本期追问

AI给数学研究带来的是断裂式革命还是传统的延续?

形式化证明助手如何改写数学的信任与协作方式?

机器学习能替人类发现数学中隐藏的联系吗?

视频精读 VIDEO STUDY

Terence Tao at IMO 2024: AI and Mathematics

来源：YouTube 视频（https://www.youtube.com/watch?v=e049IoFBnLA）
节目发布：2024-08-21 · 全文 76 段 · 页边注释 72 条 · 生词词组 94 条

目录 CONTENTS

导读　视频总结

本期讲者

陶哲轩（Terence Tao）　加州大学洛杉矶分校数学教授，2006年菲尔兹奖得主，研究横跨调和分析、数论、组合等众多领域。13岁获IMO金牌，至今仍是最年轻纪录保持者。

主持人　IMO 2024（英国巴斯）大会环节主持人，负责介绍演讲嘉宾并主持问答。

第一部分 全文对照 · 附页边注释与生词标注

01 开场：最年轻的IMO金牌得主

0:01 主持人

Hello everyone. Hello everyone. Some of you dear contestants are very young so perhaps you do not know who Professor Terence Tao is. Just a few words of introduction: He participated at the IMO for the first time when he was 11 years old and he received a bronze medal. The next year he came back and he received a silver medal. After that at the age of 13 he received a gold medal and he was the youngest participant to receive a gold medal. Then he went to university and he didn't participate at the IMO anymore. Now he is professor at the University of California LA and I can say that he is definitely the biggest IMO star and of course one of the most influential mathematicians of our time. Especially for you: Professor Terence Tao.

大家好。大家好。在座的各位参赛选手中有些人非常年轻，也许并不知道陶哲轩教授是谁。简单介绍几句：他第一次参加IMO时才11岁，获得了铜牌。第二年他再次参赛，获得了银牌。之后在13岁时他获得了金牌，并且是当时最年轻的金牌获得者。后来他上了大学，就不再参加IMO了。现在他是加州大学洛杉矶分校的教授，我可以说他绝对是IMO最耀眼的明星，当然也是我们这个时代最有影响力的数学家之一。特别为大家请出：陶哲轩教授。

陶哲轩1975年生于澳大利亚阿德莱德，1986–1988年连续三届参加IMO，13岁夺金的最年轻纪录至今未破；2006年获菲尔兹奖，现任加州大学洛杉矶分校（UCLA）教授。

02 引言：AI与研究数学

1:41 陶哲轩

Thank you. I'm very happy to be back here at the IMO. The time I had at the IMO was one of the most fun times of my life. I still look back on it fondly. I hope we all had fun, not just in the competition whether you get a good score or not but also in the social activities. They always host a really good event here. My talk is on AI and more generally machine assistance in mathematics. You've all heard about AI and how it's changing everything. I think earlier today there was a talk by DeepMind on how there's a new product AlphaGeometry that can answer some IMO geometry questions now. There will be a presentation on the AI Math Olympia right after my talk actually so please stay after my talk for that.

谢谢。我非常高兴能回到IMO。我在IMO度过的时光是我人生中最快乐的时光之一。我至今仍怀着美好的心情回忆那段日子。我希望我们大家都玩得开心，不只是在竞赛本身、不管你的分数好不好，还包括各种社交活动。这里举办的活动总是非常精彩。我的报告主题是AI，以及更广泛意义上的机器辅助数学。大家都听说过AI，也知道它正在改变一切。我想今天早些时候有一场DeepMind的报告，讲到有一个新产品AlphaGeometry，现在已经能解答一些IMO几何题了。实际上在我的报告结束后马上就有一场关于AI数学奥林匹克的介绍，所以请大家在我讲完后留下来听一下。

AlphaGeometry是DeepMind 2024年1月发表于《自然》的几何定理证明系统，在30道IMO几何改编题中解出25道，接近金牌选手水平。本届IMO 2024在英国巴斯举行。

fondly /ˈfɑːndli/ *adv.*
深情地，怀念地；look back on sth fondly 美好地回忆某事

2:31

I'll be talking more about how these tools are beginning to change research mathematics which is different from competition mathematics. Instead of having three hours to solve a problem you take months and sometimes you don't solve the problem then you have to change the problem. It is definitely not the same as math competitions although there's some overlap in skills. It's all very exciting and it's beginning to be transformative, but on the other hand there's also a sense of continuity. We've actually been using computers and machines to do mathematics for a long time and it's just the nature of the way in which we're doing it is changing but it actually is following on a long tradition of machine assistance.

