Machine-Assisted Proof

Terence Tao

Mathematicians have relied upon computers (human, mechanical, or electronic) and machines to assist them in their research for centuries (or even millennia, if one considers early calculating tools such as the abacus). For instance, ever since the early logarithm tables of Napier and others, mathematicians have known the value of constructing large data sets of mathematical objects to perform computations and to make conjectures. Legendre and Gauss used extensive tables of prime numbers compiled by human computers to conjecture what is now known as the Prime Number Theorem; a century and a half later, Birch and Swinnerton-Dyer similarly used early electronic computers to generate enough data on elliptic curves over finite fields to propose their own celebrated conjecture on these objects. And many readers have undoubtedly taken advantage of one of the broadest mathematical data sets of all, the Online Encyclopedia of Integer Sequences, which has generated numerous conjectures and unexpected connections between different areas of mathematics, and serves as a valuable mathematical search engine for researchers looking for literature on a mathematical object which they do not know the name of, but which they can associate with a sequence of integers. In the 21st century, such large databases also serve as crucial training data for machine learning algorithms, which promise to automate, or at least greatly facilitate, the process of generating conjectures and connections in mathematics.

Besides data generation, another venerable use of computers has been in *scientific computation*, which is heavily used nowadays to numerically solve differential equations and dynamical systems, or to compute the statistics of large matrices or linear operators. An early example of such computation arose in the 1920s, when Hendrik Lorentz assembled a team of human computers to model the fluid flow around the *Afsluitdijk*—a major dam then under construction in the Netherlands; among other things, this calculation was notable for pioneering the now-standard device of floating point arithmetic. But modern computer algebra systems (e.g., Magma, SAGEMath, Mathematica, Maple, etc.), as well as more general-purpose programming languages, can go well beyond traditional "number-crunching," they are now routinely used to perform symbolic computations in algebra, analysis, geometry, number theory, and many other branches of mathematics. Some forms of scientific computation are famously unreliable due to roundoff errors and instabilities, but one can often replace these methods with more rigorous substitutes (for instance, replacing floating point arithmetic with interval arithmetic), possibly at the expense of increased runtime or memory usage.

Relatives of computer algebra systems are *satisfiability (SAT) solvers* and *satisfiability modulo theories (SMT) solvers*, which can perform complex logical deductions of conclusions from certain restricted sets of hypotheses, and generate proof certificates for each such deduction. Of course, satisfiability is an NP-complete problem, so these solvers do not scale past a certain point. Here is a typical example of a result proved using a SAT solver:

Theorem 0.1 (Boolean Pythagorean triples theorem [HKM16]). *The set* $\{1,\dots,7824\}$ *can be partitioned into two classes, neither of which contains a Pythagorean triple* (a,b,c) *with* $a^2+b^2=c^2$*; however, this is not possible for* $\{1,\dots,7825\}$.

The proof required four CPU-years of computation and generated a 200 terabyte propositional proof, which was later compressed to 68 gigabytes.

Computers are of course also used routinely by mathematicians for mundane tasks such as writing papers and communicating with collaborators. But in recent decades, several promising new ways to use computers to assist in mathematical research have emerged:

- *Machine learning algorithms* can be used to discover new mathematical relationships, or generate potential examples or counterexamples for mathematical problems.
- *Formal proof assistants* can be used to verify proofs (as well as the output of large language models), allow truly large-scale mathematical collaborations, and help build data sets to train the aforementioned machine learning algorithms.
- *Large language models* such as ChatGPT can (potentially) be used to make other tools easier and faster to use; they can also suggest proof strategies or related work, and even generate (simple) proofs directly.

Terence Tao is a professor of mathematics at the University of California, Los Angeles. His email address is tao@math.ucla.edu.

Communicated by Notices *Associate Editor Laura P. Schaposnik.*

DOI: https://doi.org/10.1090/noti3041

Each of these types of tools has already found niche applications in different areas of mathematics, but what I find particularly intriguing is the possibility of combining these tools together, with one tool counteracting the weaknesses of another. For instance, formal proof assistants and computer algebra packages could filter out the now-notorious tendency of large language models to "hallucinate" plausible-looking nonsense, while conversely these models could help automate the more tedious aspects of proof formalization, as well as provide a natural language interface to run complex symbolic or machine learning algorithms. Many of these combinations are still only at the proof-of-concept stage of development, and it will take time for the technology to mature into a truly useful and reliable tool for mathematicians. However, the early experiments do seem to be encouraging, and we should expect some surprising demonstrations of new mathematical research modalities in the near future; not the science-fiction conception of a superintelligent AI that can solve complex mathematical problems autonomously, but a valuable assistant that can suggest new ideas, filter out errors, and perform routine case checking, numerical experiments, and literature review tasks, allowing the human mathematicians in the project to focus on the exploration of high-level concepts.

