Hello, everyone. Welcome to another Q&A about science and technology for kids and others.
Let's see, I have a whole bunch of… Questions?
Saved up here. Some slightly elaborate ones here.
First question from Godels is, is there an actual use case for AI math proofs?
would… You, me, ever actually find them useful in my work?
Well, let's see, let's talk a little bit about proofs, then we'll talk about proofs in math.
Then we'll talk a bit about AI,
AI doing proofs, and so on. And then we'll talk about places where I at least, might find them useful.
So… What's the idea of a proof?
I think, sort of, the notion of proofs kind of originated with old, Euclid back, in, maybe 2000… a bit more than 2,000 years ago.
Where people were saying.
you know, I have this geometrical, I have something where I'm drawing a circle, I draw another circle that intersects with it, and I draw lines from here to there, and it's like, is that line… is that angle between two lines a right angle, for example?
And you could draw a picture of the circles intersecting and so on. You could kind of measure, yeah, it seems like it's more or less a right angle. But the question was, can you actually prove that it's a right angle? Can you give, essentially, a logical, a structured argument
That says it must be a right angle, not just… it seems to be roughly a right angle.
And how do you make a proof? Well, you have to start with certain assumptions, certain axioms.
So, for example, in the geometry of Euclid, a famous axiom was that two parallel lines never cross.
That axiom is a reasonable thing to assume if you're doing, you know, geometry in a plane.
It turns out not to be true about physical space.
the discovery 100 years ago that space is curved, it's curved by the presence of mass and so on, that's what leads to gravity. That curvature in space makes Euclid's axiom that two parallel lines never cross not be true for physical space. But you can still, if you're just doing theoretical math.
you can still say, we're just going to take it as an assumption, as an axiom, that two parallel lines will love across. So, in Euclidean geometry, for example, there are about 10 axioms like that.
like, there are axioms about if there's… if… well, for example, another famous one is, if A is equal to B,
and B is equal to C,
then A is equal to C.
Now, that will be true for some meanings of equal, but for example, if you substitute for equal is a friend of, if A is a friend of B, B is a friend of C, in real life, it absolutely does not follow that A is a friend of C.
But… but it is assumed, for the notion of things being equal in a mathematical sense, that that so-called transitivity is true.
So, in any case, you start in making a proof
from a set of axioms, a set of assumptions about, sort of, abstract assumptions about things.
Those assumptions may or may not correspond to what is true in the real world if you identify these parallel lines with actual lines and space and things like this.
But it's something, it's a set of, kind of, abstract assumptions that one makes.

And then you have to say, well, what can we prove from those abstract assumptions? Take this assumption, and that one, this axiom, that axiom, you combine them together, you get another thing, and so on.
So, the, a proof
is the sequence of steps that tell you how you get… how you sort of assemble these axioms together. It's kind of like you're putting together a puzzle, where these axioms are kind of pieces of a certain shape, and you're saying, can I assemble these shapes to make this final thing that I want?
And that's kind of the idea of a proof. You're trying to prove a theorem, which is… has the same structure as an axiom, actually. You're trying to just prove a statement
By putting together these puzzle pieces that start with the axioms, you put together the puzzle pieces to make the thing that you want to prove.
And then the proof is that sort of puzzle that you made that says, this thing follows from that, follows from that, etc, etc, etc.
Okay, so
In the history of mathematics, for example, people have published a few million proofs of all sorts of different things.
from the Pythagoras' theorem about triangles, you know, A squared plus B squared equals C squared, for the lengths of the sides of a right triangle, those kinds of things, to things like Fermat's last theorem, X to the n plus y to the n equals Z to the n, for… that's… can never be satisfied for integers X, Y, and Z, an N greater than 2,
That's Fermat's Last Theorem, proved about 20 years ago now, after several hundred years of sort of people thinking it must be true, but not really having a proof.
something like… well, the Pythagorean theorem, there are many ways to prove that. It's fairly straightforward to prove from the axioms of Euclidean geometry. Fermat's last theorem turned out to be really hard to prove. It took, you know, hundreds of pages of proof, and took hundreds of years to get to that proof.
So, and one of the features of math is that there are things that are provable, there are theorems that are provable, but the proofs that at least we know for those theorems are very long.
Do we know whether we have the shortest proof of those theorems? That's a really hard thing to know, and we typically don't have any idea whether we have the shortest proof. What we try to do in math is to find some proof of that result, and that's… that's what sort of satisfies us.
We can kind of think of sort of theorems or axioms, they're the same kind of thing, as being sort of dots in some big graph, and the graph has, you know, a dot, a node for each theorem or axiom, and then it's sort of joined
to theorems which are consequences of the theorem you already got. So, a proof then becomes this path from theorem to theorem to theorem. You're sort of establishing from the thing you started with, you're establishing the theorem that you get out in the end.
And… so… Okay, in… in doing math.
For most of the history of mathematics, people would find proofs by hand.
Then, starting actually in the 1950s, there started to be automated proof generation techniques.
So if we think about a proof as being, you know, can you fit these theorems and axioms and so on together in such a way as to get the thing you want, it becomes a problem that is very much kind of like a… it's sort of a, you try many possibilities, kind of a combinatorics-type problem.
It's like you're literally trying puzzle pieces, and seeing whether they fit.
And that's the thing you can get a computer to do.

