need to become more stable; I have heard of some cases where simply restarting the computer resulted in Lean accepting the proof. Lean's *Infoview*, the window displaying the current state of the proof with all of the variables, hypotheses, and the goal, could be better organized highlighting the most relevant information. (*Paperproof* appears to be a promising step in this direction.)

Some statements that are completely trivial for mathematicians require a surprising amount of effort to formalize. Mathematical objects generally have many realizations, and the type system of Lean allows objects to extend others. The type of a commutative ring `CommRing` extends `Ring`, which extends `Semiring`, which extends `NonUnitalSemiRing`, which extends `NonUnitalNonAssocSemiring`, which extends `AddCommMonoid`, which extends `AddMonoid`, which extends `AddSemigroup`, which extends the type `Add`, a homogeneous version of `HAdd`. Since some statements for commutative rings hold for these more general types and since `Mathlib` attempts to be as general as possible, it is often difficult to locate where a particular lemma is stated.

Switching between types—a process known as *casting* or *coercion*—and resolving '`type mismatch`' errors are some of the biggest challenges in using Lean. For instance, suppose we have a group G_3 and subgroups $G_2 \subseteq G_3$ and $G_1 \subseteq G_2$, and you want to view $G_1 \subseteq G_3$ as a subgroup. A mathematician wouldn't think twice of this, but this requires a non-trivial amount of work in Lean, such as defining the canonical inclusions $i_1 \colon G_1 \to G_2$ and $i_2 \colon G_2 \to G_3$, defining the composition $i_2 \circ i_1$, and taking the image of $i_2 \circ i_1$, with similar chicanery needed to move elements around. In some sense, this is not a surprise: after all, we are building a proof from axioms and can't skip any detail. It is helpful to learn the `Mathlib` naming conventions [LC24f], and the natural language searching tools *Moogle*, *LeanSearch*,

[7] Yes, I ran `lake exe cache get, lake build` in the terminal, but should we really have to?

and *Loogle* for `Mathlib` are also great resources for looking up results, but it can still be a challenge. But as developers build more tools and tactics, I am hopeful many of these issues will go away.

Lean is a new and complex programming environment that is constantly evolving. Developing useful tools is challenging and labor intensive, and making them easy to install and user-friendly is another level of complexity. Just like other mathematical software, there is unfortunately little incentive in academia to develop these tools. I am amazed by what the Lean community has already accomplished, and while there has been a lot of focus on building tools for power-users, they also recognize the need to make Lean more accessible to beginners. The organization Lean FRO is taking admirable steps in these directions, but they could be more effective with more funding and more developers.

7. The eXperimental Lean Lab

> Science is what we understand well enough to explain to a computer, Art is all the rest.
>
> Donald E. Knuth [PWZ96, Foreward]

Over the last three years at the University of Washington, we have run over twenty formalization projects through the *eXperimental Lean Lab (XLL)*. I have been the faculty leader but have received terrific assistance and expertise from the graduate student mentors Herman Chau, Vasily Ilin, and Leopold Mayer. After trial-and-error, we have gradually learned how to design more effective projects.

7.1. Program structure.

The organization of our lab benefited from the existing structure of the *Washington eXperimental Mathematics Lab (WXML)* at the University of Washington, a program initiated by Jayadev Athreya allowing faculty to run research projects with undergraduates. We have several projects each quarter, but we always meet as a larger group. This allows everyone to learn Lean together in the beginning and also help troubleshoot errors later on. Students earn some course credit, and there are often students participating for multiple quarters, allowing them to take on more advanced projects as well as mentor other students. The program does not require much funding (especially in view of its merits): small stipends for graduate student mentors and some money for food and refreshments at the opening meeting and final presentation.

7.2. Group projects.

Individual projects can succeed for a motivated student, but we found that generally groups of 2-4 students work best. It's more enjoyable for the students, and they can learn and troubleshoot together. It can lead to the situation where one student does most of the coding, but as long as the others are learning in the process, that's fine with me.