1. Proof Assistants

The mere fact that a computation was performed using a computer does not, of course, automatically guarantee it is correct. The computation could incur numerical errors, such as those caused by replacing continuous variables or equations with discrete approximations. Bugs can be inadvertently introduced into the code, or the input data may itself contain inaccuracies. Even the compiler that the computer uses to run the code could be flawed. Finally, even if the code executes perfectly, the expression that is correctly computed by the code may not be the expression that one actually wanted for the mathematical argument.

Early computer-assisted proofs experienced many of these issues. For instance, the original proof of the four-color theorem [AH89] by Appel and Haken in 1976 revolved around a list of 1834 graphs that needed to obey two properties, called "reducibility" and "unavoidability." Reducibility could be checked by feeding each graph one at a time into a custom-written piece of software; but unavoidability required a tedious calculation comprising hundreds of pages of microfiche—verified by hand through the heroic efforts of Haken's daughter Dorothea Blostein—which ended up containing multiple (fixable) errors. In 1994, Robertson, Sanders, Seymour, and Thomas [RSST96] attempted to make the computational component of the Appel–Haken proof fully verifiable by computer, but ended up instead producing a simpler argument (involving just 633 graphs, and an easier procedure to verify unavoidability) that could be verified much more efficiently by computer code written in any number of standard programming languages.

Proof assistants take this formalization one step further, being a special type of computer language that is designed not to perform purely computational tasks, but to verify the correctness of the conclusion of a logical or mathematical argument. Roughly speaking, each step in a mathematical proof corresponds to a number of lines of code in this language, and the overall code only compiles if the proof is valid. Modern proof assistants, such as *Coq*, *Isabelle*, or *Lean*, intentionally try to mimic the language and structure of mathematical writing, although they are often substantially fussier in many respects. As a simple example, in order to interpret a mathematical expression such as a^b, a formal proof assistant may require one to specify precisely the "type" of the underlying variables a, b (e.g., natural numbers, real numbers, complex numbers), in order to determine which exponentiation operation is being used (which is particularly important for expressions such as 0^0, which have slightly different interpretations under different notions of exponentiation). Much effort has been placed into developing automated tools and extensive libraries of mathematical results to manage these low-level aspects of a formal proof, but in practice the "obvious" parts of a mathematical argument can often take longer to formalize than the "important" parts of the argument. To give just one example: given three sets A_1, A_2, A_3, a mathematician might work with the Cartesian products $(A_1 \times A_2) \times A_3$, $A_1 \times (A_2 \times A_3)$, and $\prod_{i \in \{1,2,3\}} A_i$ interchangeably, since they are "obviously" the "same" object; but in most formalizations of mathematics, these products are not *actually* identical, and a formal version of the argument may need to invest some portion of the proof establishing suitable equivalences between such spaces, and ensuring that statements involving one version of this product continue to hold for the other.

For this and other reasons, the task of converting a proof written by a human mathematician—even a very careful one—to a formal proof that compiles in a formal proof assistant is quite time consuming, although the process has gradually become more efficient over time. The aforementioned four-color theorem was formalized in Coq by Werner and Gonthier in 2005 [Gon08]. The infamous Kepler conjecture on the densest packing of $\mathbf{R}^3$ by unit balls was proven by Hales and Ferguson in 1998 [Hal05] in a very complicated (and computer-assisted) proof. In 2003, Hales launched the *Flyspeck project* to formally verify the proof, estimating that it would take 20 years to do so, although it ended up that, through a collaboration between Hales and 21 other contributors, this was achieved in "only" 11 years [HAB+17]. More recently, Scholze in

2019 launched the "liquid tensor experiment" [Com22] to formally verify a fundamental theorem of himself and Clausen on the vanishing of a certain Ext group of a "liquid vector space" in the theory of condensed mathematics. The human-written proof was "only" ten pages long, albeit with an enormous amount of prerequisite material in condensed mathematics; nevertheless, the formalization in Lean took about 18 months in a large collaborative effort. I myself led a formalization effort [Tao23] on the recent proof of Gowers, Green, Manners, and myself [GGMT23] of a conjecture in additive combinatorics; the human-written proof was 33 pages long, but largely self-contained, and a group of about 20 collaborators was able to formalize it in three weeks. Some fields of mathematics are more challenging to formalize than others; Kevin Buzzard has recently announced a project to formalize the proof of Fermat's Last Theorem, which he estimates will take at least five years.