It turns out that one of the big tricks in automated theorem proving is you know where you start from, you have those initial theorems or initial axioms, you know where you're going, and one thing you might try to do is to follow all the different paths from where you start from and see whether any of those paths land up in the place you're going.
That turns out to be not as efficient as the thing that actually happens, which is you start both from the thing you… the theorems you're starting from, and you go backwards.
from the thing you're trying to prove in the end, you somehow try and get those paths to meet in the middle. And that's, as a practical matter, how theorem-proving software programs work.
So… then the question is, well, alright, I should say another thing about proofs done by mathematicians. So.
One of the things that one hopes for, and a proof
Done by mathematicians, is that the proof will somehow explain why the result is true.
It will be… the first thing you get from a proof is to know, yes, that result is true, but the second thing is, why is it true?
What kind of mathematical kind of structures are involved in proving that? What kind of… what… give me an explanation for why it's true.
In the case of Fermat's Last Theorem, it went into incredibly elaborate areas of mathematics about geometry and algebra and all kinds of things to come back to that result that is quite easy to state.
But the fact that it went through all of this, kind of, all these sort of side roads of mathematics is sort of significant in understanding the kind of the nature of why that result is true.
as I say, we don't actually know what the shortest proof is, but we know that this is a proof where we can understand why the result is true.
And a lot of what one is trying to do in mathematics, if one is a sort of professional pure mathematician, is understanding, kind of, mathematical structures. It's much less about, kind of, the precise details that the puzzle pieces fit together. It's more about, as we try and fit the puzzle pieces together, what do we learn about new abstract concepts that we have to introduce?
It's very common that even when you have a theorem that seems quite straightforward, like Fermat's last theorem, you know, X to the n plus y to the n is never equal to Z to the n.
Etc. That even though that's really simple, it's very elaborate and complicated mathematics that you need to, to deal with.
to get to that result. I mean, a classic example of this from 1500,
was the result on cubic equations. So a cubic equation, it's an equation of the form something like X cubed plus, you know, 7X squared plus 5x plus 9 equals 0, for example. It's a thing that has an X cubed, x times X times X term in it, and the goal is
for what values of X
Is that equation true? Is it really true that x cubed plus blah blah blah equals zero?
And, that was figured out back in 1500,
And one of the big surprises was that in the midst of figuring it out, it's a very complicated algebraic expression that goes many lines long, that is the general solution to a cubic equation for any coefficients, you know, AX cubed plus bx squared plus CX plus D equals zero.
for all A, B, C, and D, you can write down a formula for what the solution is, but it's a very complicated formula.
One weird thing about that formula is it's a formula that has this thing, plus that thing, plus square root of this, plus that, and so on.

when you look at that formula, the individual pieces of that formula, for reasons one doesn't really understand, I think, even really, in a sense today, the individual pieces of that formula, even if the result is going to be an ordinary number, like, oh, I don't know.
seven and a half or something. Even if it's going to be some ordering number at the end.
the terms in that formula involve so-called complex numbers, things like square root of minus 1 and so on. And that was a… it was sort of necessary in the intermediate stages to introduce something which is sort of far away from what you were originally talking about. We're just saying what value of X
will satisfy that equation. Even if there is a value of X that's a real number in the sense it doesn't have any square roots of minus 1 and so on involved, then you still have to sort of go out of that domain into this more complicated domain. But that's sort of a typical thing that happens in mathematical proofs.
And it's very… it's very rich and important for pure mathematics, because it's kind of telling one there are sort of these other mathematical concepts. It's pulling in other mathematical concepts. It's kind of a whole mathematical narrative that explains why something is true.
Okay, so that's the traditional theory, sort of story of proofs in mathematics. There are a few million of them that have been written down in the history of human mathematics and so on.
Okay, so then enter computers.
So, like in Wolfen language, there's a function called FindEquationalProof.
Where you give it a bunch of axioms. The axioms might be things like, you know, I don't know, X dot Y equals y dot X for any X and Y.
Or it might be some other thing like that.
and you say, given these axioms, can you prove this result? You know, X dot Y dot X dot z equals z dot Y dot whatever it is, I don't know. Can you prove that? Okay, and find the equation of proof will go off and start trying to figure out all these paths and so on, and eventually it might say, yup, I've got a proof.
What does that proof look like? It's a proof that says, well, you take this expression, you substitute this one in, and you substitute this one. Each step in the proof is very straightforward. although easy to apply by computer, it's actually, you know, if you try and do them by hand, they're just very tricky. There's nothing difficult, there are no concepts that get pulled in. It's conceptually simple, but technically, but just the details are difficult. And so.
that, that ends up being… so anyway, you get these proofs that are these sort of very down-in-the-details type proofs, although they're conceptually straightforward. In traditional proofs made by human mathematicians, they tend to be not
not conceptually straightforward, they tend to be conceptually sophisticated and rich, and that's part of the point, and maybe not so technically and mechanically difficult. Computer proofs are technically and mechanically difficult, but not… don't have any new concepts in them.
So, when it comes to, sort of, finding proofs like that, there's been sort of a lot of work done on this, but the vast majority of proofs that people have ever, sort of, set up with automated theorem proving are proofs of things that people already thought were true.
There's just a sole example, to my knowledge, where that didn't happen, which was something I did in the year 2000. I was trying to look for, well, the simplest axiom for logic.
And I just did a big search of axioms, and to figure out whether one of these axioms was, in fact, an axiom for logic.

I had to use automated theorem proving. I had to try and actually do the automated proof, and eventually I found one, an axiom, which is the very simplest axiom that there can exist for logic, and I proved that it's correct using automated theorem proving.
Well, that proof is really long and complicated. It's maybe 120 steps of substitute this for this, and this for this, etc, etc, etc. If you unroll it completely and don't use, sort of, any intermediate lemurs where you've proved this and then you use what you proved again, if you unroll it completely, it's like 80,000 steps.
Well, no human has ever understood this proof.
It's a proof of a very different character from the proofs that are typically done by human pure mathematicians, where, sort of, it's very common to introduce conceptual complexity, but not technical complexity. In this proof, there's technical complexity and no conceptual complexity.
And what's the significance of the proof?
Well, the result it proves is pretty interesting. It's nice to know that that's true. What do we learn from the proof?
Not really a lot. We've never been able to learn much from that proof.
Now, maybe there is a way of interpreting that proof, or a different version of the proof, where there is a new conceptual idea that can be pulled in. We haven't figured that out.
And yes, I've tried using AIs to try and figure out pieces of that, they've not been successful. It's kind of something too down in the weeds to engage with typical LLM-type AIs, where, you know, the AI has been trained on millions of math papers, and it knows kind of what the rhythm of how things work in math papers is, but when it's confronted with this kind of write-down in the
weeds kind of mechanical structure, it's like, I've never seen something like this before.
So, okay, there's… there's one more sort of game in town. So there's automated theorem proving, where technical complexity, not conceptual complexity. One more game in town, which is so-called proof assistance.
And, there are a bunch of these. One that's pretty popular right now is called Lean. The idea of a proof assistant is
You… it's… in the initial situation, it was like, it's going to define puzzle pieces, and it's not going to solve the puzzle for you, but if you put down these puzzle pieces in a certain way, it's going to tell you, yup, those pieces fit, or no, those pieces don't fit.
So proof assistance, the sort of the idea originally was humans would construct proofs
And then the proof assistant would validate that the proof was actually true, because the individual steps that the proof assistant has to go through are very straightforward computational steps, so it's easy to know that, yes, those sort of puzzle pieces did actually fit together.
Okay, so people started using proof assistants, oh, I don't know, 25 years ago, maybe, to… I mean, the first ones were talked about in the 1960s, but they started really getting used, and not much, maybe 25 years ago, and it wasn't a big success.
I mean, I had a friend who proved a very interesting theorem, and he did it by hand, actually, well, using, actually, our Wolfram language technology, and
You know, he wrote this big paper, hundreds of pages of long, about his proof.
And people in the mathematical community were like, this proof is hundreds of pages long, how can we know it's right? And we don't really believe it, because it was a proof that had been sort of waiting to be done since the 1600s, so kind of a big deal.