7.3. Formalization projects.

There are really countless options when it comes to projects. We have had students formalize everything from Hilbert's Basis Theorem to Fibonacci identities to the continued fraction expansion of e to characterizations of ring properties in terms of their prime ideals [SP, Tag 05K7]. The first lesson

we learned is that students should already have very good command of the mathematics. Learning new mathematics and formalizing are each both difficult on their own, and combining them can be too daunting.[8]

In the beginning, we tried to give students the choice to design their own project, but this generally did not work as well as we hoped. Without experience in formalization and less background in mathematics, it is really difficult to know what is feasible. With ten week quarters, there was also not enough time to discuss and organize projects, especially considering how time intensive it is to learn Lean. In my view, the more successful projects arose when mentors had specific goals in mind and a broad outline of their formalization, while leaving room for the creativity of the students.

The topic should be tailored to the students' background and interest. For beginning undergraduates or even high school students, formalizing olympiad-style problems is a good option, with the potential added benefit of contributing to an online database. Another good option is to formalize examples and counterexamples in a subject, starting with simple examples and then allowing the students to have ownership of the project by choosing further examples themselves. (If we are training computer to think like us, surely the AI models would also benefit from examples.) One approach we thought would work well, but didn't, was formalizing exercises in a favorite undergraduate textbook: it turns out that most exercises do not lend themselves well to formalization.

7.4. **Future plans.** At the University of Washington, we are broadening the *eXperimental Lean Lab* into a *Math AI Lab*, with projects in each of the five areas of Math AI discussed in Section 2.

8. How to get involved

> The product of mathematics is clarity and understanding. Not theorems, by themselves. ... In short, mathematics only exists in a living community of mathematicians that spreads understanding and breathes life into ideas both old and new.
>
> Bill Thurston [Thu10]

There are many ways to get involved with these emerging technologies. Even if you have an aversion to computers, programming, and more screen time, there is a still a role for you in terms of mathematical guidance as well as project design and management.

8.1. **Pay attention.** Begin by reading up on the latest developments. Computer-assisted mathematics was the general theme in each of the articles in the April and July issues of 2024 *Bulletin of the AMS*, many of which were referenced above. If you want a front seat, you can join the *Lean Zulip* server [LC24g], where most developments are announced first. On the Zulip server, you can also follow along on the status of community-driven formalization projects, see their Lean blueprints, and, if inspired, claim projects of your own.

[8]This viewpoint is only in the context of a 10-week project. In my view, we *should* gradually integrate Lean in the undergraduate curriculum such as in Heather Macbeth's course *The Mechanics of Proof* [Mac24], which is an introduction to proof through the lens of Lean.

8.2. **Discuss with colleagues.** Discuss these topics with colleagues in your math department or at conferences. I have been surprised recently how often discussions turn to Math AI (and I am not always the only one bringing it up) and how beneficial it can be to hear other perspectives.

8.3. **Build a local community.** Math AI is a truly interdisciplinary topic merging topics in pure mathematics, applied mathematics, statistics, computer science, and philosophy. Universities always preach about interdepartmental collaboration, but it's easier said than done, and there's often little infrastructure to facilitate such collaboration. It is important to be very intentional about fostering an environment where you can actually meet and engage with your colleagues in other departments.

You can create a Math AI seminar, advertised in each of these departments, with speakers—both local and external—covering a broad range of topics. Speakers can be encouraged to not just present their latest research, but offer expository and accessible talks covering a latest development, e.g., an introduction to AlphaZero's reinforcement learning algorithm. When paired with a social event like a tea or happy hour, these seminars can spark discussions and potential collaboration.

