What goes into Formalizing Fermat?

Kevin Buzzard, Imperial College London

AITPM, Exeter, 8th April 2026

Before I start

Many thanks for the invitation to speak at this workshop!

I'll talk about the current state of my plan to get Fermat's Last Theorem formalized in Lean.

I'll talk about what we've done, what we're doing, and what we're going to do in the near future.

And then I'll talk about autoformalization and how this might help further along the line.

Fermat's Last theorem

Fermat's Last Theorem (FLT) is the statement that if a, b, c, n are positive integers with $n \geq 3$, then $a^n + b^n \neq c^n$.

It was proved in 1993 by Wiles and Taylor–Wiles, having been open for over 350 years.

The theorem as it stands is useless – it has no applications.

But attempts to understand the problem have been behind the development of a lot of number theory.

(some of which has been very useful.)

The proof of FLT, when written out in full, is several thousand pages long.

The proof uses lots of technical machinery, for example the theory of automorphic forms.

Automorphic forms are the key ingredients of the Langlands Program, which is still a hot topic in mathematics.

Fermat's Last Theorem

An interactive theorem prover, or computer proof assistant, is a computer programming language which is expressive enough to understand the concept of theorems and proofs.

Lean is an interactive theorem prover. It knows the axioms of mathematics and the rules of logic.

Since 2017 I have been talking about the idea of teaching Lean a proof of Fermat's Last Theorem.

(I am a student of Taylor and a grand-student of Wiles, and I was a PhD student working in the area when the proof was announced).

In 2024 I got an EPSRC grant to work on the problem.

Before I start on the talk, I want to say a little bit about why humanity might want to formalizing Fermat.

Why formalize Fermat?

Why formalize Fermat?

Everyone knows the proof is fine.

For me, there were four main reasons.

It's a test to see what systems like Lean can do.

It's a way of teaching Lean about the Langlands Program (the central questions in this program are still open).

It's a chance to make the proof interactive and explorable.

And finally, part of me just wants to check that everything is OK in the proof.

(Don't take this last reason too seriously – everything seems fine so far)

How to formalize Fermat?

How does one teach thousands of pages of technical mathematics to a computer?

This is quite a complicated question!

One has to have a system which is able to easily manipulate the complex objects that go into the proof.

I believe that Lean is that system.

But one also has to decide where to start.

Code you write at the beginning needs to be maintained until the end.

And maintaining code in a system which is changing underneath you can sometimes be hard work.

Right now (2026), Lean and its mathematics library `Mathlib` make no promises of backwards compatibility.

This is essential, because core developers are still learning how to make the system better.

Reduction to the 1980s

I wrote the grant proposal in 2023, when formalizing 2000 pages of technical mathematics sounded like a very difficult problem.

It was not impossible, it would just clearly take a *very long time*.

Typical formalization projects up until that point would be of at most a few hundred pages of informal mathematics, and would have multiple authors.

My 2023 proposal was hence *not* to formalize a complete proof of FLT.

My proposal was to *reduce* the proof of FLT to the 1980s.

One way of thinking about it:

The Wiles and Taylor–Wiles papers (from the early 1990s) are around 150 pages, but have many references.

So it's like formalizing the Wiles and Taylor–Wiles papers, but not formalizing the references.

(Actually I will be formalizing a more modern proof, following ideas of Khare and Wintenberger, and other 21st century developments.)

2025

I spent most of 2025 collaborating on a project to define a certain space of automorphic forms and to show that it was finite-dimensional.

This was joint work with Salvatore Mercuri, Matthew Jasper, Maddy Crim, William Coram, Yael Dillies. . . .

The proof of finite-dimensionality was known in the 1960s so I did not logically need to do it.

However, I felt that it was important that some of the key players in the proof made it into `Mathlib`.

Status: the theorem is sorry-free, the paper is being written, key definitions and results are being upstreamed.

2026

In 2026 I have been focussing on writing down (and reminding myself of) a detailed proof of an explicit statement of the form:

(this precise list of theorems known in the 1980s) implies FLT.

I have been giving lectures on the argument at Imperial College as part of the EPSRC Taught Course Centre.

I didn't see anything which particularly worried me.

In the course, I reduced FLT to statements from the 1980s, plus a modularity lifting theorem.

I will give three technical lectures next term where I prove the modularity lifting theorem informally.

Then either me or my PhD student Andrew Yang will start on the formal proof (Yang has already made some progress).

Mid-2026

After that, my plan is to create a “top-down blueprint.”

Let me something about blueprints, for those who haven’t seen them.

Here’s the polynomial Freiman–Ruzsa blueprint graph.

Top-down thoughts

Because my plan is to *reduce* FLT to older theorems, it feels strange to me to have the proof of FLT at the end of the work.

So in my blueprint it's at the start.