Given all the effort required, what is the value of proof formalization efforts to mathematics? Most obviously, it provides an extremely high level of confidence that a given result is correct, which is particularly valuable for results that are controversial or notorious for attracting false proofs, or for particularly lengthy proofs in fields where referees willing to verify such proofs line-by-line are in short supply. (Theoretically there could still be a hidden bug in the proof assistant compiler—which is deliberately kept as small as possible to reduce this possibility—or the definitions used in the formal statement of the result may differ in subtle but important ways from the human-readable statement, but such a scenario is unlikely, especially if the formal proof tracks the human-written proof closely.) The formalization process typically uncovers minor issues in the human proof, and can sometimes reveal simplifications or strengthenings of the argument, for instance by revealing that a seemingly important hypothesis in a lemma was in fact unnecessary, or that a low-powered but more general tool can be used in place of an advanced but specialized one. A formalization project in a modern language such as Lean will typically contribute many basic mathematical results generated through the course of the project to a common mathematical library, which makes it easier for future formalization projects to proceed.

But formal proof assistants can also enable new modalities of mathematical education and collaboration. Several experimental projects are underway to take a formal proof and convert it into more human-understandable forms, such as an interactive text in which individual steps in the argument can be expanded into more detail or collapsed into a high-level summary; this could be a particularly suitable format for future mathematical textbooks. A traditional mathematics collaboration rarely involves more than five or so coauthors, in part due to the need for each coauthor to trust and verify the work of every other; but formalization projects routinely involve scores of people who may have had no prior interaction, precisely because the formal proof assistant allows for individual subtasks in the project to be precisely defined and verified independently of the other subtasks. It is conceivable that these proof assistants could also allow a similar division of labor for the generation of new mathematical results, allowing for highly parallelized and crowdsourced collaborations at a far larger scale than previous online collaborations (such as the "Polymath" projects [Gow10]) which were limited by the need to have human moderation of the discussion. In time, the large collaborations that are already established practice in other sciences, or in software engineering projects, may also become commonplace in research mathematics; some contributors may play the role of "project managers," focusing for instance on establishing precise "blueprints" that break the project down into smaller pieces, while others could specialize to individual components of the project, without necessarily having all the expertise needed to understand the project as a whole.

Before this can happen, however, the formalization process needs to become more efficient. The "de Bruijn factor" (the ratio between the difficulty of writing a correct formal proof and a correct informal proof) is still well above 1 (I estimate ~ 20), but dropping. I believe there is no fundamental obstacle to dropping this ratio below 1, especially with increased integration with AI, SMT solvers, and other tools; this would be transformative to our field.

2. Machine Learning

Machine learning refers to a broad array of techniques for training a computer to perform a complex task—such as predicting an output corresponding to a given input drawn from a very broad class, or discerning correlations and other relationships in a data set. Many popular models for machine learning use some form of a *neural network* to encode how the computer will perform the task. These networks are functions of many variables formed by composing together a large number of simpler operations (both linear and nonlinear); typically one assigns some sort of reward function (or loss function) to such a network, for instance by empirically measuring its performance against a training data set, and then performs a computationally intensive optimization to find choices of parameters for this network to make the reward function as large as possible (or loss function as small as possible). These models have countless practical applications, for instance in image and speech recognition, recommendation systems, or fraud detection. However, they usually do not come with strong guarantees of accuracy, particularly when applied to inputs that are significantly different from the training data set, or when the training data set is noisy or

incomplete. Furthermore, the models are often opaque, in the sense that it is difficult to extract from the model a human-understandable explanation of why the model made a particular prediction, or to understand the model's behavior in general. As such, these tools would appear at first glance to be unsuited for research mathematics, where one desires both rigorous proof and intuitive understanding of the arguments.