8.4. **Attend conferences and workshops.** There is a growing number of workshops and conferences addressing AI and formalization. Conferences such as *Lean for the Working Mathematician*, *Lean Together*, or more specialized events (e.g., the 2023 Banff workshop *Formalization of Cohomology Theories* or the 2024 AIM workshop *Formalising Algebraic Geometry*) can offer exposure to the Lean community. See [LC24h] for a list of upcoming conferences. Some workshops create working groups dedicated to the formalization of a specific topic, and joining one of these groups can provide a great way to bridge the tutorial-to-project gap in learning Lean.

8.5. **Teach a class.** The best way to learn a subject is to teach it. For formalization, there are some good teaching materials [LC24c]. For general machine learning, there are many resources, e.g., [Tra19], [Pri23], and your computer science department likely has course offerings, but I am not aware of any specialized resources geared toward applications in mathematics. I am also unaware of teaching resources for the mathematics of AI, but [LSM+24] provides a detailed survey of the existing mathematical literature. See also the working document [NAC24] produced after the 2023 National Academies workshop on *AI to Assist Mathematical Reasoning*.

8.6. **Run an undergraduate research project.** You can try to do something similar to our *eXperimental Lean Lab*, advise a senior thesis, or run an REU program.

8.7. **Brainstorm.** Brainstorm mathematical questions that you might think are amenable to machine learning techniques, discuss them with your colleagues, and possibly solicit assistant from the ML community.

9. Conclusion

The adoption of AI and formalization in research mathematics will not be a passing fad. In the coming years and decades, we will witness a growing use of computers to assist in theorem proving and likewise a growing use of mathematics to understand AI. The time to get involved is now.

Acknowledgements. I would like to thank Dhruv Bhatia, Herman Chau, Andy Heald, Vasily Ilin, and Leopold Mayer for their assistance in the *eXperimental Lean Lab* in organizing and mentoring projects. I also I benefited from conversations and feedback from Mariana Alper, Fabrizio Barroero, Johan Commelin, Johan de Jong, Albert Jiang, Lenny Taelman, Amos Turchet, Ravi Vakil, and Sean Welleck.

References

[ADGR08] Jeremy Avigad, Kevin Donnelly, David Gray, and Paul Raff, *A formally verified proof of the prime number theorem*, ACM Trans. Comput. Log. **9** (2008), no. 1, Art. 2, 23.

[AIMO24] AIMO Progress Prize Committee, *AIMO progress prize: July 2024 results*, 2024, https://aimoprize.com/updates/2024-07-20-progress-prize-results.

[AM22] Jeremy Avigad and Patrick Massot, *Mathematics in Lean*, 2022, https://leanprover-community.github.io/mathematics_in_lean/.

[AMK17] Jeremy Avigad, Leonardo de Moura, and Soonho Kong, *Theorem proving in Lean*, 2017, https://leanprover.github.io/theorem_proving_in_lean/.

[Avi24] Jeremy Avigad, *Mathematics and the formal turn*, Bull. Amer. Math. Soc. (N.S.) **61** (2024), no. 2, 225–240.

[BBD+22] Charles Blundell, Lars Buesing, Alex Davies, Petar Veličković, and Geordie Williamson, *Towards combinatorial invariance for Kazhdan-Lusztig polynomials*, Represent. Theory **26** (2022), 1145–1191.

[BDJ+24] Lars Becker, Floris van Doorn, Asgar Jamneshan, Rajula Srivastava, and Christoph Thiele, *Carleson operators on doubling metric measure spaces*, 2024, arXiv:2405.06423.

[BDL06] Sandrine Blazy, Zaynah Dargaye, and Xavier Leroy, *Formal verification of a C compiler front-end*, FM 2006: Formal Methods (Berlin, Heidelberg) (Jayadev Misra, Tobias Nipkow, and Emil Sekerinski, eds.), Springer Berlin Heidelberg, 2006, pp. 460–475.

[BM24] Yoshua Bengio and Nikolay Malkin, *Machine learning and information theory concepts towards an AI mathematician*, Bull. Amer. Math. Soc. (N.S.) **61** (2024), no. 3, 457–469.