And so he spent a decade, turning that proof into something that could be sort of rigorously followed using a proof assistant, where every step was fully formalized, and you could kind of just see, you can unroll it, and you could say, yep, we can tell that this proof is correct.
Well, the problem is, nobody cared.
It's, it's like, everybody was like, yeah, that's nice.
You know.
well done spending a decade doing this, but… but it doesn't really matter. We, you know, we believe the proof anyway.
Okay? So, that was kind of a typical, experience with… with things like proof assistance, was… that that would happen. Now, the other feature of proof assistance is that they, to some extent, for some kinds of very
kind of abstract proofs, they, to some extent, help people to sort of organize their proving process. I don't know… I never did it myself that way, so I'm not sure. I mean, for me, using Wolfram language, which is primarily about computation rather than about proof.
for me, sort of organizing things in Waltham language is critical to sort of understanding how things fit together.
I think proof assistants had an attempt to do the same kind of thing in a different direction, but sort of pure mathematics. They were very tied up with category theory, which is… and type theory.
which are two kind of idealizations of mathematics. I mean, usually in mathematics, you might talk about algebra, you know, X squareds and so on, or geometry, circles, and things like this. But you can ask, kind of, what's a higher level representation of mathematics? What's a way of, sort of, talking at a meta-level about mathematics?
And one way to do that is to talk about categories, where you just say, well, there's these things, these objects, and there are these morphisms that transform objects. You're not really saying yet what the objects are, and that's kind of a higher level way of thinking about mathematics. And then there are results that you can get that are sort of true for all categories of a certain type. That… where you don't need to know whether it's talking about triangles or squares or algebra, or whatever else.
And so there's a certain set of results that can be got that way. It's a way of kind of abstracting above mathematics, organizing mathematics.
Type theory is the same kind of thing, where you're dealing with what type of a thing is this? Is it a whole number? Is it a…
fraction? Is it an algebraic expression? Is it a circle? What type is it? You can build up very elaborate types, which are… it's this which has such that this, such that that, and so on. And you can kind of, if you want to, you can state theorems
in terms of types. So a very common thing to do is to say, I'm going to have this type of thing that's a whole number, that's a whole number which
when you divide it by any other whole number, never gives, etc, etc, etc, you have a whole bunch of conditions. And then the question is, is that type inhabited, as it's called?
Is there anything that you can imagine that satisfies the conditions to be one of those types? And that's a way of stating certain kinds of theorems, basically stating theorems of the form, does there exist a blah blah blah? Is this type, this thing that's defined by these constraints inhabited?
So anyway, type theory, category theory, two things which sort of intersected with kind of the… at least the, the kind of the social trend of proof assistance. Never caught on in mainstream

mathematics, to be clear. Very much of a… a thing, if you're sort of doing the formal reaches of mathematics, something to think about.
Okay, so a new game in town is saying, can you use an LLM AI to…
Go from what's written in a math paper to auto-formalize, to automatically produce a proof-assistant result.
And so there's been a fair amount of work, there's a bunch of startup companies that are trying to do this.
And, like, so what can this do? Well, one thing you could say is, well, gosh, you know, if you formalize the proof, you know it's right.
Good idea, not so good in practice.
Because…
When you formalize something, you have to know that your formalization actually relates to how the thing you're trying to formalize. So, for example, Euclid was completely convinced, I'm sure, that he was formalizing how geometry has to work by giving his fifth postulate that two parallel lines will never meet.
Turns out, the actuality of how space works, that doesn't… isn't actually true.
So you have to be careful when you formalize things. The thing that you took, which was your idea in your mind, or the thing you wrote down in English in your paper or whatever, did you actually formalize it correctly? Is that thing that you wrote in your proof assistant code actually the thing that is the thing you were thinking about?
So, that's, that's a sort of a gap there, and it's an important gap I'll talk about in a minute.
But… so there's been a certain amount of work, and there's a big effort, actually, to formalize Fermat's Last Theorem using a proof of systems. What will we learn from that formalization? I'm not really sure.
you know, we… will we learn that the theorem is true? Maybe.
I say maybe because it depends on what axioms you put in at the beginning.
And, for example, when… when Ferwett's Law's Theorem was first proved by a chap called Andrew Wiles back in the… when was it, 1990s, I saw him actually
Soon after that, and I sort of said, well, so what axioms do you think you use to prove this theorem?
And he said, well, I don't really know, and he thought about it a bit. He said, but I don't really care. That's not the point. Doing this very rigorous thing of going from these formal, abstract axioms to the final proof is not the main point. The main point is kind of the texture and concept of what's going on.
So…
In any case, I suppose we'll learn from the formalization of Fermat's last theorem. Maybe we'll learn what axioms you can use. I don't know, because as soon as you say, well, you know, if we use very, very low-level axioms, the simplest axioms of arithmetic, for example, then it'll be a heavier lift to get to the theorem, and maybe it won't even make it. If we start introducing axioms from fields like set theory, those are axioms that no human has real intuition about.
out, I think.
They're axioms that are very formalistic axioms about infinite sets and so on, and it's just like, well, it's the tradition to assume that this is the way things… the way you should set up the axioms.
But, so, I'm not sure exactly what's going to be achieved there. I mean, there'll be a certain amount learned about, sort of, the engineering of how to do such a proof.