Nevertheless, there have been recent promising use cases of a suitably chosen machine learning tool to produce, or at least suggest, new rigorous mathematics, particularly when combined with other, more reliable techniques that can validate the output of these tools. For instance, a fundamental problem in the mathematical theory of fluid equations such as the Euler or Navier–Stokes equations is to be able to rigorously demonstrate finite time blowup of solutions u in finite time from smooth initial data. The most notorious instance of this concerns the incompressible Navier–Stokes equations in three dimensions, the resolution of which is one of the (unsolved) Millennium Prize problems; this still remains out of reach, but recent progress has been made on other fluid equations, such as the Boussinesq equations in two dimensions (a simplified model for the incompressible Euler equations in three dimensions). One route to establishing such a singularity lies in constructing a *self-similar* blowup solution u, which is described by a lower-dimensional function U that solves a simpler PDE. A closed-form solution for this PDE does not seem to be available; but if one can produce a sufficiently high-quality *approximate* solution $\tilde{U}$ to this PDE (which approximately obeys certain boundary conditions), it can be possible to then rigorously demonstrate an *exact* solution U by an application of perturbation theory (such as those based around fixed point theorems). Traditionally, one would use numerical PDE methods to try to produce these approximate solutions $\tilde{U}$, for instance by discretizing the PDE into a difference equation, but it can be computationally expensive to use such methods to obtain solutions with the desired level of accuracy. An alternate approach was proposed in 2019 by Wang, Lai, Gómez-Serrano, and Buckmaster [WLGSB23], who used a *Physics Informed Neural Network* (PINN) trained to generate functions $\tilde{U}$ that minimized a suitable loss function measuring the extent to which the desired PDE and boundary conditions are being approximately obeyed. As these functions $\tilde{U}$ are generated through a neural network rather than a discretized version of the equation, they can be faster to generate, and potentially less susceptible to numerical instabilities. As it turned out, a contemporaneous work by Chen and Hou [CH22] was able to establish finite time blowup for this equation using more traditional numerical methods; however, the machine learning paradigm shows great potential as a complementary approach to these sorts of PDE problems. For instance, one can envisage a hybrid approach in which a human mathematician first proposes a blowup ansatz, for which a neural network then tries to find a rough approximate solution, and then more-traditional numerical methods are used to refine that solution to one that is accurate enough for the rigorous stability analysis to be applied.

Another example of the use of machine learning in mathematics is in the field of knot theory. Knots have an extremely diverse set of topological invariants: the signature of a knot is an integer associated to the homology of a surface bounding the knot (a Siefert surface); the Jones polynomial of a knot can be described using the representation theory of braids; and most knots (excluding torus knots) have a canonical hyperbolic geometry on the complement that can be used to describe a number of hyperbolic invariants, such as hyperbolic volume; and so forth. *A priori*, it is not obvious how these invariants coming from very different areas of mathematics are related to each other. However, in 2021, Davies, Juhász, Lackenby, and Tomasev [DJLT21] investigated this issue through machine learning. By training a neural network on an existing database of nearly two million knots (together with a million randomly generated additional knots), they were able to make this neural network model predict the signature of a knot from about two dozen hyperbolic invariants with high accuracy. However, the prediction function generated was quite opaque and did not initially reveal much insight about what the precise relationship between signature and hyperbolic invariants was. Nevertheless, it was possible to proceed further by a simple tool known as *saliency analysis*, which, roughly speaking, measured the influence each individual hyperbolic invariant had on the prediction function. This analysis revealed that of the two dozen hyperbolic invariants used, only three of them (longitudinal translation, and the real and complex parts of meridional translation) had a significant effect on the prediction function. By visually inspecting scatterplots of the signature against these three invariants, the authors were able to conjecture a more comprehensible relationship between these quantities. Further numerics disproved their initial conjecture, but suggested a modified version of the conjecture which they were able to prove rigorously. This interplay between machine-generated conjectures and human verification (and modification) using theory is a promising paradigm that seems applicable to many other fields of mathematics.