[Bou61] Nicolas Bourbaki, *Éléments de mathématique. Fascicule XXVIII. Algèbre commutative. Chapitre 3: Graduations, filtrations et topologies. Chapitre 4: Idéaux premiers associés et décomposition primaire*, Actualités Scientifiques et Industrielles [Current Scientific and Industrial Topics], vol. No. 1293, Hermann, Paris, 1961.

[BP24] Kevin Buzzard and Mohammad Pedramfar, *The natural number game*, 2024, https://adam.math.hhu.de/.

[Buz20] Kevin Buzzard, *10 minute Lean tutorial : proving logical propositions*, 2020, https://www.youtube.com/watch?v=P0HVMMG7pqE.

[Buz24a] ———, *Formalising mathematics*, 2024, https://github.com/ImperialCollegeLondon/formalising-mathematics-2024.

[Buz24b] Kevin Buzzard, *Mathematical reasoning and the computer*, Bull. Amer. Math. Soc. (N.S.) **61** (2024), no. 2, 211–224.

[Chi24] Lyndie Chiou, *Elliptic curve 'murmurations' found with AI take flight*, Quanta Magazine (2024), https://www.quantamagazine.org/elliptic-curve-murmurations-found-with-ai-take-flight-20240305/.

[CT24] Johan Commelin and Adam Topaz, *Abstraction boundaries and spec driven development in pure mathematics*, Bull. Amer. Math. Soc. (N.S.) **61** (2024), no. 2, 241–255.

[Dav24] Alex Davies, *Working with machines in mathematics*, Bull. Amer. Math. Soc. (N.S.) **61** (2024), no. 3, 387–394.

[DH80] Philip J. Davis and Reuben Hersh, *The mathematical experience*, Birkhäuser, Boston, MA, 1980, With an introduction by Gian-Carlo Rota.

[DM24] DeepMind, *AI achieves silver-medal standard solving International Mathematical Olympiad problems*, 2024, https://deepmind.google/discover/blog/ai-solves-imo-problems-at-silver-medal-level/.

[Drö24] Christoph Drösser, *AI will become mathematicians 'co-pilot'*, Scientific American (2024), https://www.scientificamerican.com/article/ai-will-become-mathematicians-co-pilot/.

[DT24] Silvia De Toffoli, *Proofs for a price: tomorrow's ultra-rigorous mathematical culture*, Bull. Amer. Math. Soc. (N.S.) **61** (2024), no. 3, 395–410.

[DVB+21] Alex Davies, Petar Veličković, Lars Buesing, Sam Blackwell, Daniel Zheng, Nenad Tomašev, Richard Tanburn, Peter Battaglia, Charles Blundell, András Juhász, Marc Lackenby, Geordie Williamson, Demis Hassabis, and Pushmeet Kohli, *Advancing mathematics by guiding human intuition with AI*, Nature **600** (2021), no. 7887, 70—74.

[FBH+22] Alhussein Fawzi, Matej Balog, Aja Huang, Thomas Hubert, Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Francisco J. R. Ruiz, Julian Schrittwieser, Grzegorz Swirszcz, David Silver, Demis Hassabis, and Pushmeet Kohli, *Discovering faster matrix multiplication algorithms with reinforcement learning*, Nature **610** (2022), no. 7930, 47–53.

[Gon08] Georges Gonthier, *Formal proof—the four-color theorem*, Notices Amer. Math. Soc. **55** (2008), no. 11, 1382–1393.

[Gra24] Andrew Granville, *Proof in the time of machines*, Bull. Amer. Math. Soc. (N.S.) **61** (2024), no. 2, 317–329.

[GSG+23] Zhibin Gou, Zhihong Shao, Yeyun Gong, Yelong Shen, Yujiu Yang, Minlie Huang, Nan Duan, and Weizhu Chen, *Tora: A tool-integrated reasoning agent for mathematical problem solving*, ArXiv **abs/2309.17452** (2023).

