Mathematics is the canary

What long-horizon coding agents did to mathematicians, and what RSEs can learn from the incident and from how they reacted

Dr. Kolen Cheung, Research Software Engineer
Research Software & Analytics Group, University of Exeter

September 30th, 2026

Abstract Suppose we could verify every step of a program, the way a proof is checked, and get a program that is 100% correct. Mathematicians are already at that limit: Lean checks a proof with a small kernel, and agents got good wherever there is a verifiable signal. In September 2026 OpenAI announced a Lean-certified finite-time blowup for the Navier–Stokes equations, found by about 10,000 agents in 88 hours, and the mathematics community reacted with an intensity that software engineers did not. I go through what the kernel does and doesn't check, why I call it an incident rather than an AlphaGo moment, and what mathematicians said they are after: an answer is not a solution, and the product of mathematics is understanding.

Then I dial it back to research software, on two axes. On correctness, we have more leaks and no kernel, so our job is to minimize the leaks, by finding or building an oracle, so that a human check stays feasible when everything else is accelerated. On understanding, mathematics shows that complete correctness still leaves none, so for us it matters more, not less. Following Naur, long-horizon agents write dead programs, and our project structure already makes us deliver some without any LLM. I end with a proposal, one among many, that the value of an RSE lies in the communication and translation layer between researchers and code, and with the questions I don't have an answer to yet.

Table of contents

Suppose software were like mathematics 2
Mathematics is the canary 2
Where this talk goes 3
The limit 3
Where agents got good 3
Lean: proofs as programs 3
Why mathematicians wanted it, before AI 4
Hales and the Kepler conjecture 4
Scholze and the Liquid Tensor Experiment 4
The incident 4
From crutch to certificate 4
The Jacobian conjecture, in two minutes 6
Why a Millennium Prize Problem is a big deal 6
Navier–Stokes, in ninety seconds 7
Four corners, one week 7
Is this an AlphaGo moment? 8
What the kernel doesn't check 9
The reaction 10
An answer is not a solution 10
What the Fields medallists' declaration upholds 11
Not every mathematician agrees 11
Same capability, two communities 12

Dialling back 13
Four differences 13
Two axes 13
The picture 14
Correctness: shrink where you have to look 14
The bottleneck 15
Oracles that carry a theory 15
Why understanding gets lost 15
What do RSEs uphold? 16
Understanding: where our value lies 16
RSE as translator 17
Born dead 17
Over to you 18
What mathematicians uphold. Do we? 18
Questions I don't have an answer to yet 19
Discussion 19
References 19

Slides to this talk: blog.kolen.dev/RSE/UoE/2026-09-30-agentic-math.html

This is a companion write-up to a talk I gave at the RSA technical catch-up on 30 September 2026. It ties together what I have been writing since the OpenAI Navier–Stokes incident, and asks what research software engineers (RSEs) should take from it.

The slides and this write-up come off a single source, so everything that was on screen is below. The prose around it is what I said on the day, cleaned up. Where I ran out of time, which was most of the last section, it is what I would have said, written from the slides and from the posts I was borrowing from. The questions from the audience are kept where they were asked, together with my answers.

Suppose software were like mathematics

- Suppose we could verify every step of a program, the way a proof is checked, and get a program that is 100% correct.
- This is not how most software is written, and not how it will be.
- But take the limit first, then dial back. Mathematicians are already at that limit.
- What happened to them, and how they reacted, tells us what the dial back means for us.

The centre of this talk is what happened to mathematics less than a month ago, when OpenAI announced a solution to the Navier–Stokes Millennium Prize Problem. I claim it is a big deal, not only to mathematicians, but also to RSEs.

The approach I take is a thought experiment first: treat software like mathematics. Suppose we can verify every step of a program, the way a mathematical proof is checked, and get a program that is 100% correct. This is not how most software is written. But I want to take this limit first, see what is happening to mathematics there and what the consequences are, and then dial it back to us, because software is not really like mathematics. I think the dialling back implies two things: understanding becomes even more important, and our job becomes minimizing the leaks.

> **i** From the audience
>
> *Some software is written like this, though. So it is maybe more than a thought experiment, in that some software is already proved.*
>
> Yes. But there is a reason most software is not done like that, and I come back to it in Four differences.

Mathematics is the canary

> Mathematics is the canary in a coal mine for all parts of intellectual life. Its claims can be carefully checked and so we are witnessing the disruption in real time. […] Science, law, public policy, and art will all have to find ways to assess what is valuable amidst the abundant output.

— Bryna Kra, Deep theorems were scarce, September 2026

Research software included. See my link post.

Why should you, as an RSE, care about what happens to mathematicians? There has been a new blog post about mathematics and AI almost every day for the last two or three weeks, and this quote is from one of them, by Bryna Kra (Kra 2026). Mathematics gets hit first because its claims can be carefully checked, so we are witnessing the disruption in real time. And the quote ends on value: everyone else will have to find ways to assess what is valuable amidst the abundant output. We will spend a lot of time thinking about that.

Where this talk goes

The limit

Verifiable signals, and Lean: a proof checked by a machine.

The incident

From the IMO to the Jacobian conjecture and Navier–Stokes.

The reaction

What mathematicians say they are after. And a parallel in cybersecurity.

Dialling back

What changes for research software, on two axes.

Then discussion, and a list of questions I don't have an answer to yet.

The limit is the 100% correct limit: you have a verifiable signal, and long-horizon agents that can give you a proof. Then the incident, or you might say the state of AI mathematics: from medals at the International Mathematical Olympiad (IMO), to the Jacobian conjecture, to Navier–Stokes. Then the reaction, and then back to us.

The limit

Where agents got good

- In May: from autocomplete to agents, and the December 2025 inflection (journal club).
- Why they got good where they did: a **verifiable signal**.
  - At training time: reinforcement learning with verifiable rewards (RLVR).
  - At test time: an agent in a loop runs the tests, reads the error, and fixes it.
- In August: "You never delete human verification. You **relocate** it somewhere smaller." (JIT talk)
- Today: what is the smallest place it can go? Mathematics has an answer.

We talked about this in the journal club in May: agents got very good at finishing long-horizon tasks roughly at the end of last year. And they became really, really good around April this year, the so-called Mythos moment, when a lot of security bugs were discovered this way.

The key to understanding that is what happens when you put an agent in a loop and give it a verifiable signal. It gets good, in two ways. One is at training time: reinforcement learning with verifiable rewards (RLVR), a buzzword of the last year. You ask the agent to do something, sometimes it gets it right, and the part it got right gives a signal back, which is back-propagated so that it can learn. The other is at run time, which is what happens on my computer: the agent gets signals like a compiler error fed back to it, and it tries to figure out what happened. These two very similar kinds of signals make the agents very, very good.

I ended the JIT talk in August on the line that you never delete human verification, you relocate it somewhere smaller: from assembly, to source, to a spec, to a harness you can audit. So today's question is what the smallest place is that it can go, and mathematics has an answer.

Lean: proofs as programs

- A theorem is a type, and a proof is a program of that type. If the proof compiles, the theorem follows from the axioms.
- A small **kernel** checks every inference. Tactics, automation and the `mathlib` library produce the terms that it checks.
- To a programmer: a type checker for mathematical reasoning.

```
theorem add_comm' (a b : Nat) : a + b = b + a :=
  Nat.add_comm a b
```

Software has formal verification too, e.g. seL4, CompCert, and Amazon's use of TLA+. We will come back to why it stays rare.

The other thing we need to understand first is Lean. Lean at its core treats mathematics as a program, and uses a compiler to check that your program has no errors. So if it compiles, that proves the theorem, with some caveats. It has a very small kernel, which humans have verified, and then you bootstrap from it: you write libraries in Lean, like `mathlib`, in modules, and that gives the language the capability to prove serious theorems.

The important word is kernel. However the proof was produced, by a human, by tactics, or by an LLM, a small trusted piece of code checks every step. You don't need to trust whoever wrote the proof.