And the work begins by reducing FLT to two simpler statements.

Here is the first chapter of the blueprint.

It tells a short story.

The next chapter tells a different story.

Future blueprint

Due to current limitations in the blueprint software, these two stories are currently unrelated.

My vision is that one should be able to open and close nodes to unfold the proof.

Note also that the two blue nodes in the first graph are of two completely different natures.

One is Wiles' theorem, which needs to be proved.

The other is Mazur's theorem, which was proved in 1978 and the plan is to assume it.

Looking forward

By the end of 2026, I envision a nice blueprint, clearly explaining how to reduce FLT to the 1980s.

I envision `Mathlib` being better at understanding what's going on in the Langlands program.

And I envision there being a fair amount of formalization work to do.

Unlike the bottom-up work in 2025, this stuff is quite technical.

Probably you will have to be something like a PhD student or above in number theory in order to contribute.

Or could you be a machine?

I want to spend the rest of the talk discussing this idea.

Autoformalization

I wrote the grant application in 2023 when the world was a very different place.

ChatGPT was beginning to write Python code.

But there were billions of lines of Python code on the internet, and only millions of lines of Lean code.

Also, there was no “Rosetta stone” containing many pairs of natural language statements paired with Lean translations.

Autoformalization is the process of getting a machine to translate from natural language into Lean.

In 2023, it seemed that key ingredients were missing and that autoformalization was a difficult problem.

Autoformalization in 2026

But things move on, and despite there still only being millions of lines of Lean code, and no Rosetta stone, autoformalization is here.

Examples of what happened in March:

Math Inc autoformalized the Cohn–Kumar–Miller–Radchenko–Viazovska proof of 24d sphere packing.

This (and 8d sphere packing) was the result which won Viazovska the 2022 Fields Medal.

The formalization apparently took one week and was 200K lines of code.

Note that Mathlib is only 2.25M lines of code, and took 8.5 years to write.

Autoformalization in 2026

Also in March the Archon group, based in Beijing, formalized AI-generated output of two research-level lemmas (Q4 and Q6 from First Proof), writing 12K lines of code in 4 days at a cost of $2000.

So that's $1000 per research level lemma.

Fabian Gloeckle and Meta formalized Darij Grinberg's 700 page textbook on algebraic combinatorics, using 13K Claude Opus agents to write 130K lines of Lean code in a week at a cost of $100K.

This latter work is around $200 per page, or $300 per graduate level theorem.

For those who are thinking that this is a lot of money for a graduate level theorem – this would take a *huge* amount of time to formalize manually.

I have informally heard about several other autoformalization projects.

What about autoformalization in 2028?

So far I have claimed:

A full proof of FLT is about 2000–3000 pages.

Meta formalized 700 pages of mathematics in one week.

So how about this:

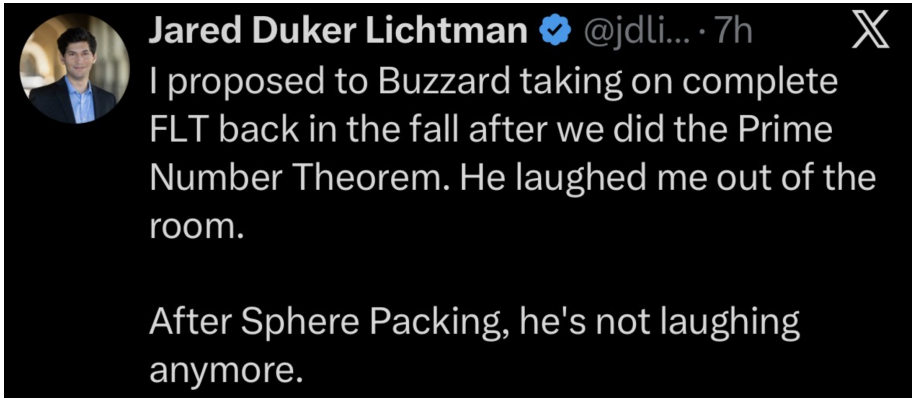
I go off on a round-the-world cruise with the EPSRC money.

I come back in 2028 and then just use available tools to formalize all of FLT in a week.

I claim victory, having delivered far more than I promised to EPSRC, and everyone is happy.

Sounds good!

Math Inc's take on the matter



Despite this tweet from a Math Inc employee, I never stopped laughing.

I reserve the right to stop laughing later, but not just yet.

Let me try and explain why.

What can autoformalization *do*?

Autoformalization now, like formalization five years ago, is picking its targets wisely.

Companies need to demonstrate that that the software works.

So they choose projects which look impressive, and which work.

But these projects are actually very carefully-chosen.

Currently it is *not* the case that we can formalize an *arbitrary* 700 page textbook.

Why not?

Definitions

The issue is definitions.

Formally, all definitions are equal.