[Hal07] Thomas C. Hales, *The Jordan curve theorem, formally and informally*, Amer. Math. Monthly **114** (2007), no. 10, 882–894.

[Hal12] ———, *Dense sphere packings*, London Mathematical Society Lecture Note Series, vol. 400, Cambridge University Press, Cambridge, 2012, A blueprint for formal proofs.

[Hal14] ———, *Mathematics in the age of the Turing machine*, Turing's legacy: developments from Turing's ideas in logic, Lect. Notes Log., vol. 42, Assoc. Symbol. Logic, La Jolla, CA, 2014, pp. 253–298.

[He21] Yang-Hui He, *The Calabi-Yau landscape—from geometry, to physics, to machine learning*, Lecture Notes in Mathematics, vol. 2293, Springer, Cham, [2021] ©2021.

[HHM+10] Thomas C. Hales, John Harrison, Sean McLaughlin, Tobias Nipkow, Steven Obua, and Roland Zumkeller, *A revision of the proof of the Kepler conjecture*, Discrete Comput. Geom. **44** (2010), no. 1, 1–34.

[Hil90] David Hilbert, *Ueber die Theorie der algebraischen Formen*, Math. Ann. **36** (1890), no. 4, 473–534.

[HLOP24] Yang-Hui He, Kyu-Hwan Lee, Thomas Oliver, and Alexey Pozdnyakov, *Murmurations of elliptic curves*, Experimental Mathematics (2024), 1–13.

[Jut77] L. S. Jutting, *Checking Landau's grundlagen in the automath system*, Ph.D. thesis, Eindhoven University of Technology, 1977.

[KEH+09] Gerwin Klein, Kevin Elphinstone, Gernot Heiser, June Andronick, David Cock, Philip Derrin, Dhammika Elkaduwe, Kai Engelhardt, Rafal Kolanski, Michael Norrish, Thomas Sewell, Harvey Tuch, and Simon Winwood, *seL4: Formal Verification of an OS Kernel*, Proceedings of the 22nd ACM Symposium on Operating Systems Principles (SOSP), ACM, 2009, pp. 207–220.

[Kut23] Gitta Kutyniok, *The mathematics of artificial intelligence*, ICM—International Congress of Mathematicians. Vol. 7. Sections 15–20, EMS Press, Berlin, [2023] ©2023, pp. 5118–5139.

[Lau19] Kenny Lau, `Mathlib/RingTheory/Polynomial/Basic.lean`, 2019, https://leanprover-community.github.io/mathlib4_docs/Mathlib/RingTheory/Polynomial/Basic.html.

[LC24a] The Lean Community, *The Lean mathematical library*, Certified Programs and Proofs (CPP) 2020 (Jasmin Blanchette and Catalin Hritcu, eds.), ACM, 2020, pp. 367–381.

[LC24b] ———, *100 theorems*, 2024, https://leanprover-community.github.io/100.html.

[LC24c] Lean Community, *Learning Lean 4*, 2024, https://leanprover-community.github.io/learn.html.

[LC24d] ———, *Lean playground*, 2024, https://live.lean-lang.org/.