> **i** From the audience
>
> *One thing I have always struggled with about Lean: sure, it can prove that whatever you put in is a proper proof, but that is not interpretable by humans. Who checks that this is the correct translation of what I want?*
>
> Put a pin in it. That is exactly one of the problems, and it is What the kernel doesn't check.

Why mathematicians wanted it, before AI

Hales and the Kepler conjecture

- Announced 1998, then years waiting for a dozen referees.
- Published 2005: the referees believed it but could not check it fully.
- Formally verified by the Flyspeck project in 2014.

Scholze and the Liquid Tensor Experiment

> I think this may be my most important theorem to date. […] nobody else has dared to look at the details of this […] Better be sure it's correct…

- A challenge to the Lean community in December 2020, completed in 2022.

Nothing to do with advancing AI. Both stories are in Hartnett (2026). Scholze's challenge is on the Xena blog (Scholze 2020).

Mathematicians wanted formal proving long before AI. Lean itself started in 2013. By then a couple of different kinds of proof had appeared in the community. The more famous one is the four-colour theorem, about colouring maps: it was proved by reducing it to a large number of cases, each of which a computer checks, and that gives you a proof that any map can be coloured with four colours. But a proof like that is very difficult to digest.

The one on the slide is the Kepler conjecture. Hales announced a proof in 1998, which was a big deal, and it used a lot of computer programs. Then, in the peer review process, the mathematics community failed him. Working through another person's computation is very alien to mathematicians. It is not how they usually write proofs, it is not how they want to think about proofs, and it doesn't give them more understanding. He thought the referees were looking at the proof, and for years they really weren't. It was published in 2005, seven years later, with the referees saying they believed it but could not check it fully.

So he went for a formal proof, because he really wanted to know that it was true. If humans don't want to check it, why don't we make a computer check it? That project, Flyspeck, succeeded in 2014 (Hales et al. 2017). It is unrelated to Lean (it used HOL Light and Isabelle), but it is one of the motivations for why things like Lean exist.

I didn't go through the right-hand column on the day, but it is the stunning one. Scholze, a Fields medallist, was not sure of his own proof, because nobody else had dared to look at the details. So in December 2020 he asked the Lean community to formalize it (Scholze 2020), and they finished in July 2022.

The incident

From crutch to certificate

July 2024

AlphaProof + AlphaGeometry 2: IMO silver. Problems manually translated into Lean; up to three days.

July 2025

Gemini Deep Think and an OpenAI model: IMO gold, in natural language, within the 4.5 hours.

July 2026

A counterexample to the Jacobian conjecture, found with Claude Fable 5.

September 2026

OpenAI: Navier–Stokes blowup. ~10,000 agents, 88 h search + 17 h Lean, 165 pages.

In 2024, Lean was what let the model do it at all. By 2026, the proof is found in natural language, and Lean is the certificate at the end.

This is a timeline of AI mathematics, if you want to use that term.

In summer 2024, Google DeepMind released AlphaProof and AlphaGeometry, names that ring a bell from AlphaGo. They reached silver-medal level at the IMO. But it ran for up to three days, where the contestants get two sessions of 4.5 hours. Anyway, it proved that a computer with a certain harness can do very complicated mathematics. (By the way, mathematicians wouldn't really think of IMO problems as mathematics. They are a bit more like very difficult arithmetic.)

One detail matters here: it was human-assisted, in that people converted the IMO problems into Lean statements first, and then the system proved them in Lean. So Lean was part of the harness that helped the language model think. Two years ago, which is ages ago, the models were very bad at logic and often made mistakes. With Lean guiding them, when they made a mistake in the logic, the compile failure told them.

In 2025, both Google and OpenAI reached IMO gold level, within the real time limit of 4.5 hours, and reasoning in natural language. So they didn't need that harness anymore.

Then summer 2026. In July, there was a counterexample to the Jacobian conjecture, found with Fable ("A Counterexample to the Jacobian Conjecture" 2026), the best long-horizon agent at the time. And in September, OpenAI announced its solution to the Navier–Stokes Millennium Prize Problem (OpenAI 2026). It took about 10,000 agents and 88 hours of search to find a proof in natural language, and only after that, another 17 hours to convert it into Lean and certify it. It also generated a paper for humans to read, which is 165 pages.

So in 2024, Lean was what let the model do it at all. By 2026, the proof is found in natural language, and Lean is the certificate at the end.

> **i** From the audience
>
> *Lean is a type of source code. Do you know how many lines of code that Lean proof is?*
>
> For this particular one, I don't know. The number I have in mind is that to turn human mathematics into a Lean proof, you multiply by 10 or 20: one page of proof from an undergraduate textbook takes 10 to 20 times as many lines in Lean. The history of this is in a book I really like, Kevin Hartnett's *The Proof in the Code* (2026).

> **i** From the audience
>
> *So we're talking about an amount of Lean code that no human could ever read.*
>
> You can say so. And that is true even of the Liquid Tensor Experiment. Scholze proved something a human can digest, although it is very complicated, and asked the community to help formalize it. Even that is very dense. But it has value, because it is modular: it went into `mathlib`, and it can be reused in the future. To give a spoiler, the AI proofs don't have that feature. Certifying it once is great, but it doesn't give you machinery that you can reuse for a proof in the future.

i From the audience

So the 88 hours were to come up with a natural-language solution, which then needs to be converted to Lean. Has anyone checked that the conversion is faithful? There is still the question of what it is proving.

Once there is a Lean proof, the Lean proof is the thing you need to audit, and whether it is faithful to the natural-language one doesn't matter much anymore. But what it is proving: yes, exactly. That is the same question as before.

The Jacobian conjecture, in two minutes

- A polynomial map $F : \mathbb{C}^n \to \mathbb{C}^n$ whose Jacobian $\det DF$ vanishes nowhere is locally invertible everywhere.
- Keller (1939) conjectured that it is then invertible, with a polynomial inverse. "Locally everywhere does not imply everywhere" (John D. Cook).
- False for $n = 3$ (Alpöge, with Claude Fable 5, July 2026):

$$\begin{aligned} F(z_1,z_2,z_3) = \big(&(1+z_1z_2)^3z_3 + z_2^2(1+z_1z_2)(4+3z_1z_2), \\ &z_2 + 3z_1(1+z_1z_2)^2z_3 + 3z_1z_2^2(4+3z_1z_2), \\ &2z_1 - 3z_1^2z_2 - z_1^3z_3 \big) \end{aligned}$$

- $\det DF = -2$, yet $F(0,0,-\frac{1}{4}) = F(1,-\frac{3}{2},\frac{13}{2}) = F(-1,\frac{3}{2},\frac{13}{2})$.

Easy to check, and it looks like a miracle. Tao still wrote a whole post digesting it. The question was asked by Akhil Mathew.

I want to illustrate what these theorems are, and I chose the Jacobian conjecture because it is easier to digest in this setting.

You have a function mapping from n-dimensional space to n-dimensional space, where the space is complex. You can calculate its derivatives, and their determinant is called the Jacobian. Then you can ask whether it is zero or not. If it is not zero, the function is locally invertible. Think of a line in one dimension: if the slope is non-zero everywhere, you probably have a one-to-one function that you can invert. That is the intuition. There is one more requirement, that it must be a polynomial map, like the example here.

The conjecture is that if it is locally invertible everywhere, then it must be globally invertible. It is just a guess. It may not be true, and it turns out that it is not.

The way it was disproved is by a counterexample, and that is what makes it so digestible: the statement is easy to understand, and the counterexample is easy to understand too. It is one line. Here is a function, calculate the determinant, it is -2, which is nowhere zero. But you can find a few different points that give you the same value, so it cannot be globally invertible. Anyone can check this with a computer algebra system in a minute (Cook 2026).

It also gives you a sense of when a conjecture is interesting to mathematicians: it is very simple to grasp, and it stood for almost a century.

But why this map? F has degree 7, so the determinant could have had degree up to 18, and all of its non-constant coefficients happen to vanish. Tao still wrote a whole post explaining the construction geometrically (Tao 2026a). Checking and understanding are different tasks. Also notice who asked the question: Akhil Mathew. Choosing a question worth asking was the human part.