But, and I think… so… so there's kind of a… a strange thing, and I think also, I mean, I've, myself, have looked at formalization of math
for lots of purposes, for… well, I… obviously, Waltham Language and Mathematica are the world's largest, sort of, formalization of computational mathematics, of mathematics where you can actually compute an answer, where it's not a question of doing a proof where you say, I think this is equal to this or something, now prove it.
what we're doing is to say, given this sort of question, grind through and get to the answer. It's a somewhat different sort of workflow that turns out to be much more valuable in most practical cases.
But in any case, the,
It's… it's kind of one of these things where I've certainly thought about formalization of mathematics, organized a bunch of things with mathematicians who mostly have said, we just don't really care about this very much. And… but then…
It's… it's something where there isn't really a commercial market for that kind of formalization. Well, there is one. It's a different kind of one.
Which is, when you build computer systems, and when you build things like networking systems and so on, you want to know, is this thing correct? Is it… is it going to fail?
For example, if you're making a cryptocurrency, you probably want it to be… you probably want to be fairly certain that people can't spend the same coin twice.
And that's the thing which the cryptocurrency is set up to work according to, you know, code and the operation of the virtual machine that is operating the cryptocurrency. And there's a question of, can you prove
that you can't spend a coin twice, for example, spend the same coin twice. So that's a place where you might want to do this kind of, you know, automatically prove a theorem, or even make a proof assessment where you can say, I'm going to construct a proof, can you confirm that it's true?
Well, actually, when it comes to programs, it's really hard to make proofs.
Most programs that one cares about in practice, it's… it's inconceivably difficult to make a proof.
One of the results that's a fundamental result about mathematics and about computation, it's a result that comes out of Godel's theorem from 1931, is that there are arbitrarily long proofs.
there may be things where you say, is this going to be true? Is it going to be true? And you keep going, you keep going, you make a longer and longer proof, you've got a million-step proof, you still haven't quite managed it. In the million and one step, is it going to finally say, yup, it's true, or will it wander forever and never, in fact, be provable?
So, what Godel's theorem and undecidability and my sort of favorite phenomenon of computational irreducibility, they all point in the same direction, which is you can never know how long the proof will be in advance. You can never say, okay, it's going to be true, there's got to be a proof of at most a thousand steps that will give it to us.
And so, it will always be the case that there can be arbitrary long proofs.
Now, in mathematics, that doesn't tend to bite one very much, in the kind of mathematics that human mathematicians have done, and that's partly because the kinds of things they've tried to prove are these things that have all this sort of conceptual structure and so on, and are not just things that go, you know, many, many, many steps of a sort of low-level proof.
In computing, it's a different story. Programs aren't set up to have that same kind of conceptual, kind of structure improvability. They just do what they do to compute what they compute. And

typically, it's super hard to be able to verify, to prove a theorem about how this program will work.
You know, in Wolfram language, for example, there are many functions that effectively prove theorems about things. You know, you say, is this quantity greater than zero?
And it's some messy thing with square roots and all that kind of thing. And Wolfram language will just say yes or no. And that's effectively a theorem. And the way it works inside is not by constructing a proof of the kind that a human might do, it uses algorithms that can confirm that the thing is always true, for example.
But at the level of what it's… how it's used, it's kind of like a proof.
So, I think… in,
So, but when it… when it comes to, sort of, the general program.
that it is extremely difficult to sort of make a proof about how that program works. There are other special cases. For example, one is, if you have… if you're trying to compile code.
you want to know, will this thing always be an integer, or can it sometimes be a fraction? And you might have, effectively, a proof that it will always be an integer, and that allows you to write… to take your code and turn it into the right machine code, and so on. Those are little mini theorems that show up all over the place in doing practical computation.
But the bigger question of, sort of, a big program, can you verify it, it's super hard. And I think it's theoretically hard, and probably, in practice really undoable, except in very simple cases.
You know, can you prove that this program will never have a bug? There's even a problem there, I think I've talked about this another time, there's even a problem there in knowing what it means to have a bug. You have to have another specification that says that it is itself a program.
In any case, the, so…
The, this is sort of a practical application of this kind of proving things idea,
But the main application is proving things about software, things about networking protocols, business rules, things like this, not really so much math. Math tends to build these taller towers that are very conceptual, as opposed to what programs do, which are much more sort of down in the weeds and have these problems of undecidability and so on.
So I've certainly wondered, because there's been quite a lot of developments in recent times in auto-formalization and being able to kind of take an outline of a proof
and turn it into something where it's really fitting the puzzle pieces together with a proof assistant, doing that automatically with AI. I've been curious, can I use that? And I did realize a few weeks ago, something that should have been more obvious to me for ages, that I personally have a great use case for that.
I'm not sure that that many people in the world really care about this use case directly, but I certainly have a good use case whose implications are important, but the details, the technical details, are definitely a niche kind of thing. Let me explain what that is.
So it has to do with actually determining whether some particular setup can have undecidability. And the main way one does that is by showing that this particular thing, with these particular rules, you can set it up so that you can arrange those rules so that it will basically work like a computer, or work like any computer.
So you can imagine, you've got a computer and it's got some hardware, it's got little switches, you know, a microprocessor with little transistors in it that act like switches, and it's got certain machine instructions that it will do this, it will add a number, it will store this in a particular memory location, etc, etc, etc. Then the question is, do I have enough instructions in my computer that I can do any computation you might specify?

Five.
Well, that requires knowing, is this going to be a universal computer? If the computer only has the operations of addition and subtraction, that's not enough.
to be able to do any computation. You have to be able to store things, you have to be able to say, if this happens, then do that rather than this, and so on and so on and so on.
So, there's a certain sort of minimum set of operations that you need to make a universal computer. But there's not…
only one such minimal set. There are many systems that are very simple, that are just sort of lines of black and white cells, like a cellular automaton, for example, where, depending on the particular rules for updating those black and white cells, you might be able to make a computer, or might not.
basically the heuristic is, if the behavior of the system looks simple, it's probably not going to be sophisticated enough to make a computer. If it looks more complicated, it probably is going to be sophisticated enough. That's kind of the informal version of my principle of computational equivalence.
That, essentially, anything that is not obviously simple in its behavior will be effectively doing a computation that's as sophisticated as any computation can be.
So the, so then there's a question of, well, in this principle of computational equivalence, it makes that informal statement. Can one prove that statement? Can one show that these actual things one can pick up that look simple, but nevertheless behave in complicated ways, are actually capable of universal computation?
So… in, Over time.
Well, I've tried to do several of those proofs. Typically, I'm not the one at the front lines being the automated theorem prover. I'm dealing with, in most cases, humans doing that.
Back in the 1990s, there was one of these that I… that I did, that I had a research assistant spend, what was in the end, a couple of years kind of grinding through these very arcane sort of microscopic details of how bit patterns can relate to these bit patterns and so on. And eventually, yup, you could show that you could put together a proof
That this particular very simple system was computationally universal.
Then, 2007, I was trying to do this for another very simple computational system, a Turing machine, and I'd sort of found the simplest possible candidate
For what could be a universal computer in that class of systems, a very, very tiny thing, just as the other one, which was the cellular automaton, was also a very, very tiny thing.
That sort of was… was very simple to specify the rules, but they did complicated things, and my speculation was it was possible to arrange those… to set things up so that you could use those rules to do any computation you want.
So, 2007, I put up a little prize for somebody to prove or disprove that this particular Turing machine was computation universal, and a young chap in England, in about 6 months or so, produced this proof. It's this complicated 50-page document, full of code and so on, that is the proof
That this particular very simple system is computationally universal.
Okay, so, my idea is…
How about auto-formalizing those proofs? How about making those into formal proofs in a proof assistant? How about getting an AI to do that for one?
And then, how about saying, well, let's use the pattern of those proofs to go on and try and extend the evidence, or not, for the principle of computational equivalence by finding out