我要讲的更多是这些工具正在如何改变研究数学，这和竞赛数学是不一样的。不是给你三个小时去解一道题，而是你要花上几个月，有时候你解不出来，那就得换一个问题。这跟数学竞赛肯定不一样，尽管在技能上有一些重叠。这一切都非常令人兴奋，而且开始带来变革，但另一方面，这里面也有一种延续性。其实我们使用计算机和机器来做数学已经很久了，只是我们使用的方式在发生变化，但这其实是延续了机器辅助的悠久传统。

全篇立论基调：AI不是凭空出现的断裂，而是数千年「机器辅助数学」传统的延续。后文从算盘讲到Lean的整个历史回顾，都在支撑这一框架。

transformative /trænsˈfɔːrmətɪv/ *adj.*
带来变革的，颠覆性的

continuity /ˌkɑːntəˈnuːəti/ *n.*
连续性，延续性

03 机器算数学的千年传统

3:15

So here's a question: How long have we been using machines to do mathematics? The answer is thousands of years. Here's a machine that the Romans used to do mathematics. The abacus is one of the early machines. There are even some earlier ones. That's kind of boring. That's not a really smart machine. What about computers? How long have we been using computers to do mathematics? That's about 300–400 years. I think that's a bit weird because our modern computers – we didn't have the electronic computers until the 1930s and 1940s. But computers weren't always electronic. Before that they were mechanical and before that they were human.

那么问题来了：我们使用机器做数学已经有多久了？答案是几千年。这是罗马人用来做数学运算的一种机器。算盘就是最早的机器之一。甚至还有更早的。这有点无聊，它算不上真正智能的机器。那计算机呢？我们用计算机做数学有多久了？大概有三四百年了。我觉得这有点奇怪，因为我们现代的计算机——直到 1930、40 年代才有电子计算机。但计算机并不一直都是电子的。在那之前是机械的，再往前是人。

computer一词直到20世纪中叶都指「做计算的人」这一职业；1940年代电子计算机（如ENIAC）出现后，词义才转移到机器上。

abacus /ˈæbəkəs/ *n.*
算盘

3:56

The computer was actually a job profession – someone who computes. Here is a cluster of computers during World War II to compute ballistics and other things. They had a whole cluster of computers who were mostly girls because the men were fighting the war. With adding machines and it was basically – there were programmers who basically just told the girls what to do. The basic unit of computational power at the time was not the CPU, it was the kilgirl – how much computation you can do with 1,000 girls working like this for one hour.

computer（计算机）其实曾经是一种职业——做计算的人。这是二战期间的一群 computer，用来计算弹道之类的东西。他们有一大群 computer，大多是女孩，因为男人们都上战场了。她们用加法机工作，基本上——有程序员，他们的工作基本就是告诉这些女孩该做什么。当时计算能力的基本单位不是 CPU，而是kilgirl——就是 1,000 个女孩这样工作一小时能完成多少计算量。

kilogirl（千女时）是二战计算团队流传的真实玩笑单位。当时弹道表由大批女性计算员用机械加法机完成，其中一部分人后来成为世界上第一批程序员。

ballistics /bəˈlɪstɪks/ *n.*
弹道学

cluster /ˈklʌstər/ *n.*
集群，一组；此处双关：既指人群也暗指计算机集群

04 从数表到OEIS：表中出真知

4:40

We've been using computers actually even before that, since the 1700s or even earlier. The most basic use of computers in those times was to build tables. You may have heard of the logarithm tables of Napier. If you wanted to compute sines and cosines and so forth, you used a computer to generate the tables. When I was still in high school we still in our curriculum learned how to use these tables that were just being phased out. Of course now we had calculators and now computers. We still use tables today. In mathematical research we rely on tables – we call them databases now but they're still the same thing. There are many important results in mathematics that were first discovered through tables. In number theory one of the most fundamental results is called the prime number theorem. It tells you roughly how many primes there are up to a large number X and it was discovered by Legendre and Gauss. They couldn't prove it but they conjectured it to be true because Gauss and others they basically had computers – in fact Gauss himself