Why a Millennium Prize Problem is a big deal

- The Clay Mathematics Institute named seven problems in 2000, with $1 million for each: "to recognize achievement in mathematics of historical magnitude".
- The Riemann hypothesis and P vs NP are on the list. Navier–Stokes is the one that needs the least background to state.
- In 26 years, one has been solved: the Poincaré conjecture, by Perelman, who turned the prize down (and the Fields Medal).

OpenAI says it doesn't intend to claim the prize. Clay's rules need publication, two years, and general acceptance by the community.

I skipped this slide on the day. It is here so that you know why Navier–Stokes made the news. The Millennium Prize Problems were chosen in 2000 as some of the most difficult problems mathematicians were grappling with, and of the seven, I'd argue Navier–Stokes is the one requiring the least background to understand the statement and the idea behind the construction.

Navier–Stokes, in ninety seconds

$$\partial_t u + (u \cdot \nabla) u = -\nabla p + \nu \Delta u + f,$$
$$\nabla \cdot u = 0.$$

- u the velocity of the fluid, p the pressure, ν the viscosity, f an external force.
- Start from a smooth, nice initial velocity u_0 (and a smooth, nice f). Does the solution stay smooth forever, or can it blow up in finite time?
- OpenAI: it can blow up. A collapsing vortex spins up, like water turning faster as it nears a drain.

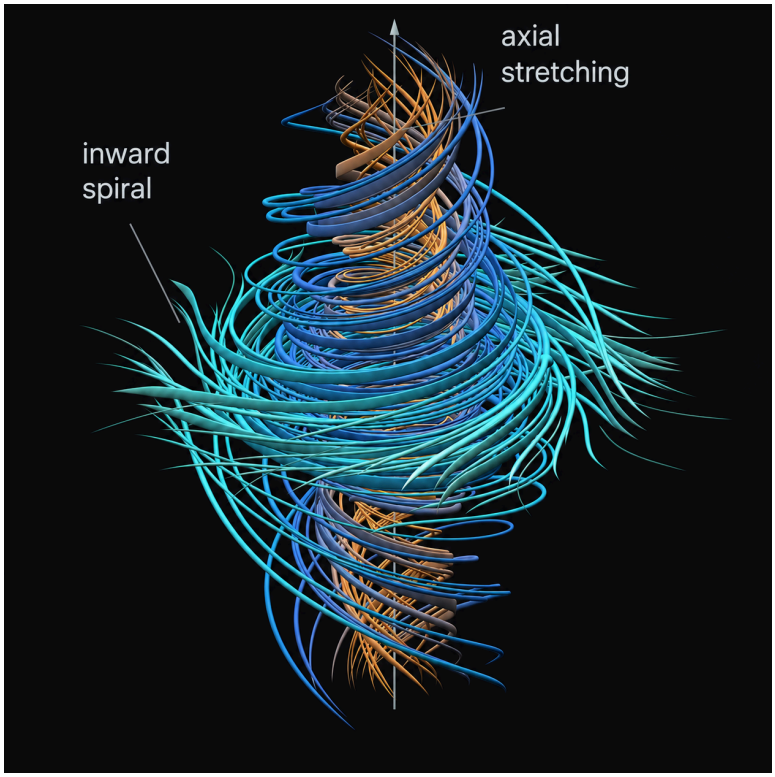


Figure 1: A collapsing vortex: trajectories spiral inward and stretch along the axis. Image: OpenAI.

For the mathematics, see Unpacking the Navier–Stokes blowup. For why I call it an incident, see the series guide.

This one is the big deal. These are the equations, which we know from fluid dynamics. They are just partial differential equations, simple equations with just enough physics to be interesting.

The question is, intuitively: if everything is nice, so the initial velocity of your fluid is smooth, the force you drive it with externally is smooth, and you have viscosity, would the solution always stay nice? What nice means is in my post. Here, just take it as satisfying some good properties, including smoothness.

The prize statement has four forms, A to D. OpenAI proved the C and D forms: when f is non-zero, the solution can run into a singularity. So it is very similar to the last slide, an explicit construction showing that something does not always hold. In this case, that the solution will always be nice. The picture is OpenAI's illustration of the construction: the speed diverges while the total energy stays bounded.

Four corners, one week

	no force, $f = 0$	with a force, $f \neq 0$
Euler, $\nu = 0$	blowup claimed by OpenAI; a separate candidate from Anandkumar's group	blowup claimed by Alpöge and Buckmaster

	no force, $f = 0$	with a force, $f \neq 0$
Navier–Stokes, $\nu > 0$	still open (Clay's A/B)	blowup claimed by OpenAI (Clay's C/D)

- Only the bottom row is a Millennium Prize Problem. Euler is "also open and very important", but off the list.
- The forced approach was already public, in the work of Córdoba and Martínez-Zoroa.

Claims as of 11 September 2026. See the four corners.

I think this table is one of the key things you need in order to understand the incident, meaning how OpenAI affected the mathematics community.

Ask two different questions: is ν zero, and is f zero? A non-zero external force is called forcing in the community. ν is the viscosity, a kind of friction in the fluid that slows things down. The $\nu = 0$ case is called Euler. If you find a counterexample in the forced Navier–Stokes case, that is the C or D form. Navier–Stokes without forcing, where the solution is always nice, is the A or B form. It is still open, and it is much harder: if it turns out to be true, you can't prove it by a counterexample.

What happened is that in the same week, a few different groups of people were solving different corners of this table, and they were essentially forced to announce their proofs in such a hurry that they were not happy with how they released them.

OpenAI solved the unforced Euler corner first. That solution then became part of the prompt for the roughly 10,000 agents: here is the resolution of the Euler case, try to use the idea on the more challenging one.

Alpöge and Buckmaster had been working on forced Euler as a personal collaboration, and rushed their papers out (Buckmaster 2026a), in a form Buckmaster himself calls AI slop (Buckmaster 2026b). Anandkumar's group (Anandkumar 2026) went public roughly half an hour after circulating the work internally. And the forced approach was already public in the work of Córdoba and Martínez-Zoroa, which is the attribution question that comes up again below.

> **i** From the audience
>
> *It may be worth mentioning that they used about 100 agents for the Euler case and 10,000 for the forced Navier–Stokes case. That is the difference in complexity between the problems, I think.*
>
> What you say is true, but I wouldn't read too much into it. It shows that massive parallelism worked. It does not show that it was necessary: Alpöge and Buckmaster got a comparable class of result with two people and commercial tools. Even someone at OpenAI says so. Noam Brown, on the Dwarkesh Podcast: "this was not due to multi-agent. I wouldn't even attribute 10% of the credit to multi-agent" (Brown 2026). Personally, I think the long-horizon capability is mandatory. But does it have to be that parallel? Not so sure.

> **i** From the audience
>
> *Can you remind me what long horizon means in this case?*
>
> Here, the 88 hours is the long horizon: how long an agent can keep going autonomously, in a loop, without going off the rails.
>
> One analogy I can make with humans is string theory. They find the biggest paper they can, like A1, because one single line of the formula is that long, and they keep deriving: this is one line, I make a certain transformation, that is the second line. Another example is calculating the digits of π. It is not that difficult, but if you want 1,000 digits it is very laborious, and centuries ago mathematicians hired other people to do it.
>
> So this is one form of long horizon: you persist in one direction and nothing stops you. You are tireless, you don't feel hungry, and you don't get frustrated and give up to do something else.

Is this an AlphaGo moment?

Like AlphaGo

- A public demonstration of a capability few expected.
- Superhuman persistence and parallelism: 10,000 agents, long calculations nobody would do by hand.

Not like AlphaGo

- Go is fully specified, and a match ranks playing strength across the whole game. Mathematics is many kinds of work: a **jagged frontier**.
- Someone still has to check the result.
- Missing: an abstraction that makes a problem easy, like Wiles' modular forms and elliptic curves, or Scholze's perfectoid spaces.

It doesn't need to be an AlphaGo moment to disrupt how mathematics trains people, assigns credit, and supports careers.