whether other systems that have simple rules but look complicated in their behavior are also computationally universal.
Okay, so I tried this, using the kind of latest, greatest tools for auto-formalization and AI and LLMs and proof assistance and so on.
And… It's interesting, because it's not obvious what the answer is. In other words.
the system produces a proof. Here it is. It's in the proof assistant language, and you can verify it using the proof assistant.
Now, does it actually prove what you think it proves? That's a hard problem. And it's been, you know, we just started looking at that.
And really, it's really hard to tell. I mean, it's a difficult human thing to try and prove that the proof is actually a proof of the thing you think it's a proof of.
Because that's where you have to take this sort of formalized statement and attach it to something which is our version of what we think a universal computer is.
And that's a subtle thing, because when you have a universal computer, what you're doing is you're saying, I can take whatever program I want to write, and I can effectively compile it, convert it into the machine code of that machine, so that it will operate.
But you can kind of cheat, because the compiler that goes from your program to the machine code of the machine could itself be doing a complicated computation, and that would be kind of a cheat. And so you have to verify that that's not what's going on. And stating that condition mathematically is not so trivial.
So, I don't know. I mean, it… these tools ground around for a long time, and then auto-formalized one of these proofs.
But I don't know, I mean, it was only a few days ago. I don't know if it's actually true. And… but that's a case where I think it's an interesting thing to try and go on and look at, sort of, what this is a case where I think, like this axiom for the logic that I found years ago, I think this is a case
where one can really use the latest, greatest technology, and grind all that GPU time and so on, and find something which is, in the end, sort of a gem of kind of computational and mathematical results. Even though you were down in the mud, sort of looking at all these details, in the end, you get something that's really kind of an interesting thing to have.
And it's a case where we kind of know what we're trying to prove, and the question is, can we prove it?
So that's a case where I think, may be able to use the kind of latest, greatest technology to establish something which I at least find very interesting.
Boy, that was a long explanation.
Let's see…
Kaylee is asking, will there be famous AI proofs the way there are famous human ones today?
I think it depends on… there are questions which are out there for a long time, and then they finally get proved, and those tend to be the more famous ones. I mean, I… I think, like, the, you know, AI or automated proof that I did in the year 2000 for my little
thing that gets called Wolf and Axon these days, is,
I don't know, is it famous?
maybe the axiom is a little bit famous, I think the proof isn't famous at all, really. Just the fact that there is a proof.
Now, you know, I think it's usually the case, the proof itself isn't what gets famous. It's the… that there is a proof, and maybe it's so-and-so who did that proof.

And certainly,
You know, people who make software that tries to compute things or prove things would like people to say, it was proved by my software system.
we always like to see that when people put that in papers, you know, it was done with Mathematica or Wolf and language or whatever. Sadly, most people don't do that, because they just think, well, it was… it was done, you know, and I'm sure it's right, because it was just a computer that did it.
So, I don't know whether that will… how that will socially develop.
Let's see…
Let's see…
Well, this is a question from Reflection.
How is this related to… A physical proof.
Can anything be proved?
Well… Proof.
There's a kind of abstract, formal idea.
You have certain axioms, and then you prove things.
If you say, Prove that by doing, like, an experiment in the natural world.
That's a really different meaning, at some level, about proof.
It's like, prove that this drug works by feeding it to, you know, 200 people and showing that they all get better, or whatever.
It's…
That's a very different kind of proof from the kind of proof that one expects in mathematics.
Proof in mathematics is once… once you've fitted those puzzle pieces together, they'll always fit together. There's no doubt. There's… it's a… it's a… it's a thing, it's an abstract thing.
And if you test your drug on 200 people and they all get better.
There's really nothing to say that the 201st person you test the drug on will also get better. That 201st person might have different genetics, or different… or might be… have eaten a tomato right before they took the drug, and that might make the drug not work. There are so many things that can happen in the world
As… as the world is.
So those kinds of sort of physical proofs are really of a different nature, and require this kind of induction that say, if I saw it work 200 times, it's going to work the 201st time, just trust me, so to speak. But you can't show that that must always be the case in the same way that you do in mathematics.
Now, sort of a footnote to that, in our sort of fundamental theory of physics that we've developed in the last few years, that is you do get, in principle, to make sort of ultimate mathematics-style statements about the physical world.
But the statements about the world as we experience it are far, far away from the things down at the level of the sort of atoms of space and so on that are really, kind of, the place where you might be able to make sort of mathematics-style proofs.
this is a complicated philosophical can of worms, the extent to which it is possible to prove things in physics if you believe you know how all of physics works at a sort of formal level. If you've turned physics into mathematics.
then you can sort of make proofs in physics, but that's not the everyday version of how physics works and so on. That's a… that's a far down, a deep rabbit hole that ends with this thing I call the Rulliad, and it's… it's sort of not, not everyday thinking about physics and so on.