[LC24f] ———, *Mathlib naming conventions*, 2024, https://leanprover-community.github.io/contribute/naming.html.
[LC24g] ———, *The Lean zulip chat*, 2024, https://leanprover.zulipchat.com/.
[LC24h] ———, *Lean-related conferences and events*, 2024, https://leanprover-community.github.io/events.html.
[Lea24] Lean4 authors, *Setting up Lean*, 2024, https://docs.lean-lang.org/lean4/doc/quickstart.html.
[LSM+24] Zhaoyu Li, Jialiang Sun, Logan Murphy, Qidong Su, Zenan Li, Xian Zhang, Kaiyu Yang, and Xujie Si, *A survey on deep learning for theorem proving*, 2024, arXiv:2404.09939.
[Mac24] Heather Macbeth, *The mechanics of proof*, 2024, https://hrmacbeth.github.io/math2001/.
[Mas21] Patrick Massot, *Why formalize mathematics?*, 2021, https://www.imo.universite-paris-saclay.fr/~patrick.massot/files/exposition/why_formalize.pdf.
[Mas24] ———, *Lean Blueprints*, 2024, https://github.com/PatrickMassot/leanblueprint.
[McL24] Colin McLarty, *Poincaré on the value of reasoning machines*, Bull. Amer. Math. Soc. (N.S.) **61** (2024), no. 3, 411–422.
[MM24] Patrick Massot and Kyle Miller, *Verbose lean 4*, https://github.com/PatrickMassot/verbose-lean4/tree/master.
[Mor20] Kim Morrison, *Infinitude of primes — a Lean theorem prover demo*, 2020, https://www.youtube.com/watch?v=b59fpAJ8Mfs.
[Mou24] Leo de Moura, *How the Lean language brings math to coding and coding to math*, 2024 Computer Aided Verification (2024), https://www.amazon.science/blog/how-the-lean-language-brings-math-to-coding-and-coding-to-math.
[Mum66] David Mumford, *On the equations defining abelian varieties. I*, Invent. Math. **1** (1966), 287–354.
[Mum20] David Mumford, *The astonishing convergence of AI and the human brain*, 2020, https://www.dam.brown.edu/people/mumford/blog/2020/Astonishing.html.
[NAC24] National Academics Community, *AI for math resources*, 2024, https://docs.google.com/document/d/1kD7H4E28656ua8jOGZ934nbH2HcBLyxcRgFDduH5iQ0/edit.
[Num24] NuminaMath, *How NuminaMath won the 1st AIMO progress prize?*, 2024, https://huggingface.co/blog/winning-aimo-progress-prize.
[OAI24] OpenAI, *Learning to reason with LLMs*, 2024, https://openai.com/index/learning-to-reason-with-llms/.
[Och24] Rodrigo Ochigame, *Automated mathematics and the reconfiguration of proof and labor*, Bull. Amer. Math. Soc. (N.S.) **61** (2024), no. 3, 423–437.
[Ord20] Toby Ord, *The precipice: Existential risk and the future of humanity*, Hachette Book Group, Inc., 2020.
[PF24] Tomaso Poggio and Maia Fraser, *Compositional sparsity of learnable functions*, Bull. Amer. Math. Soc. (N.S.) **61** (2024), no. 3, 438–456.
[Pol98] Robert Pollack, *How to believe a machine-checked proof*, Twenty-five years of constructive type theory (Venice, 1995), Oxford Logic Guides, vol. 36, Oxford Univ. Press, New York, 1998, pp. 205–220.
[PPF+20] Jonathan Protzenko, Bryan Parno, Aymeric Fromherz, Chris Hawblitzel, Marina Polubelova, Karthikeyan Bhargavan, Benjamin Beurdouche, Joonwon Choi, Antoine Delignat-Lavaud, Cédric Fournet, et al., *Evercrypt: A fast, verified, cross-platform cryptographic provider*, 2020 IEEE Symposium on Security and Privacy (SP), IEEE, 2020, pp. 983–1002.
[Pri23] Simon J.D. Prince, *Understanding deep learning*, The MIT Press, 2023.
[PWZ96] Marko Petvšosek, Herbert S. Wilf, and Doron Zeilberger, $A = B$, A K Peters, Ltd., Wellesley, MA, 1996, With a foreword by Donald E. Knuth, With a separately available computer disk.
[San17] Grant Sanderson, *Neural networks*, 2017, https://www.3blue1brown.com/topics/neural-networks.
[Sch21] Peter Scholze, *Half a year of the liquid tensor experiment: Amazing developments*, 2021, https://xenaproject.wordpress.com/2021/06/05/half-a-year-of-the-liquid-tensor-experiment-amazing-developments/.