Some people say this is an AlphaGo moment. Buckmaster, in his statement (Buckmaster 2026b), calls it "a Deep Blue-Kasparov moment" for mathematics, referring to the first time a chess world champion was beaten by a machine. I would say not exactly. There are similarities, but there are two reasons it is not.

First, it is a jagged frontier. Even from just the two proofs we have seen, both are by counterexample. Proofs that something never happens, we haven't seen that often. And if you need to build a totally new framework in order to gain the insight to prove something, we haven't seen that happen yet. So it is very good at this kind of thing, and not so good at others.

Second, you still need a human to check it. In chess and in Go it is obvious who wins. There, it is also more difficult to say what the value of the human is. You might say it is part of our culture, but it is hard to say more. In mathematics, and I think in software engineering as well, we can point at it.

This is the other kind of thing we haven't seen. Fermat's Last Theorem was conjectured in the 17th century and proved in 1995. The way Andrew Wiles proved it was to establish a correspondence between elliptic curves and modular forms. Once you have that, you can translate the statement to the other side, and over there it takes a few blackboards. But you need this really abstract correspondence. So far we haven't seen this kind of advance from AI.

And it doesn't need to be an AlphaGo moment to disrupt how mathematics trains people, assigns credit, and supports careers.

> **i** From the audience
>
> *All these proofs seem to be by counterexample, which is the sort of thing where you can just do a lot of work to find one. Are there examples of AI proving that there is no counterexample?*
>
> I don't know as many, partly because there have been so many results in the last couple of months, and partly because we are not mathematicians, so some of the theorems they name don't even ring a bell for me. There must be some. Even among IMO problems, some must be proving that something is true for all cases, where the two here are both "there exists". How much of the recent work is of that kind, I don't know, and it would be interesting to see. But the highest-profile proofs are not like that, and I personally believe it is better at finding counterexamples. There are so many open conjectures, the Erdős problems for example, that it is probably easier to find a famous one you can disprove than one you can prove.

> **i** From the audience
>
> *You're saying we haven't actually learned anything from the Navier–Stokes proof?*
>
> I would say that too. Someone finds a counterexample, boom. It comes up again in how the mathematics community responded.

What the kernel doesn't check

1. **The statement.** Does the Lean theorem say Clay's C/D? Is `divergence` the classical divergence? Does smooth include $t = 0$? A mistake here gives a verified proof of a different theorem.
2. **The assumptions.** No `sorry`, no custom `axiom`, no `native_decide`. `#print axioms` should list only `propext`, `Classical.choice` and `Quot.sound`.

3. **Junk values.** Lean's functions are total: the integral of a non-integrable function is 0. A bound on the energy can hold when the energy is infinite.

Perhaps a day or two of expert attention, instead of a year of refereeing. Details in the Lean section.

This comes back to the question from earlier: what has the Lean proof not given you, and what do you still need to check?

First, the statement. Remember that in 2024, humans converted the IMO problems into Lean statements, and the computer proved those. So humans were doing the verification that the Lean statement really means what we mean. It is the same for any AI proof in Lean. This is the boundary: you need to confirm that the conversion into a Lean statement really captures what we are saying, so that you are not proving the wrong thing. By the way, there is now a project that collects this kind of Lean proof: Palomar, a registry of Lean-verified mathematics, which Tao calls "the analogue of a preprint server for Lean proofs" (Tao 2026b). It checks mechanically that a proof proves exactly the statements claimed, and uses an LLM to check that the informal description matches them. So even there, the fidelity of the statement is the part that is left to judgement.

Then the assumptions. There is something interesting in Lean called `sorry`. I would say it is like `NotImplementedError` in Python: I claim this, and I will do it later. That is how mathematicians write proofs in Lean, ergonomically. They gradually build up the idea, and fill in a `sorry` here and there. In the case of Scholze's theorem, the moment it was proved was the moment the last `sorry` disappeared from the proof.

And there are some other minor things to check, such as junk values, which together guarantee that the proof is really saying what it says.

So on one hand, Lean on its own is not everything. On the other hand, it is a very narrow boundary that you need to check: perhaps a day or two of expert attention, instead of a year of refereeing.

The reaction

An answer is not a solution

- De Toffoli and Duede (2026), After Math: OpenAI produced an *answer*, and an answer is not a *solution*. "Historically, the two notions have tended to run together."
- Thurston (1994): "We are not trying to meet some abstract production quota of definitions, theorems and proofs. The measure of our success is whether what we do enables people to understand and think more clearly and effectively about mathematics."
- Thurston again (2010): "The product of mathematics is clarity and understanding. Not theorems, by themselves."
- The Clay Institute's own page for this problem: "Why ask for a proof? Because a proof gives not only certitude, but also understanding."
- Buckmaster (2026b), on his own Lean-verified Euler write-up: it "can only be described as AI slop."

Mathematicians aren't mourning their craft. See After Math, translated for software engineers.

Now back to the mathematicians, to see how they reacted. If I summarize this slide, they are saying that getting an answer is not the point. De Toffoli and Duede distinguish an answer from a solution. Others say that they are not after an answer, they are after understanding.

And they said so decades ago, long before AI mathematics. Thurston wrote that mathematics is not about theorems, proofs and lemmas. There was already a trend in mathematics as a profession towards publish or perish, where more volume is better, so a leader in the field had to remind everyone: this is a by-product. Understanding is the most important thing.

Software engineers tend to read the reaction to coding agents as craft versus product: the craft-lovers grieve, and the product people move on. Seen that way, mathematicians look like craft-lovers who are still in the bargaining phase. I think that misreads them. They are saying the product was never the true/false answer. The understanding *is* the product.

Let's also try a proof by contradiction. Suppose only the answer matters. What is the value of knowing that the Navier–Stokes equations can blow up? Not knowing it never stopped us from simulating them, and knowing it hasn't changed a thing about how we solve them.

What the Fields medallists' declaration upholds

A Severe Misalignment of AI in Mathematics, 11 September 2026:

- **Understanding:** "solving problems is only a tool and proxy for achieving the primary goal of conceptual understanding and insight"
- **Exposition:** rushed announcements leave "no time for a proper writeup, the isolation of new methods and ideas"
- **Credit:** "severe attribution and plagiarism questions"
- **Students:** "The most precious resources of our profession are students and ideas"

And from the essays that followed: journals should "distinguish and credit the roles of discovery, proof, formalization and explanation" (Kra 2026).

We will come back to this list in the discussion: what is our equivalent of each, and do we uphold it?

The Fields medallists' declaration (Avila et al. 2026) is quite impressive. It started with 25 signatories and it is 27 now. The Fields Medal is very hard to get, and some people call it the Nobel equivalent in mathematics. The joint statement is very short and easy to digest.

The first point is that solving problems is only a tool and proxy for achieving the primary goal of conceptual understanding and insight. That is what mathematics is about, according to them.

Then exposition: you need to teach other people. It has really happened in history that someone came up with a proof they think is correct, and nobody else understands it, so it doesn't have much value in the community. Mochizuki's proof of the abc conjecture is the example. It resembles the AI proofs, and in that sense the AI proofs are not the first to give you an answer that nobody can read.

Then credit. Remember the two-by-two table: the forced approach comes from the work of Córdoba and Martínez-Zoroa, as Buckmaster and others have acknowledged. I think failing to give that work enough recognition accounts for much of the dispute.

And students. Being a mathematics student right now is hugely affected. From what I read, many of them are panicking and asking senior people for help.

Not every mathematician agrees

Timothy Gowers didn't sign, because of the "tool and proxy" sentence:

- "there is a spectrum of attitudes in mathematics to the relationship between problem-solving and conceptual understanding" (his *Two Cultures of Mathematics*, 2000)
- On the flood of AI results: it "would also be likely to increase the amount that *was* properly digested, which seems like a pretty good bargain."
- His worry is elsewhere: "the social structures that currently support this digestion process will be destroyed and not adequately replaced."

Remember the problem-solving end of that spectrum. See A flood from within the community.