Lenu asks, should students still learn to construct proofs if AI can do it better?
I mean, you can ask that about many kinds of things. Personally, I've never been a big fan of proofs by hand. I don't really believe them, because I always think, I might have made a mistake. And I only really sort of believe things. I much more believe things that a computer computes for me than anything where I say, I can prove it, look, I can explain the steps. So I haven't been a big enthusiast of proofs.
Proofs in pure mathematics, where it's conceptual, those are achieving a quite different thing from these very detailed mechanical proofs.
And I'm not… you know, it's interesting to see that you can make proofs, and that you can see that you can make proofs. Let me give you an example.
a lot of things about infinity are things where you might say, well, I'm just gonna… let's say we want to show that,
Let's say Fermat's Last Theorem, X to the n plus y to the n equals Z to the n. Well, X cubed plus Y cubed equals Z cubed.
Are there integers X, Y, and Z that satisfy that equation?
Well, we can go and say, yeah, you know, we can check up to X, Y, and Z equals up to 100, let's say. We can check with all combinations of numbers from, you know, 72 plus 34 equals whatever. We can check that it doesn't work for any of those numbers.
But that doesn't tell us that it will never work, that for any of the infinite number of possible numbers, that that equation is never satisfied. To figure that out, when sort of infinity is in the picture, then the only way to do that is with a proof, an abstract proof.
There's no way that we can say, let's just try cases and let's compute things. We have to have sort of an abstract proof. Now, a little bit of a footnote to that, because you can introduce the notion of infinity, and you can do, essentially, computations with the notion of infinity. Like, for example, 1 over infinity, we can say is 0.
That we don't need… there's no proof involved there, it's just a fact about… a computational fact that that works out that way.
So… Yeah, I think, I think, this…
This question, you know, when it comes to proofs that have conceptual content.
then it's pretty interesting for humans to do that. When it comes to proofs that are down in the weeds, you know, with all these micro details, that's really not a thing for humans.
And, they neither learn by doing them, nor necessarily learn from the details of what's done there, apart from just knowing that the result is true.
Lenu asks, could there be proofs that are true but fundamentally impossible to communicate?
Well, I think my proof of the simplest axiom for Boolean algebra is probably a good candidate for that.
I mean, it really is incomprehensible. You follow its steps, and by step three, you're, like, really scratching your head. What the heck is going on here? It's like, you can see that it works, but it's like.
What is this? Why, you know, it's some very weird thing that happens to work that way.
Let's see… Gregory's asking, what's the strangest kind of proof you think could exist?
I mean, there are different kinds of proofs right now. There are proofs
Proofs require axioms about how a proof could be made. That's part of the axioms
for determining proofs. So, for example, a very common one is the axiom of induction. It's a fundamental axiom of the piano axioms of arithmetic.
Induction says, if you've got a thing, and it's true.

for… and you say, if that thing is true for a certain value n.
then we can show that it's true for value n plus 1. So, in other words, if we can show that for all integers n Something is true for the integer n plus 1, That…
So we take that as something we proved.
And then we say, and let's also prove it for the case n equals 0 to begin.
Well, induction is then the statement that if you know it for the base case n equals 0, and you know the induction step.
If it's true for case N, then it's true for case N plus 1.
then the axiom of induction tells you that… that then you can… then you can claim that it's true for all N.
In other words, if it's true for a particular N, and from that n you can get to the next n, so to speak, and there's a base case that it's true for n equals 0, you can kind of hop down this line, saying, it's true for that, then it must be true for the next thing, and the next thing, and the next thing.
So that axiom of induction is kind of equivalent to saying that numbers are just in a line.
If numbers somehow branched out, if you were saying it's true, it's true, it's true, it's true, it's true, but then, wait a minute, there's a branch in these numbers. There's one set of numbers that goes this way, another set of numbers that goes that way. But when you're sort of establishing it keeps on being true, well, which branch is it true on?
And that's… so induction is effectively the statement there are no branches in the sequence of numbers, even as you go infinitely far.
And… but the accident of induction is able to make a statement about… is making a statement about, sort of, what's true for any number, and that's what you need to prove this theorem about what's true for all numbers, an infinite collection of numbers. By the way, when it comes to that phenomenon of branching, there's a thing called transfinite induction, which is induction not on individual numbers, but on sets of objects, and then you do have this kind of branching phenomenon
And the axiom of transfinite induction is an axiom that appears in set theory, which is a sort of more complicated axiom system than the axioms of arithmetic. And so you can prove different things with transfinite induction than you can prove with ordinary induction.
So, and there are many things which are not provable, with ordinary arithmetic, which become provable when you add this transparent induction.
So… Let's see. Proofs. Well, so they're proofs by induction.
going down the line. They're proofs by transplant induction, where you go out into this tree of possibilities. They're proofs
that, Kind of reducto ad absurdum.
Show that if this and this and this, then you end up with a contradiction.
That's another common form of proof.
I think I can imagine proofs that…
In this sequence where you go from induction to transfinite induction, kind of sequences of lines to infinite trees.
You can imagine more elaborate versions of that.
More elaborate forms of induction.
that are related to more elaborate kinds of computational structures, not just the periodic structure of go from this to this to this, or the nesting structure where you go from this tree to a

subtree to a subtree. More elaborate versions of that, one can imagine, and those could be more elaborate types of axioms with more elaborate types of proofs.
There's a few examples, at least.
Boy.
Gregory is asking, should proofs be optimized for correctness, brevity, or interpretability?
Okay, the next thing one wants to say is, Most results in mathematics There's at most one proof. That anybody knows, or has written down.
But every result will have a large, even infinite number of possible proofs.
So you can think about this.
as…
talked about proofs. We talked about puzzle pieces and fitting together puzzle pieces. Imagine you have one of those,
What are the sliding block puzzles, where you're sliding around these blocks, and you're trying to get the blocks from one configuration to another configuration.
you can think about that sequence of configurations you're trying to go to. Can you get from this configuration to that configuration as kind of, like…
Like doing a proof.
And, in fact, in general.
You… you know that when you're doing these puzzles, solving these puzzles, there's typically many more than one way to solve the puzzle.
Just like if you're having game, like, for example, the Towers of Hanoi puzzle, where you have a bunch of discs, and they're from big… they're on one peg, they're from big to small at the beginning, they're three pegs, typically, and you're trying to move things around, always having it be the case that there's never a disc that's… that's on top that's bigger than one underneath. You're always trying to move those discs
from, you know, this place to that place, and back, and so on. And it's kind of this nested structure that ends up being sort of the algorithm to solve Towers of Hanoi.
But there are actually many different ways to do it. You could go, you know, the first disk on top, you can go to peg 3 or to peg 2 first, and you can do the next one, and peg this, and that. There's many possibilities, many branchings in the strategy you can use to solve Towers of Hanoi. And actually, you can make kind of a graph, the game graph.
of what those possible moves are, and it's one of those nested fractal Sapinski, patterns.
So, there are many different paths, which all are like the proof, in some sense, that you can go from one configuration of Towers of Hanoi to another. There are many different paths of proof. And that's a… that's surely a general feature of proofs in mathematics as well, but nobody studies that. I mean, this idea that there are many paths of proofs is a thing that's sort of a higher level of mathematics. It's a higher level of metamathematics. It's in the space of all possible proofs. What does it look like?
I think it's very interesting to try and understand the geometry of the space of possible proofs. It's kind of like a proof is a path, and you're kind of saying, well, how do all these paths that are the proofs of all the things we can prove in algebra, let's say, how do they all fit together? And, I've tried to study this a bit, but…
this whole question about, you know, different paths or different proofs, and you can then say, well, how about a proof that two proofs are equivalent? You've got these two paths, and then you need kind of this webbing between the two paths that says, yep, this proof is equivalent to this other proof. You can make… you can move this proof to be that proof.