They are just some computer code which defines a type or a term.

But to the mathlib community, definitions cover a *huge* spectrum.

For example the definition of the natural numbers is just a couple of lines long.

But the definition of a Riemannian manifold took many months of thought, experiment, and PR review.

And, of course, it relied on many other definitions which also needed to be carefully constructed.

Formalization as publicity stunt

Historically, the formalization community has been very clever.

It has chosen targets which look impressive but which are accessible with known tools.

This has happened from the beginning of this century, when people were manually formalizing things like the odd order theorem, the Kepler conjecture, the four colour theorem and so on.

These are all famous and long proofs and there are good reasons for formalizing them.

But they are essentially elementary proofs about elementary objects (finite groups, spheres, graphs).

These are theorems whose *statements* you can explain to a first year undergraduate.

Because ten years ago, that was what the community and the tools could do.

Formalization as publicity stunt

Later on, when the tools became more powerful, people started proving theorems about Ext groups in the category of liquid vector spaces.

These are complex definitions which are typically covered in a graduate level course.

Now it feels to me like the community could manually formalize almost any piece of known mathematics, given enough time.

But for *autoformalization*, I am not so sure.

Let's look back at what was autoformalized in the last few months.

Autoformalization successes

Prime number theorem (Math Inc's first autoformalization project): elementary definitions, could explain the statement to a schoolchild.

First Proof questions 4,6,9 (the three which were autoformalized) are elementary to state (but tricky to prove!) questions about polynomials, graphs and matrices.

Sphere packing: as long as you are happy with measure on $\mathbb{R}^{24}$ (which is in mathlib) the statement is elementary. The Leech Lattice needs to be defined, but that's just "the $\mathbb{Z}$-span of these 24 explicit vectors in $\mathbb{R}^{24}$."

Grinberg algebraic combinatorics: again the definitions involved only involve one or two lines of Lean.

So in all cases, every definition involved (in the statements *and* the proofs) was either (a) in mathlib already or (b) easy to define in an idiomatic way.

Fermat's Last Theorem is not like that.

Definitions

Some examples of definitions needed in the full proof of FLT:

Automorphic forms for GL_2 and its inner forms.

Shimura varieties and their canonical models.

L^2 cohomology.

The construction of the complex manifold associated to a smooth scheme over $\mathbb{C}$.

None of these are in Mathlib.

But furthermore it would be *hard* to get these results into mathlib.

Right now, these results need very careful planning.

So why not plan?

So why am I not dropping everything and formalizing these definitions?

Because almost all of the applications of these constructions are to prove theorems which were known in the 1980s.

So these definitions (which would take a substantial amount of time) are not needed for my project of reducing FLT to the 1980s.

They *would* be needed for Math Inc's plan to formalize all of FLT.

But they would also be a huge distraction to my plans.

I need to formalize plenty of subtle definitions just to reduce FLT to the 1980s.

My challenges to the autoformalization community

To those people who think that autoformalization will completely solve FLT, I have two challenges for you.

The first challenge: autoformalize Mazur's theorem.

```
example (E : WeierstrassCurve ℚ) [E.IsElliptic] :
    (AddCommGroup.torsion E(ℚ) : Set E(ℚ)).ncard ≤ 16
```

The proof is a 150 page paper from 1977.

Mathlib has everything for statement of the theorem, but the proof needs integral models of modular curves.

(This is a baby special case of integral canonical models of Shimura varieties)

All known proofs of FLT need Mazur's theorem.

Second challenge

The second challenge is a definition.

All known proofs of FLT need moduli spaces of PEL abelian varieties.

So the challenge is to define these moduli spaces, as algebraic varieties over a number field, with their universal abelian variety.

Although this is a definition, it will be extremely hard-won.

Any proof of FLT will need to prove hard theorems about these objects.

It is not clear to me that autoformalization is ready for this.

But, of course, autoformalization is going to get better.

It remains to see how much better.

Hilbert modular forms

To test how good AI was at formalisation of definitions I tried an example.

A definition I will need for the proof of FLT I'm formalizing is Hilbert modular forms.

I asked an AI to formalize the definition and it got it wrong.

I complained and the second definition was less wrong.

The third definition was nearly right.

By this point I was thinking that it would just be easier to do it manually myself.

An interesting question (which Bin Dong asked me yesterday): how to make a Claude skill (for example) which plays the role that I was playing there.

Summary

FLT is ticking along nicely.

I am still confident that by 2029 we will have reduced it to statements from the 1980s.

I currently cannot really see a path towards formalizing all of it by 2029.

AI optimists will of course tell you that it will all be done within a year.

Currently, complex and missing definitions are a serious bottleneck.

AI cannot autoformalize these things yet.

Things will change but I am not clear when they will change.

Until then, we are formalizing definitions manually.

Thanks for listening!