Gowers is another very famous mathematician, and he explains why he didn't sign (T. Gowers 2026). Gowers and Tao are two prominent mathematicians who have promoted collaboration in mathematics: massive collaboration, more like open source software than a paper with a few co-authors. The two are very alike in that. But Tao signed the declaration, and Gowers didn't.

The first quote is from 2000 (W. T. Gowers 2000). Even then, he identified two different kinds of motivation. Some mathematicians care more about problem solving: you ask me a problem, I give you a solution. Others emphasize understanding, and something called theory building, which is understanding in a more distilled sense.

His second point is about the flood of AI results. Some of it has already come in the last several months, and there will be more. If a bunch of AI results are released and they all come with understanding, great. But if they only give you an answer, it is still a good bargain. That is his point of view.

What he worries about is something else, which is the social structure of the mathematics community as a whole: how people are rewarded, which is primarily based on proofs, and how students are trained. A student gets an easier problem, works on it for a few months, gets a proof, and gradually gains what is sometimes called mathematical maturity before starting on the hard things. Now think about AI capability. Even if it is a jagged frontier, you can

think of it as a certain level, and it wipes out all the easy problems that a third-year graduate student can prove. What do they do now to gain that capability?

So mathematicians are not one group either. I'll come back to where RSEs sit on his spectrum.

Same capability, two communities

	Cybersecurity	Mathematics
Persistence, parallelism	Mythos Preview chains four vulnerabilities into one browser exploit	10,000 agents over 88 hours
Hard to digest	curl's security reports arriving 4–5× faster than in 2024	a Lean-verified proof still has to be understood
A rumour is enough	probes about ten minutes after a public fix	OpenAI started its search on a rumour, which was wrong
The rush lands on people	Daniel Stenberg and the curl team	Buckmaster, Anandkumar: unfinished work rushed out

Security answered with coordinated disclosure. My proposal for mathematics: progressive disclosure.

I am still in mathematics, but here a parallel with software engineering starts, in particular with cybersecurity.

What happened at the Mythos moment is that very persistent, hugely parallel AI agents arrived. Persistent means they can discover bugs that nobody else has, just because they keep looking. And they can reason about those bugs and chain them together into a more severe vulnerability (Carlini et al. 2026). In mathematics: 10,000 agents and 88 hours to solve the Navier–Stokes problem.

The second property is that the output is hard to digest. curl, which everyone uses, has four to five times the volume of security reports it had in 2024 (Stenberg 2026), and since the Mythos moment many open source projects have been flooded in the same way. Similarly, a Lean-verified proof is a very impressive, very long thing that is difficult to digest.

The third property is kind of interesting: a rumour is enough. Anil Madhavapeddy opened a public pull request fixing a bug in one of his own open source projects, and within about ten minutes his own website was being probed for it. He then verified that an off-the-shelf model, given the affected code and a hint about where to look, could produce an exploit in under a minute (Madhavapeddy 2026). Just tell them there is a reward somewhere here, and they can find it.

In mathematics, OpenAI heard a rumour in late summer that another company had resolved two Millennium Prize Problems, and it launched a large-scale search of its own (OpenAI 2026). Once the agents had the Euler result, the search was pointed at Navier–Stokes. Only later, in its exchanges with Buckmaster in the first week of September, did it turn out that nobody had proved a Millennium Prize Problem: the rumour was about Alpöge and Buckmaster's personal collaboration on forced Euler. So a rumour was enough. Somebody else has proved one, so my AI must be able to do it too; tell the agents there is something to prove here, and they prove it.

Lastly, the rush. Daniel Stenberg of curl describes the flood of reports, his wife worrying about his hours, and his concern for his teammates. Over here, Buckmaster and Anandkumar both had a very rushed last week before releasing their work, trying to get it out before OpenAI's announcement.

Many of us maintain open source, so the left column is closer to home than it looks. In security, a disclosed bug can be exploited, so there is a real deadline. In mathematics there usually isn't one, so the harm is the rush itself, and that is why I call it an incident.

> **i** From the audience
>
> *Do we have a notion of how much it costs? 88 hours of agents, presumably running on very large machines, is quite a considerable expense.*
>
> Yes, we have numbers. Using Simon Willison's calculation (Willison 2026a), OpenAI's output tokens would cost about $15 million at public API prices across all the problems it attempted, including about $6.5 million for Navier–Stokes. The prize is $1 million, and OpenAI isn't claiming it. So one way to think about it is that you spend $15 million to win one.

> **i** From the audience
>
> *And that is to displace the effort of a very small profession. There are orders of magnitude fewer professional mathematicians in the world than software engineers. What we are seeing is a domain with huge wealth stomping over one that just doesn't have the manpower to respond.*
>
> You can say so, and that is why it is so outrageous. One way I think about it is thermal equilibrium. Software engineering and mathematics used to be two isolated gases, and now AI is forcing them to reach an equilibrium, which is very disruptive.

> **i** From the audience
>
> *The two communities also have very different incentives. Mathematicians want understanding, and there is some ethics among them. The labs are not really interested in mathematics. They are more interested in what they can actually monetize, like drug development.*
>
> At this point, it is a benchmark to them. Just like Go was a benchmark to DeepMind: they didn't care about Go.

Dialling back

Four differences

1. **Mathematics is self-contained. We are not.** We work with PIs and domain experts, and have less say in where the work goes.
2. **Requirements are rarely as compact as a conjecture.** Even the master equations and initial conditions don't define a program's behaviour: discretization, approximation and algorithm all have to be specified too.
3. **No watertight abstractions, and no kernel.** Formal verification exists for software, but the specification becomes the laborious part, and see point 2.
4. **No shared culture.** Mathematicians have standards that reached an equilibrium over centuries, and could react as a community. RSEs don't, yet.

Now dial back from the limit. I ran out of time here on the day, so most of this section is what I would have said.

This is also where I come back to the first question from the audience. Some software is formally proved, and Amazon's use of TLA+ is a well-known example. But it stays rare, and I think these differences are why. I wish I could just write down the master equations and the initial conditions, and that would define, somewhat uniquely, how the program should behave. It doesn't. And even if you did formally prove your software, the formal proof would have to include a laboriously enumerated specification as well.

The fourth difference is about us as a community, and I come back to it in What do RSEs uphold?

Two axes

Correctness

- In mathematics, it can be made nearly complete: a kernel, and one opening.
- We have more leaks and no kernel, so there is more to look at.
- **The art is to minimize the leaks**, so that looking becomes feasible.

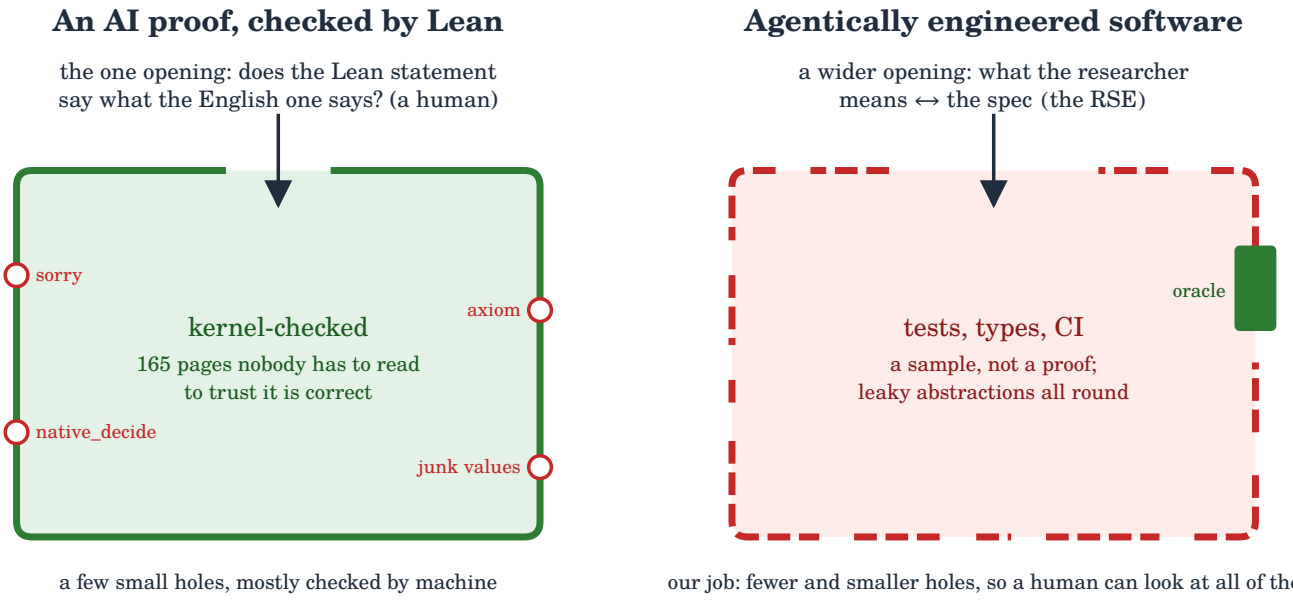
Understanding

- In mathematics, complete correctness still left no understanding.
- We have less correctness assurance, so understanding matters **more**, not less.
- Long-horizon agents write dead programs.