And sometimes, in the space of possible proofs, proofs, those two paths will be kind of such that you can just make little incremental changes to each path to say, yup, these things are the same. Or sometimes there may be a big hole.
in proof space. And it may be the case that you sort of… there's one proof over here, and there's no sequence of small changes to the proof that will take you over here to that other proof. Nobody's really studied that. Kind of the homotopy of proof space, of kind of the… in mathematical terms, kind of how the topology of proof space, how… what holes are there in the space of proofs? It hasn't been studied. I've looked at it a bit, but there's much more that can be done there. But that's kind of a meta level of thinking about proofs.
I think, It's,
Yeah, so another question that's here from Brady. Can AI resolve all zero-day exploits before software is released and attacked by having the AI find the exploit?
I don't know. I really doubt it.
You know, just this week.
you know, new software released, which at least has the claim that it's sort of better than pretty much any human, and finding vulnerabilities in software. What does that even mean? You know, the question is.
You have a piece of software that's, for example, supposed to only let the people who are supposed to log into your computer system log into your computer system. So the question is, is there a bug that would allow the bad guys to log into your computer system as well?
Back in the 1960s, when programming was much younger, a very typical kind of bug when you're trying to log into a computer system is so-called buffer overflow. If you just, you know, you type your login, you know, mine's, you know, s.wolfram.
Whatever. The, and,
But let's say you don't type in your actual login, you just start typing random characters, thousands of characters. Well, if the software isn't written quite right, then it could be the case that when you try and store those thousands of characters in your password system, your password system will go crazy. You'll have a, you know, there'll be a bug in the software that causes that big, long number to start scribbling data
somewhere in memory you weren't supposed to have access to, and so on. Of course, those particular kinds of bugs have long ago been fixed, but much more elaborate versions of things like that afflict software all the time.
And, you know, there's really probably no piece of software in the world that has no bugs. No things that it does that aren't what you hoped it would do.
Sometimes there's a very precise definition of what you hope it will do, sometimes it's somewhat fuzzier, but you can still ask the question, is there a way of getting it to do things you hoped it wouldn't do?
And… For humans.
There are various approaches you can take. Oh, I don't know, an example would be you take the input that you thought you were going to give, and you kind of fuzz around that input. You see what happens if you sort of randomly change the input.
Does it do things, or you can do things where you say, let's watch the actual execution of the program, watch a bunch of timing details about how the program is executed? Does that reveal to us things that were inside places where we weren't supposed to have access, and so on?
All sorts of things like this.

And so the question is, can AI systems find places where there might be problems? And they seem to be getting a lot better at it.
I don't know whether that's because there's certain kinds of things that human programmers put into these programs.
By mistake, or because of things, you know, there are certain… when you write programs, there are certain very common types of mistakes one makes, like off-by-one errors. You know, you say, I'm going to remove things from this list, and then it's how many times do… no, a typical example would be, I have,
Sequence of values.
And then I'm looking at the differences between these sequences of values. How many differences are there? If there are n values, there are n minus 1 differences. And you kind of get that wrong, you're not thinking straight about that, and that's a typical kind of error.
And these are kinds of things that happen in human written software. And I think what the AIs… the AIs are pretty good at knowing us at this point. They've seen trillions of words that we've written, and so on. They know us.
And so, the mistakes that human programmers make, they have a decent chance, maybe, of being able to notice.
the mistakes that an AI programmer makes, well, maybe they're derived from human programs and have human-like mistakes, so maybe they can find those too. I'm not sure. I think programs where you did a big search
to find a program that does this, and no human has ever understood that program, much like the axiom for Boolean Algebra. I've done a bunch of these kind of search for an algorithm things, where you get out this thing where it's a particular algorithm, and it's incomprehensible why it works.
those will have a different character in terms of being able to find bugs. But I think…
It's quite likely that With the current generation…
of AI systems, there'll be at least a lot… well, it's kind of a cat and mouse game, because both… it will be easier for the AI to find the vulnerabilities before the software is shipped, but it will also be easier for people to find the vulnerabilities from the outside.
You know, it'd be easier to find bugs to fix, but it's also easier to find bugs that haven't been found before.
And I think, you know, the term zero-day exploit, basically the point there is, if there's some bug in, I don't know, the operating system for a common type of computer, let's say, or in a web browser, or something like this.
Once the world knows there's a bug, everybody's gonna download the fix.
Because fixes are usually not that hard to come by. It's just a question of knowing that there is a fix and then downloading it. So, as soon as it's out.
That there's this bug, then people get notified, and the automatic updates happen, and, you know, billions of copies of that browser get updated, and then that bug no longer lets you blow up the browser or whatever else you were going to do.
But… There's the moment when the bug is discovered.
And the question is what happens then? And that's a time when you could potentially sell the bug to somebody. I don't know what the going rate is these days. For a while, it was $10 million. It was kind of the going rate for a zero-day exploit.
And, you know, that, that people, and…

Both, sort of private entrepreneurs and governments and so on that are looking for these kinds of bugs, because if you find one of these bugs, you can use it to break into computer systems to do all sorts of other things, whether for crime or warfare or whatever else.
And these things tend to be of, thought to be pretty valuable.
Of course, once you know that there's a bug there, you can go fix it, and then you don't have that vulnerability anymore. But so this notion of a zero-day exploit has to do with, do you know about it before anybody else knows about it?
And so, sort of the question here was, will it be the case that all things that you… that anybody could ever know about will be known about, sort of, up front as a result of AI? I think the answer is there'll be a lot more that's known up front, but not everything. And also, on the other side, there's a lot more that can be discovered with AI. And I think any program that is not pretty trivial
Any program that's thousands and thousands of lines long is going to have many issues, and many of those issues are not going to be about
particular details of the code as such, but rather they're about, did the code really connect to the assumptions you were making and the things you want it to do? It's kind of outside of the formal structure. What is the, you know, what is the sort of real-world connection of this code?
Let's see… There's a question from Chewy,
Is an AI hallucination a bug, or just part of how the system works?
Well, the thing is that the way AI
AI systems and machine learning systems work.
is a little different from the way programs are normally set up to work. Normally, with a program, you say, I've got to do this particular kind of thing, and I'm going to build the steps using my computational primitives, using my computational language, whatever, I'm going to build the steps to do this computation.
Okay, what happens in machine learning systems? Instead there, what's typically done is say, I'm going to give you a zillion examples of how I want things to work. I'm going to give you a zillion pictures of cats and dogs, and I'm going to say, every time you see a picture like this, call it a cat. Every time you see a picture like that, call it a dog. It's programming by example.
But the examples will not cover everything you want to ask the system to do. That… that's… whereas in the case of writing an ordinary program, you're saying, I have an algorithm that covers every case I might deal with.
in the case of machine learning, you're giving examples, and the examples will never cover every case you might want to deal with. So the machine has to kind of extrapolate. It has to say, I kind of know what… I've got an idea of what a cat is, so I'm going to say that this thing that has an extra ribbon in its head is still a cat.
Even though I had nothing in my training data that had a cat with a ribbon on its head.
So to speak.
So, the… and hallucinations… Come when they are extrapolations that we think don't make sense.
And, because, you know, if there was a particular piece of training data, well, the system might still not reproduce that piece of training data, but at least there's a fighting chance.
If that piece of training data wasn't there, and it's extrapolating from the training data, it might extrapolate in a way that we think is kind of crazy.
Is it wrong?