Many of the applications of machine learning require a large amount of training data, ideally represented in some standardized format (e.g., a vector of numbers) so that existing machine learning algorithms can be applied to it with relative ease. The precise representation of data can be of critical importance; a correlation between different

components of the data may be easily discoverable by machine learning algorithms in one data representation, but nearly impossible to find in another. While some fields of mathematics are beginning to compile large databases of useful objects (e.g., knots, graphs, or elliptic curves), there are still many important classes of more vaguely defined mathematical concepts that have not been systematically placed into a form usable for machine learning. For instance, returning to the example of PDE, there are thousands of different partial differential equations studied in the literature, but often with large variability in notation and algebraic arrangement of terms, and there is nothing resembling a standard database of commonly studied PDEs together with their basic properties (e.g., whether they are elliptic, parabolic, or hyperbolic; what is known about the existence and uniqueness of solutions, conservation laws, etc.). Such a database could potentially be useful in making conjectural predictions about the behavior of one PDE based on results for other PDEs, or in suggesting possible analogies or reductions from one PDE to another; but the lack of any canonical normal form to represent such equations (or at least a "fingerprint" to identify them [BT13]) makes it difficult at present to even build such a database, let alone feed it into a neural network. It is conceivable in the future though that advances in both proof formalization and artificial intelligence may make it more feasible to generate and utilize such databases (which could contain both "real-world" and "synthetic" sets of data).

3. Large Language Models

Large language models (LLMs) are a relatively recent type of machine learning model that is suited for training on extremely broad and large data sets of natural language texts. A popular large language model is the GPT (generative pretrained transformer), which as the name suggests is built around the *transformer* model—a variant of a neural network designed to predict the next word (or "token") in a string of words, in which some long-term "attention" to words early in the string is retained in order to simulate the context of the sentence. By iterating this model, one can then produce a lengthy text response to a given text prompt. When trained on a small amount of data, the outputs of such models are unimpressive—not much more sophisticated than trying to iterate the "autocomplete" text input feature on a smartphone, for instance—but after extensive training on extremely large and diverse data sets, the outputs of these models can be surprisingly coherent and even creative, and can generate text that is difficult to distinguish from human writing at first glance, although on closer inspection the output is often nonsensical and not connected to any ground truth, a phenomenon known as "hallucination."

One can of course attempt to apply such general-purpose LLMs to try to attack mathematical problems directly. Occasionally, the results can be quite impressive; for instance Bubeck et al. document a case in which the powerful large language model GPT-4 was able to provide a complete and correct proof of a problem from the 2022 International Mathematical Olympiad, which was not in the training data set for this model. Conversely, the model is not well suited for performing precise calculations, or even basic arithmetic; in one instance [BCE+23], when asked to compute the expression $7 * 4 + 8 * 8$, GPT-4 promptly came up with the incorrect answer of 120, then proceeded to justify the calculation with a step-by-step procedure that returned the correct answer of 92. When questioned about the discrepancy, GPT-4 could only offer that its initial guess was a "typo." These issues can be somewhat compensated for by using "plugins" for GPT-4, in which it is trained to send specific types of queries, such as mathematical calculations, to an external tool (such as Wolfram Alpha) rather than to guess the answer through its internal model, although the integration between the tools are not seamless at present. In a somewhat similar vein, a recent proof-of-concept [RPBN+23] has shown that LLMs can be used to find examples in various problems in combinatorics and computer science that outperform previous human-generated examples, by asking those models to generate a program to create such examples rather than to try building the example directly, and then executing that program in another language to reliably verify the quality of the output, which is then sent back to the original model to prompt it to improve its guesses. There has also been recent progress in using LLMs to enhance existing symbolic proof engines to attack narrow classes of mathematics problems, such as Olympiad geometry problems [TWL+24].

In my own experiments with GPT-4 (which you can find at `https://terrytao.wordpress.com/mastodon-posts/`), I have found the most productive use cases have been to generate basic computer code in various languages (Python, SAGE, LaTeX, Lean, regex, etc.), or to clean up messy and unorganized sets of data (e.g., to arrange a pile of references scraped from the internet into a coherent LaTeX bibliography, after providing GPT-4 with a few examples of the desired format for the bibliography items to get it started). In such cases, it often produced satisfactory or nearly-satisfactory output on the first attempt, with only a small amount of revision needed to obtain the type of output I was seeking. I have also had some limited success in getting GPT-4 to suggest relevant literature or techniques for an actual math problem. In one test case, I asked it how one would calculate the rate of exponential decay in the tail probability of a sum of independent random variables, in order to assess whether it knew about the