[Sch22] ———, *Liquid tensor experiment*, Exp. Math. **31** (2022), no. 2, 349–354.

[SHM+16] David Silver, Aja Huang, Chris J. Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, Sander Dieleman, Dominik Grewe, John Nham, Nal Kalchbrenner, Ilya Sutskever, Timothy Lillicrap, Madeleine Leach, Koray Kavukcuoglu, Thore Graepel, and Demis Hassabis, *Mastering the game of Go with deep neural networks and tree search*, Nature **529** (2016), no. 7587, 484–489.

[SHS+18] David Silver, Thomas Hubert, Julian Schrittwieser, Ioannis Antonoglou, Matthew Lai, Arthur Guez, Marc Lanctot, Laurent Sifre, Dharshan Kumaran, Thore Graepel, Timothy Lillicrap, Karen Simonyan, and Demis Hassabis, *A general reinforcement learning algorithm that masters chess, shogi, and Go through self-play*, Science **362** (2018), no. 6419, 1140–1144, arXiv:https://www.science.org/doi/pdf/10.1126/science.aar6404.

[Shu24] Michael Shulman, *Strange new universes: proof assistants and synthetic foundations*, Bull. Amer. Math. Soc. (N.S.) **61** (2024), no. 2, 257–270.

[SP] The Stacks Project Authors, *Stacks Project*, http://stacks.math.columbia.edu, 2024.

[SYA24] Peiyang Song, Kaiyu Yang, and Anima Anandkumar, *Towards large language models as copilots for theorem proving in Lean*, 2024, arXiv:2404.12534, https://github.com/lean-dojo/LeanCopilot.

[Tao23] Terence Tao, *A Maclaurin type inequality*, 2023, arXiv:2310.05328.

[Tao24] ———, *Equational theories project*, 2024, https://teorth.github.io/equational_theories/Âă.

[Thu10] Bill Thurston, *What's a mathematician to do?*, MathOverflow, 2010, https://mathoverflow.net/q/44213.

[Tra19] Andrew W. Trask, *Grokking deep learning*, Manning Publications, 2019.

[Ven24] Akshay Venkatesh, *Some thoughts on automation and mathematical research*, Bull. Amer. Math. Soc. (N.S.) **61** (2024), no. 2, 203–210.

[Wag21] Adam Zsolt Wagner, *Constructions in combinatorics via neural networks*, CoRR **abs/2104.14516** (2021), arXiv:2104.14516.

[Wei56] André Weil, *Abstract versus classical algebraic geometry*, Proceedings of the International Congress of Mathematicians, 1954, Amsterdam, vol. III, Erven P. Noordhoff N. V., Groningen, 1956, pp. 550–558.

[Wie24] Freek Wiedijk, *Formalizing 100 theorems*, 2024, https://www.cs.ru.nl/~freek/100/.

[Wil23] Rebecca Willett, *Mathematical foundations of machine learning*, 2023, National Academies, AI to Assist Mathematical Reasoning: A Workshop.

[Wil24] Geordie Williamson, *Is deep learning a useful tool for the pure mathematician?*, Bull. Amer. Math. Soc. (N.S.) **61** (2024), no. 2, 271–286.

[WS23] Sean Welleck and Rahul Saha, *LLMSTEP: LLM proofstep suggestions in Lean*, arXiv preprint arXiv:2310.18457 (2023), https://github.com/wellecks/llmstep.

[WWS+22] Jason Wei, Xuezhi Wang, Dale Schuurmans, Maarten Bosma, Fei Xia, Ed Chi, Quoc V Le, Denny Zhou, et al., *Chain-of-thought prompting elicits reasoning in large language models*, Advances in neural information processing systems **35** (2022), 24824–24837.

Current address: Department of Mathematics, University of Washington, Box 354350, Seattle, WA 98195-4350, USA

Email address: jarod@uw.edu