Well, you have to have a mathematical definition. Let's say it's cats versus dogs picture. What is the mathematical definition of a cat picture versus a dog picture? There really isn't one. And so, that's a case where it's not kind of a thing where you can prove it's right or prove it's wrong. It's just like, does that seem like it's doing the right thing or not?
So, it isn't really the case. So, hallucinations are not, in a sense, bugs in that same sense. Now, if people make the mistake of using, sort of, these machine learning systems to try and do things which are sort of explicit computations, you know, adding numbers up, or whatever else it is. then they kind of get what they get. I mean, people say, when you look at how an LLM
an AI, a neural net, adds numbers up, it will do things that are sort of like, oh, it's got 100 plus You know, 123 plus 242.
you see it do things that are a bit like what humans do. It'll say, well, it's roughly 100, and it's roughly this, and it's about that, and I'll take this number off here and put it there, and so on and so on and so on. It's kind of a patchwork.
In a standard computational algorithm, you add numbers in an incredibly regimented, precise way.
But people, if they can tell what the neural net is doing, it's really hard to tell what it's doing, but there are some ways you can probe it to get some idea. You'll discover that it's doing these very human-like things of saying, well, it's roughly this, and it's that, and take this off, and so on, and so on, and so on. It's kind of like, is this really going to work?
Well, it's hard to know. You might give it a number of the size of a million or something, and suddenly that sort of set of heuristics that get put together doesn't work anymore.
So, if you try to set your system up so that you are just using the neural net to do something which ultimately really matters, but the computation is precisely correct, you're probably setting yourself up for failure. I mean, the way things get built in practice, there are sort of foundation models that know lots of things about the world.
I like to think of our technology with Wolfram language and Wolfram Alpha and so on as providing kind of a foundation tool for those foundation models, a foundation tool that actually definitively knows how to compute things, and definitively has knowledge about the world. And that's a… that's a place where
So it's sort of a complementary to what happens in neural nets, where it's all… everything is done by example, and it's, and it's… it's, it has to kind of guess
for itself, And, sort of, the big…
unknowable and advanced result about neural networks is the ways they extrapolate things seem to follow more or less the ways we think it's reasonable to extrapolate things. They more or less agree with us about what's a picture of a cat versus a picture of a dog.
And presumably they do that because the essence of the architecture of an artificial neural net is similar enough to the essence of the architecture of human brains that we more or less say, that's, you know, you did what I would have done, because you kind of work more or less the same way.
I think I need to get going to my… Day job soon, but,
There's a question here from…
Gregory, asking, should proofs be optimized for correctness, brevity, or interpretability?
Depends what you're trying to prove. You know, if you're trying to prove that your cryptocurrency is not going to double spend coins.

I don't think beyond knowing that it's correct, I don't think anybody cares what the details of the proof are inside. If you're trying to establish some… some conceptual framework in pure mathematics, then it matters a lot what's inside.
If you're trying to sort of have people, make,
oh, I don't know, do things like memorize proofs, then brevity obviously matters. I see another question came in from LC. What makes a cat a cat? Well, that is indeed the question. Whether,
And that's what, you know, when you look at a neural network that's seen, you know, 100,000 pictures of cats, and seen gazillions of pictures of other things, and you say, what makes a cat a cat?
The neural net can't tell you. It's… that's not its nature. Just like with our human brains, there are many things where if you said, you know, what makes a cat a cat.
you have an interpretation in your brain that tells you to associate this particular image with a cat, but if somebody says to you, dissect why you thought that was a cat.
That's not something we think about. It's something that's happening at a subconscious level in our brains with those 100 billion neurons that we have in our brains, doing all this complicated processing that is not being sort of packaged into something which is like a thought that we can tell, say, talk about in words. I mean, you know, what makes a cat a cat?
at the level of the, sort of, biology, that's, you know, we can have all kinds of discussions about the taxonomy of cats, and the morphology of cats, and so on, and, but when it comes to, sort of, a visual image, I think
it's, it's in the end… so, so an interesting point there would be, we have an image of a cat.
And… The things that are not cats are far enough away in visual space that we can reasonably say.
That thing's a cat, that thing's not a cat. Okay, let's talk about crocodiles and alligators.
I always… well, I actually remember the difference, but… but they're kind of similar looking, and if you say… when there are things that sort of push in on similarity, like it's a crocodile, it's an alligator.
Cats are a little bit singular. They're not… it's not like there's a sub-cat or a cat, too, that is really close to a cat in its visual appearance, but which one would
perhaps even pedantically say, that's not a cat. That's a cat, too, instead. And, you know, we see it in lots of other kinds of animals. Is it a, I don't know, a frog or a toad? Is it a turtle or a tortoise? You know, some of those are sub-cases of others, but… but,
it's something where, where sort of an interesting feature of, is it a cat is, to some extent, in the space of possible images, how close are the things that a cat like?
And there are some places where, for the case of cats, things may be fairly far away. Other cases… and it may be that
Knowing that it's a cat becomes more difficult over time, because maybe somebody breeds, you know, a modified cat that's sort of a cross between a cat and a dog or something, and those things look awfully like cats. And you… before the year 2035, you didn't have to care about a caddog. But then after 2035, somebody's bred a caddog, and then you have to worry, you know, is the thing you're seeing really a cat?
Or is it a catog? Because that's a thing that's sort of muscled in on the… on the visual space of images of cats.
Okay, I think we're degenerating here into, and to,
I don't know what, but, we should probably wrap up for now, but thank you for… I think I went on about, the very first question that was asked here for a very long time, but, hopefully…

That was of interest to people. Always helps me to think these things through and kind of try and explain them to you folk. So, thank you for giving me that opportunity, and thanks for joining me.
And, I'll, talk to you another time.
Bye for now.