relevant theorem in this regard (Cramér's theorem) without providing it with key words such as large deviation theory. As it turned out, GPT-4 did not exactly locate this theorem and instead produced a string of mathematical nonsense, but curiously it did manage to reference the logarithmic moment generating function, which is a key notion in the statement of Cramér's theorem, even if it did not seem to "know" exactly how this function was relevant to the problem. In another experiment, I asked GPT-4 for suggestions on how to prove a combinatorial identity I was working on. It gave a number of suggestions that I had already considered (asymptotic analysis, induction, numerics) as well as some generic advice (simplify the expressions, look for similar problems, understand the problem), but also suggested a technique (generating functions) that I had simply overlooked, and ended up solving the problem fairly readily. On the other hand, such a list of advice would probably have been of little use to a novice mathematician, who did not have enough sufficient experience to independently gauge the usefulness of each of the proposed suggestions. Nevertheless, I see a role for these tools in drawing out a user's latent knowledge of a problem, simply by being a good listener and proposing reasonably relevant ideas that the user is expert enough to evaluate.

Github Copilot is another GPT model that is integrated into several popular code editors. Trained on large data sets of code in different languages, it is designed to make autocomplete suggestions for code that is partially written, utilizing contextual clues such as informal descriptions of the task to be performed elsewhere in the code. I have found it works surprisingly well for writing mathematical LaTeX, as well as formalizing in Lean; indeed, it assisted in writing this very article by suggesting several sentences as I was writing, many of which I retained or lightly edited for the final version. While the quality of its suggestions is highly variable, it can sometimes display an uncanny level of simulated understanding of the intent of the text. For instance, when writing another expository LaTeX note on how to estimate integrals, I described how the integral was to be broken up into three parts, and then gave the details of how to estimate the first part. Copilot then promptly suggested how to estimate the second part by a similar method, changing the variables around in what turned out to be a completely correct fashion. The frequency of these experiences has led to a small but noticeable speedup in my writing of both LaTeX and Lean, and I expect these tools to become even more useful in the future, as it becomes possible to "fine-tune" these models on one's personal writing style and preferences.

4. Can These Tools Be Combined?

The various technologies discussed above have very diverse strengths and weaknesses, and none of them at their present level of development are suitable as general-purpose tools for mathematicians, on par with ubiquitous platforms such as LaTeX or the arXiv. However, there are promising recent experiments in creating more satisfactory tools by combining two or more separate technologies together. For instance, one plausible way to combat the hallucinatory nature of LLMs when generating proofs is to require the model to format its output in the language of a formal proof assistant, with any errors generated by the assistant sent back to the model as feedback. This combined system seems suitable for generating short proofs of simple statements [YSG+23]; as such tasks are often the limiting factor in efficiently formalizing proofs, this type of paradigm could greatly accelerate the speed of such formalization, particularly if these models become fine-tuned on formal proofs in particular, as opposed to general text, and are integrated with more traditional automated theorem proving methods, such as the deployment of SMT solvers.

With their ability to take natural language inputs, LLMs are also potentially a user-friendly interface that allows mathematicians without particular software expertise to use advanced tools. As mentioned before, I and many others already routinely use such models to generate simple code in various languages (including symbolic algebra packages), or to create intricate diagrams and images; it seems reasonable to expect that in the near future, one could also communicate through such models to design and operate something as complex as a machine learning model using only high-level, conversational instructions. More ambitiously, one could hope eventually to be able to generate (first drafts of) entire research papers, complete with formal verification, by explaining a result in natural language to an AI, who would try to formalize each step of the result and query the author whenever clarification is required.

The human-intensive nature of formal proof verification in its current form means that it is not feasible at present for a significant fraction of current research papers to be fully formalized in real time. However, it is plausible that many of the tools already used to verify specific computation-intensive components of a research paper, such as a numerical integration or PDE solver, a symbolic algebra computation, or a result established using an SMT solver, could be modified to produce formal proof certificates. Furthermore, the class of calculations that could be formalized in this fashion could be greatly expanded from where things stand in current practice. To give just one example, in the field of PDE, it is common to devote pages of calculation to estimating some integral expression