The reflection I had, and you might call it an aha moment, is about having an answer versus having an understanding. In software, I call the first one correctness. I really care that my software is correct, all the time when I write software. This was the moment I realized that making sure it is correct is not enough.

So these are the two implications of dialling back. Correctness is the axis where we kind of have a fix: unit tests, specifications, static analysis, the verifiable signals an agent can pick up and act upon. Understanding is the axis where nothing gives a verifiable signal, and Navier–Stokes shows that fixing correctness alone is not enough.

The picture



An AI proof in Lean is like the left: very watertight. Green means you don't need to check it. Any opening is where you need to inspect: the translation of the statement, `sorry`, axioms, and so on.

Software written by agents is like the right. There is a famous saying that all abstractions are leaky. That only happens to software, by the way. It doesn't happen to mathematics, where they are all watertight. And because of that, you have many more holes: you can't contain a concept in one place, and it appears elsewhere. So there are more things you need to check, and the tests inside are a sample, not a proof.

So our job in designing agentically engineered software is primarily to reduce the number and the surface area of the leaks, so that we can focus our effort on those holes only.

Correctness: shrink where you have to look

- Not new. We never read every line of a human PR either: tests, types, CI and small diffs exist to make review feasible.
- **Find or build an oracle**: something independent, compact, and easy to trust.
  - Scientific computing: the model equations, manufactured solutions, the observed order of convergence, conservation laws.
  - A small pure function reviewed by hand, used to check the fast one.
  - A reference implementation: Bun's 535,000 lines of Zig → Rust in 11 days, against a test suite written in TypeScript. 19 regressions still got through.
  - SQLite accepts agentic bug reports with a reproducible test case, and refuses agentic code.
- Review the script that generates the diff, rather than the diff.

This is not new. Even with a PR written by a human, you often can't take a close look at every line, and standard software engineering practices exist to make human review feasible. With agents, they apply the same way, or even more so.

An oracle is my generalization of what I have been arguing for scientific computing: somewhere that correctness can be defined outside the code, compactly, and checked independently. Where one exists, research software sits closer to the mathematics end. SQLite's policy is the smallest example (Willison 2026b): a reproducible test case gives the maintainer something concrete to check, without having to trust the report. Bun is the largest (Sumner 2026): the old program was the specification and the test suite was the guard, and 19 regressions still got through, because a test suite is a sample.

The bottleneck

- Goldratt's theory of constraints (*The Goal*, 1984): throughput is set by the bottleneck. Speeding up anything else doesn't help. Amdahl's law is the special case for parallel computing.
- Agents speed up everything except the human check, so the human check is the bottleneck.
- The lever we have is its size: design so that the part a human has to look at is small.
- Why not just carry on at our own pace? Baumol's cost disease (Baumol and Bowen, 1966): a string quartet still needs four players and the same 40 minutes, so as everything else gets more productive, it gets relatively more expensive. What only humans do becomes the expensive part, and that is where the pressure to cut lands. Mathematicians and RSEs could become the next artists.

Amdahl's law is about parallelization, but the core reason is beyond parallelization: what you can't accelerate becomes the bottleneck. Goldratt said it for factories (Goldratt and Cox 1984).

Some will say this is no big deal, so why do I need to jump on the bandwagon? I think the answer is Baumol's cost disease (Baumol and Bowen 1966). The bottleneck doesn't stay a fixed, harmless cost. Everything around it gets cheaper, so the human part becomes the expensive part, and the pressure is either to skip it, which is shipping code nobody has read, or to stop paying the people who do it. The arts survive on subsidy, and on people valuing them for their own sake. Gowers already worries that policy-makers will think mathematicians are no longer needed. This can happen to mathematicians, or to RSEs, if we are not careful. The alternative to paying for the part that can't speed up is making it smaller, which is the lever above.

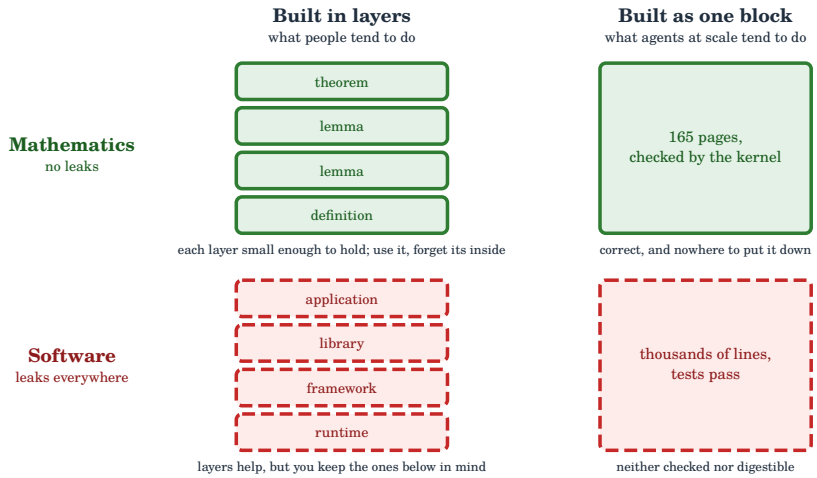
Oracles that carry a theory

- Bun's oracle, the old program and its tests, gave correctness and no understanding.
- In scientific computing, the oracle is the physics, which is itself a theory, and the researcher already holds it.
- So I think scientific computing has an answer on both axes, and product code only on correctness.

See Where do correctness and understanding live in research software? and Can a theory be laid out?

This is why I only claim an answer in the case of scientific computing. Any oracle shrinks where you have to look. An oracle that is also a theory somebody holds does more. For scientific software, the theory of the program is, in large part, the theory of physics, and the expected knowledge of physics does a lot of the heavy lifting in rebuilding the theory of the program. Why are they doing this approximation? It must be X, Y, and Z. Someone with a common understanding of physics can fill in the blank.

Why understanding gets lost



Not people versus machines as such: the four-colour theorem and Hales' Kepler proof were monoliths made by people, and Mochizuki's layers are ones nobody else shares.

This problem, of agents writing something very long, software or a proof, that is indigestible to humans, is not new to software engineers. For the last 18 months or so, people have been saying that it is happening to them: more and more is built by AI, and they are gradually losing their grasp of what is happening.

The picture is an illustration of abstraction. There are two things in it, and they are independent. Solid or dashed is the field: in mathematics all abstractions are watertight, and a lemma an AI proved doesn't leak either. In software you also have modules and layers, but each of them is a little bit leaky. Layers or one block is how it was built. It is an exaggeration, but what an agent gives you is like the right-hand column: it doesn't have much structure. It can of course write classes. What it doesn't build is a theory, which is what I worry about.

A layer is an abstraction when its statement is much smaller than what is inside it, so that you can use it and forget the inside. Why do people build that way? Because we have to. A human can't hold 165 pages in their head, so we invent the definition and the lemma that let us put things down. An agent with 10,000 parallel attempts doesn't share that bottleneck, so nothing pushes it to find the abstraction. Henry Cohn calls the result "cryptic, messy, ad hoc, poorly motivated" (Cohn 2026).