involving one or more unknown functions (such as solutions to a PDE), using bounds on such functions in various function space norms (such as Sobolev space norms), together with standard inequalities (e.g., the Hölder inequality and the Sobolev inequalities), as well as various identities such as integration by parts or differentiation under the integral sign. Such calculations, while routine, can contain typos (such as sign errors) of various degrees of severity, and can be tedious to referee carefully, with the calculations themselves providing little insight beyond verifying that the final estimate holds. It is conceivable that tools could be developed to establish such estimates in an automated or semi-automated fashion, and the current lengthy and unenlightening proofs of such estimates could be replaced by a link to a formal proof certificate. More ambitiously, one might be able to ask a future AI tool to produce the best estimate it can, given some set of initial hypotheses and methods, without first performing some pen-and-paper calculation to guess what that estimate would be. At present, the state space of possible estimates is too complex to be automatically explored in such a fashion; but I see no reason why this sort of automation will not be achievable as technology advances. When this becomes the case, mathematical explorations become possible at scales that are not currently feasible. Continuing the example of PDE, papers in this field typically study one or two equations at a time; but in the future one may be able to study hundreds of equations at once, perhaps working out an argument in full for just one equation and letting AI tools then adapt the arguments to large families of related equations, querying the author as necessary whenever the extension of the arguments is nonroutine. Some hints of this type of large-scale mathematical exploration are beginning to emerge in other areas of mathematics, such as automated exploration of conjectures in graph theory [Wag21].

It is not clear at present which of these experiments will end up being the most successful in bringing advanced computer assistance to the typical working mathematician. Some proofs of concept are not currently scalable, particularly those that are reliant on extremely computationally intensive (and often proprietary) AI models, or require a large amount of expert human input and supervision. However, I am encouraged by the diverse efforts to explore the space of possibilities, and believe that there will be many further examples of novel ways to perform machine-assisted mathematics in the very near future.

5. Further Reading

The subject of machine-assisted proof is quite diffuse, distributed across various areas of mathematics, computer science, and even engineering; while each individual subfield has plenty of activity, it is only recently that efforts have been made to build a more unified community bringing together all of the topics listed here. As such, currently there are few places where one can find holistic surveys of these rapidly developing modalities of mathematics. One starting point is the proceedings [Kor23] of a June 2023 National Academies workshop on "AI to Assist Mathematical Reasoning" (which the author was a co-organizer of); as one of the outcomes of that workshop, Talia Ringer led an effort to compile AI for mathematics resources, the results of which may be found at `https://docs.google.com/document/d/1kD7H4E28656ua8jOGZ934nbH2HcBLyxcRgFDduH5iQ0`. For instance, in that document is a link to the "Natural Number Game" that is an accessible and interactive way to get acquainted with the Lean proof assistant language. Many of the examples discussed here were also drawn from a February 2023 IPAM workshop on "Machine assisted proof" (which the author also co-organized), whose talks may be found online.

We thank the anonymous referee for corrections and suggestions.

References

[AH89] Kenneth Appel and Wolfgang Haken, *Every planar map is four colorable*, Contemporary Mathematics, vol. 98, American Mathematical Society, Providence, RI, 1989. With the collaboration of J. Koch, DOI 10.1090/conm/098. MR1025335

[BT13] Sara C. Billey and Bridget E. Tenner, *Fingerprint databases for theorems*, Notices Amer. Math. Soc. **60** (2013), no. 8, 1034–1039, DOI 10.1090/noti1029. MR3113227

[BCE+23] Sébastien Bubeck, Varun Chandrasekaran, Ronen Eldan, Johannes Gehrke, Eric Horvitz, Ece Kamar, Peter Lee, Yin Tat Lee, Yuanzhi Li, Scott Lundberg, Harsha Nori, Hamid Palangi, Marco Tulio Ribeiro, and Yi Zhang, *Sparks of artificial general intelligence: Early experiments with GPT-4*, 2023. arXiv:2303.12712.

[CH22] Jiajie Chen and Thomas Y. Hou, *Stable nearly self-similar blowup of the 2D Boussinesq and 3D Euler equations with smooth data, parts I and II*, 2022. arXiv:2210.07191; arXiv:2305.05660.

[Com22] Johan Commelin, *Liquid tensor experiment* (German, with German summary), Mitt. Dtsch. Math.-Ver. **30** (2022), no. 3, 166–170, DOI 10.1515/dmvm-2022-0058. MR4469845

[DJLT21] Alex Davies, András Juhász, Marc Lackenby, and Nenad Tomasev, *The signature and cusp geometry of hyperbolic knots*, 2021. arXiv:2111.15323.