If there is one thing I would bring to the journal club, it is Peter Naur's 1985 paper, *Programming as Theory Building* (1985). It is the closest thing I know to thinking about programming the way mathematicians think about mathematics: you build a theory, and the theory is in your head. For Naur, if nobody holds the theory of a program, the program is dead.

So one horrible conclusion I arrived at is that long-horizon agents write dead programs. If you ask them to just keep writing, and you don't come back to check, the program becomes very complex, and in the end nobody holds its theory.

What do RSEs uphold?

- **Young.** The term was coined in 2012 (the Software Sustainability Institute's Collaborations Workshop). The first RSE conference was in 2016.
- **Not one thing.** Calling us all RSEs is like calling physicists, biologists, chemists and computer scientists all scientists: true, but very different cultures. Even mathematicians have Gowers' two cultures.
- **No shared values yet.** Some of us are strongly against AI. Some are against involvement with the military (RSECon25). Some are deeply concerned about the environment. Some see our value in problem solving. Some in being part of Digital Research Infrastructure, which includes people.
- Without shared values, we can't react to the flood as one community, the way mathematicians did.

I didn't get to the value question on the day. Mathematicians could react with such intensity because they are one community that has, more or less, its own standards, which reached an equilibrium long ago. I feel the value of the RSE as a profession is not so well defined.

RSE is quite new. And a research software engineer is not one thing: calling us a single entity is like saying that physicists, biologists, chemists and computer scientists are all scientists. It is true, but they have very different cultures. As Gowers observed, even among mathematicians the two cultures are a big divide.

So what do we uphold? It varies. And I think not having a common culture and shared values is partly why RSEs, and by extension research data scientists and others, can't be effective in reacting to the flood coming from AI.

Understanding: where our value lies

- Mathematics can say what it values from the inside, because it is self-contained. We can't: our value is defined at the boundary with the researchers.
- Kra: deep theorems were a proxy for understanding, and AI made the proxy cheap.
- For us, the code is the proxy. If code is what we are valued for, an LLM replaces us.
- So what is the product? The theory we build with the researchers: which question they are really asking, what "correct" means to them, which approximation is acceptable.
- My proposal, one among many: our value lies in the communication and translation layer. Technical ability still matters, if not for implementing, then for the taste to translate well and correctly.

This is unsettling, but if true, illuminating. Mathematics is more or less self-contained, and we are not, so that asymmetry pushes us to define our value at the boundary: why do RSEs deserve their place in academia, and what value do they provide?

We only know how to react to AI if we know where our value lies, and why we are irreplaceable. Otherwise RSEs should be replaced by LLMs. I think this is why the mathematics community reacted with such intensity.

Here, I only propose that possibly our value lies in being the communication and translation layer.

RSE as translator

Researcher

What is in their head, and what they say.

RSE

Listening and asking: user stories into requirements. And back again: training.

LLM

A lower-level translator: the spec into a program.

Program

Checked against the oracle, where there is one.

Like checking an English theorem against its Lean statement, the human work is at the boundary. Unlike Lean, there is no kernel after it.

RSE work is primarily problem solving: they have a problem, and our expertise is to solve it technically. Part of the value then lies in communication: extracting user stories, and formulating requirements from them. Extracting user stories is communication. They won't just tell you. The art of listening and asking extracts it from them, much like counselling does. The other direction of communication is training: upskilling them to use what we provide, even though the theory of the program isn't in their head yet.

That is what I mean by the RSE as a translator. We translate, by communicating with real people, what they say and what is in their heads into concrete, definable requirements for research software. An LLM can then act as a lower-level translation layer, which takes the spec we translated into an actual program.

This is similar to formal proving by agents: a human still needs to make sure of the translation between the mathematical statement in natural language and the formally defined statement in Lean. The difference is that after our boundary there is no kernel.

Born dead

- Naur (1985): a program dies when the team holding its theory is dissolved, yet it "may continue to be used for execution in a computer and to produce useful results."
- Cost recovery moves the RSE to the next project. A Julia project: two RSEs designed it, handed it over, and never touched it again. Born dead, without any LLM.
- We are paid for the proxy, the code. The product, the theory, leaves with us.
- A partial way out: the researcher already holds much of the theory (the physics), and training can move more of it across.

I haven't solved this. See Born dead.

There are two implications of Naur's dead program for us. First, we are adopting agentic engineering in our profession, so dead programs are a problem we are going to face. Second, and I think this is even more horrifying, our project structure already makes us deliver dead software.

Think about a Julia library a colleague and I built. One day we went to a whiteboard and drew a diagram, and there was a certain mental picture in our heads. Once the library was delivered, that was gone. The theory is not in the PI's head. And in Naur's sense, even documentation is not enough to carry the theory. That is the horrifying bit. If anything, I've shown that research software is often born dead even without LLMs.

I don't have the answer. I think scientific computing might be a little safer in this regard, because what you write in the program is primarily based on scientific theory, so you already have that theory in mind.

i From the audience

Take the Met Office's Unified Model. None of the science captured in it exists only in the code: any addition to the model is accompanied by a scientific publication describing the new science. The implementation could be flawed, but you could go back and trace all the papers, and create something equivalent, drawn from the science.

That is part of the answer. But it is the theory of the domain. In Naur's sense, the program itself, how the software runs, has a theory too. Think of microservices, where everything passes JSON to everything else: that is a kind of theory, a view of how the world should work. Or you can say, we use object-oriented programming, and this is my theory. I think Naur has in mind something as complicated as Word, which captures a lot of the real world and has no theory outside itself. That is why I say scientific computing may be a bit safer: the way we structure the program is, to some extent, implied by the science.

i From the audience

Hasn't industry considered this too, with something like domain-driven design? You structure the code to reflect the domain it is used in, which doesn't have to be scientific. It could be a business domain, with the ubiquitous language and so on.

You have a point. One difference I'd point out is that science also has the active practice of propagating knowledge. Students come to study physics. There are lectures and seminars, where someone is listening and learning, and the giving is itself a learning process: how can I explain such a complex thing so that others can understand? That is closer to how mathematicians think about propagating understanding. In a business organization, however well you model it, there is no intrinsic theory of how the world works, and no such common practice of reaching a shared understanding. Quantum mechanics is complicated enough, but there are maybe two main ways of teaching it to undergraduates, so physicists can all understand each other.

i From the audience

(From the chat.) I like the fact that you mention dead code. We sometimes call it stale code. I never really thought about our RSE projects going stale the moment we hand them over.

Yes. I don't have an answer to that, but that is my realization. It takes a lot of work on the part of the postdoc, the PhD student or the PI we hand it over to, to bring it back to life.

i From the audience

About that whiteboard conversation: why not have the PI in more of those conversations?

We did try to put it into the documentation, but that kind of documentation isn't always seen as valuable. There is also something special about that mental model, which is that it involves a lot of mathematics. There is a joke that to mathematicians, everything is trivial. If they don't understand something, they keep talking for hours, and at some point it becomes trivial. That is what mathematical maturity means, in a sense: you can distil the understanding into something you can explain in a few lines, like Fermat's Last Theorem in a few blackboards once you have the correspondence. So the theory we had is in itself difficult to transmit, unless the other party is also a mathematician.

Over to you

What mathematicians uphold. Do we?

Mathematicians uphold	Our equivalent? Do we uphold it?
Understanding is the goal (Fields medallists)	communication?
Time for a proper writeup (Fields medallists)	documentation that explains *why*?

Mathematicians uphold	Our equivalent? Do we uphold it?
Credit for formalization and explanation (Kra)	credit for tests, documentation, maintenance?
A solution that opens lines of work for others (Cohn)	open source projects? knowledge exchange?
Students and ideas (Fields medallists, Gowers)	handover, training?

We didn't get to this table on the day. It takes what the mathematicians say they uphold, and asks what our equivalent is, and whether we uphold it. The right-hand column is a list of guesses for the room to argue with.

One more question goes with it: where do RSEs sit on Gowers' spectrum? If RSE work is primarily problem solving, that is exactly the end that agents are automating.