[Gon08] Georges Gonthier, *Formal proof—the four-color theorem*, Notices Amer. Math. Soc. **55** (2008), no. 11, 1382–1393. MR2463991

[Gow10] W. T. Gowers, *Polymath and the density Hales-Jewett theorem*, An irregular mind, Bolyai Soc. Math. Stud., vol. 21, János Bolyai Math. Soc., Budapest, 2010, pp. 659–687, DOI 10.1007/978-3-642-14444-8_21. MR2815619

[GGMT23] W. T. Gowers, Ben Green, Freddie Manners, and Terence Tao, *On a conjecture of Marton*, 2023. arXiv:2311.05762.

[HAB+17] Thomas Hales, Mark Adams, Gertrud Bauer, Tat Dat Dang, John Harrison, Le Truong Hoang, Cezary Kaliszyk, Victor Magron, Sean McLaughlin, Tat Thang Nguyen, Quang Truong Nguyen, Tobias Nipkow, Steven Obua, Joseph Pleso, Jason Rute, Alexey Solovyev, Thi Hoai An Ta, Nam Trung Tran, Thi Diep Trieu, Josef Urban, Ky Vu, and Roland Zumkeller, *A formal proof of the Kepler conjecture*, Forum Math. Pi **5** (2017), e2, 29, DOI 10.1017/fmp.2017.1. MR3659768

[Hal05] Thomas C. Hales, *A proof of the Kepler conjecture*, Ann. of Math. (2) **162** (2005), no. 3, 1065–1185, DOI 10.4007/annals.2005.162.1065. MR2179728

[HKM16] Marijn J. H. Heule, Oliver Kullmann, and Victor W. Marek, *Solving and verifying the Boolean Pythagorean triples problem via cube-and-conquer*, Theory and applications of satisfiability testing—SAT 2016, 2016, pp. 228–245, https://doi.org/10.1007/978-3-319-40970-2_15. MR3534782

[Kor23] Samantha Koretsky (ed.), *Artificial intelligence to assist mathematical reasoning: Proceedings of a workshop*, National Academies Press, 2023, http://dx.doi.org/10.17226/27241.

[RSST96] Neil Robertson, Daniel P. Sanders, Paul Seymour, and Robin Thomas, *A new proof of the four-colour theorem*, Electron. Res. Announc. Amer. Math. Soc. **2** (1996), no. 1, 17–25, DOI 10.1090/S1079-6762-96-00003-0. MR1405965

[RPBN+23] Bernardino Romera-Paredes, Mohammadamin Barekatain, Alexander Novikov, Matej Balog, M. Pawan Kumar, Emilien Dupont, Francisco J. R. Ruiz, Jordan S. Ellenberg, Pengming Wang, Omar Fawzi, Pushmeet Kohli, and Alhussein Fawzi, *Mathematical discoveries from program search with large language models*, Nature **625** (December 2023), no. 7995, 468–475, DOI 10.1038/s41586-023-06924-6.

[Tao23] Terence Tao, *Formalizing the proof of PFR in Lean4 using Blueprint: a short tour*, 2023. https://terrytao.wordpress.com/2023/11/18.

[TWL+24] Trieu H. Trinh, Yuhuai Wu, Quoc V. Le, He He, and Thang Luong, *Solving olympiad geometry without human demonstrations*, Nature **625** (January 2024), no. 7995, 476–482, DOI 10.1038/s41586-023-06747-5.

[Wag21] Adam Zsolt Wagner, *Constructions in combinatorics via neural networks*, 2021. arXiv:2104.14516.

[WLGSB23] Y. Wang, C.-Y. Lai, J. Gómez-Serrano, and T. Buckmaster, *Asymptotic self-similar blow-up profile for three-dimensional axisymmetric Euler equations using neural networks*, Phys. Rev. Lett. **130** (2023), no. 24, Paper No. 244002, 6, DOI 10.1103/physrevlett.130.244002. MR4608987

[YSG+23] Kaiyu Yang, Aidan M. Swope, Alex Gu, Rahul Chalamala, Peiyang Song, Shixing Yu, Saad Godil, Ryan Prenger, and Anima Anandkumar, *LeanDojo: Theorem proving with retrieval-augmented language models*, 2023. arXiv:2306.15626.

Terence Tao

Credits

Photo of Terence Tao is courtesy of Terence Tao.