Questions I don't have an answer to yet

- Where is the line between research software you can ship unread, and research software you can't? (research software)
- Can a theory be laid out, and can it be rewarded? (theory and reward)
- Is "dead" relative to the reader, so that complexity sets the horizon beyond which agents write dead programs?
- Under cost recovery, how do we deliver the theory and not just the code?
- What is the RSE's irreplaceable value?

These are the questions I am still working on, and the third one came up in the discussion.

I think whether a theory can be rebuilt is related to complexity. If someone wrote a 100-line program ten years ago and dropped it on me, I could probably rebuild its theory. So the claim that you cannot rebuild a theory must be related to complexity in some way. I am still thinking about it, but it is kind of like a phase transition. We can describe all the physics of the individual atoms in a gas, but we can't use that to describe the gas as a whole, only because it is so complex. You go through a phase transition, and you need something like the Navier–Stokes equations to describe it in another limit. I think a program is like that too: it becomes so complex that local understanding doesn't translate into global understanding. Which is the Jacobian conjecture again: you understand it everywhere locally, and you don't understand the program as a whole.

> **i** From the audience
>
> *That transition, from local understanding not always leading to global understanding, is called emergence, and it is studied everywhere in science. Maybe what you are saying is that we are starting to see it in software. Every floating-point operation is deterministic and fully understandable, and the outcome is beyond us.*
>
> Yes, and probably especially in the kind of software an agent writes. In physics this is sometimes called the Wilsonian view, or effective field theory. It is related to the Navier–Stokes singularity too. The singularity they proved means nothing in physics, because the equations are only the continuum limit. If you find an analytical solution with a singularity, it means that at that point something more interesting is happening at the molecular level. You just can't have a singularity in reality. Effectively, a gas works like a continuum, but it actually isn't one.

Discussion

The discussion on the day is folded into the sections above, at the points where each question was asked. Thanks to everyone in the room and in the chat for them.

References

"A Counterexample to the Jacobian Conjecture." 2026. Preprint, Ulam, July 20. https://www.ulam.ai/research/jacobian.pdf.

Anandkumar, Anima. 2026. "Stable Singularity of the Euler Equations on $\mathbb{R}^3$." *What's New*, September 10. https://terrytao.wordpress.com/2026/09/10/stable-singularity-of-the-euler-equations-on-r3/.

Avila, Artur, Manjul Bhargava, Caucher Birkar, et al. 2026. "A Severe Misalignment of AI in Mathematics." *What's New*, September 11. https://terrytao.wordpress.com/2026/09/11/a-severe-misalignment-of-ai-in-mathematics/.

Baumol, William J., and William G. Bowen. 1966. *Performing Arts: The Economic Dilemma; a Study of Problems Common to Theater, Opera, Music and Dance*. Twentieth Century Fund.

Brown, Noam. 2026. “Noam Brown – Agent Swarms, Alignment, & Recursive Self-Improvement.” Interview by Dwarkesh Patel. *Dwarkesh Podcast*, September 17. https://www.dwarkesh.com/p/noam-brown.

Buckmaster, Tristan. 2026a. “Finite-time blowup with smooth forcing for incompressible porous media, for Boussinesq, and for 3d incompressible Euler.” Mastodon post. Mastodon, September 8. https://mastodon.social/@tristanbuckmaster/117233413705701198.

Buckmaster, Tristan. 2026b. “Statement.” September 8. https://cims.nyu.edu/~tristanb/statement.pdf.

Carlini, Nicholas, Newton Cheng, Keane Lucas, et al. 2026. “Assessing Claude Mythos Preview’s Cybersecurity Capabilities.” Anthropic, April 7. https://www.anthropic.com/news/mythos-preview.

Cohn, Henry. 2026. “The Technical Debt of AI-Generated Mathematics.” *What’s New*, September 15. https://terrytao.wordpress.com/2026/09/15/the-technical-debt-of-ai-generated-mathematics/.

Cook, John D. 2026. “Locally Everywhere Does Not Imply Everywhere.” *John D. Cook | Applied Mathematics Consulting*, July 21. https://www.johndcook.com/blog/2026/07/21/jacobian-conjecture/.

De Toffoli, Silvia, and Eamon Duede. 2026. “After Math.” *What’s New*, September 12. https://terrytao.wordpress.com/2026/09/12/after-math/.

Goldratt, Eliyahu M., and Jeff Cox. 1984. *The Goal: Excellence in Manufacturing*. 1st ed. North River Press.

Gowers, Timothy. 2026. “Why I Didn’t Sign the Fields Medallists’ Letter.” *Gowers’s Weblog*, September 17. https://gowers.wordpress.com/2026/09/17/why-i-didnt-sign-the-fields-medallists-letter/.

Gowers, W. T. 2000. “The Two Cultures of Mathematics.” In *Mathematics: Frontiers and Perspectives*, edited by V. I. Arnold, Michael Atiyah, Peter Lax, and Barry Mazur. American Mathematical Society. https://www.dpmms.cam.ac.uk/~wtg10/2cultures.pdf.

Hales, Thomas, Mark Adams, Gertrud Bauer, et al. 2017. “A Formal Proof of the Kepler Conjecture.” *Forum of Mathematics, Pi* 5: e2. https://doi.org/10.1017/fmp.2017.1.

Hartnett, Kevin. 2026. *The Proof in the Code: How a Truth Machine Is Transforming Math and AI*. First. Quanta Books / Farrar, Straus and Giroux.

Kra, Bryna. 2026. “‘Deep Theorems Were Scarce and Difficult and so Became an Effective Mechanism to Identify Deep Thought. AI Has Broken This System.’” *What’s New*, September 13. https://terrytao.wordpress.com/2026/09/13/deep-theorems-were-scarce-and-difficult-and-so-became-an-effective-mechanism-to-identify-deep-thought-ai-has-broken-this-system/.

Madhavapeddy, Anil. 2026. “Just a Rumour of a Bug Is Enough to Find a Security Exploit These Days.” *Front Matter*, August 22. https://doi.org/10.59350/tngsm-6rx23.

Naur, Peter. 1985. “Programming as Theory Building.” *Microprocessing and Microprogramming* 15 (5): 253–61. https://doi.org/10.1016/0165-6074(85)90032-8.

OpenAI. 2026. “On the Navier–Stokes Millennium Prize Problem.” OpenAI, September 8. https://openai.com/index/navier-stokes-solution/.

Scholze, Peter. 2020. “Liquid Tensor Experiment.” *Xena*, December 5. https://xenaproject.wordpress.com/2020/12/05/liquid-tensor-experiment/.

Stenberg, Daniel. 2026. *The Pressure*. May 26. https://daniel.haxx.se/blog/2026/05/26/the-pressure/.

Sumner, Jarred. 2026. “Rewriting Bun in Rust.” *Bun Blog*, July 8. https://bun.com/blog/bun-in-rust.

Tao, Terence. 2026a. “A Digestion of the Jacobian Conjecture Counterexample.” *What’s New*, July 21. https://terrytao.wordpress.com/2026/07/21/a-digestion-of-the-jacobian-conjecture-counterexample/.

Tao, Terence. 2026b. “Palomar – a Registry of Lean Verified Mathematics.” *What’s New*, August 18. https://terrytao.wordpress.com/2026/08/18/palomar-a-registry-of-lean-verified-mathematics/.

Thurston, William P. 1994. “On Proof and Progress in Mathematics.” *Bulletin of the American Mathematical Society* 30 (2): 161–77. https://doi.org/10.1090/S0273-0979-1994-00502-6.

Thurston, William P. 2010. “Answer to ‘What’s a mathematician to do?’“ MathOverflow, October 30. https://mathoverflow.net/questions/43690/whats-a-mathematician-to-do.

Willison, Simon. 2026a. “On the Navier–Stokes Millennium Prize Problem.” Simon Willison’s Weblog, September 8. https://simonwillison.net/2026/Sep/8/on-navier-stokes/.

Willison, Simon. 2026b. “Sqlite AGENTS.md.” *Simon Willison’s Weblog*, May 27. https://simonwillison.net/2026/May/27/sqlite-agents/.