其实在那之前我们就已经在用 computer 了，从 1700 年代甚至更早开始。那个年代 computer 最基本的用途就是造表。你可能听说过纳皮尔的对数表。如果你想计算正弦、余弦之类的，就得用 computer 来生成这些表。我上高中的时候，课程里还在教我们怎么用这些表，当时它们正要被淘汰。当然后来我们有了计算器，现在又有了计算机。我们今天仍然在用表。在数学研究中我们依赖各种表——现在我们管它叫数据库，但本质上还是一回事。数学中有很多重要的结果最初都是通过表发现的。在数论里，最基本的结果之一叫素数定理。它大致告诉你不超过一个很大的数 X 有多少个素数，这是由勒让德和高斯发现的。他们没能证明它，但猜想它是对的，因为高斯他们基本上手里有 computer——事实上高斯本人

纳皮尔1614年发表对数表，把乘除化为加减。素数定理π(x)≈x/ln x实际于1896年由Hadamard与de la Vallée Poussin独立证明，陶此处口误说成1907年左右。

phased out *phr.*
被逐步淘汰（phase out 分阶段停用）

conjectured /kənˈdʒektʃərd/ *v.*
猜想，推测（数学术语：提出猜想，conjecture 亦作名词）

5:41

Was a bit of a computer – to compute tables of the first million primes and try to find patterns. A couple centuries later, there's another really important conjecture. The prime number theorem was eventually proven in 1907 or so. But there's another really central problem in number theory called the Birch and Swinnerton–Dyer conjecture which I won't talk about here but it was also first discovered by looking at lots of tables – this time tables of data about elliptic curves. A table that lots of mathematicians use now including myself is something called the Online Encyclopedia of Integer Sequences. Maybe you've encountered it yourself. You may recognize many integer sequences just from memory like if I tell the sequence 1, 1, 2, 3, 5, 8, 13 – you know that is the Fibonacci sequence. The OEIS is a database of hundreds of thousands of sequences like this.

就有点像一台 computer——去计算前一百万个素数的表，试着从中找出规律。几个世纪后，又出现了一个非常重要的猜想。素数定理最终在1907年左右被证明。但数论中还有另一个非常核心的问题，叫做Birch和Swinnerton–Dyer猜想（BSD猜想），我在这里不会展开讲，但它同样最初也是通过查看大量表格发现的——这次是关于椭圆曲线的数据表。现在包括我在内的很多数学家都在用的一个表叫做“整数序列在线百科全书”（OEIS）。也许你自己也接触过。你可能光凭记忆就能认出很多整数序列，比如我说出1、1、2、3、5、8、13——你就知道那是斐波那契数列。OEIS就是一个收录了数十万条这样序列的数据库。

BSD猜想是克雷研究所七个「千禧年难题」之一，悬赏一百万美元。OEIS由Neil Sloane于1964年创建，现收录37万余条整数序列，是数学界最常用的数据库之一。

6:40

Many times when a mathematician is working on a research problem there is some natural sequence of numbers associated. Maybe there's a sequence of spaces depending on n and you compute the dimension or how many or the cardinality of a set or something. You can compute the first five or six or 10 of these numbers, put it in, and then compare it to the OEIS. If you're lucky this sequence has already been put there by somebody else who discovered it from a completely different source, coming from studying some other mathematical problem. That really gives you a big clue that there's a connection between two problems and many promising productive research projects have come up that way.

很多时候，数学家在研究某个问题时，都会自然地关联到某个数列。比如有一列依赖于n的空间，你去计算它的维数，或者某个集合有多少元素、它的基数之类的。你可以算出前五个、六个或十个数，把它输进去，然后和OEIS里的做比较。如果你运气好，这个序列已经被别人放进去了，而那个人是从完全不同的来源发现它的，来自研究另一个数学问题。这就给了你一个很大的线索，说明这两个问题之间存在联系——很多有前景、有成果的研究项目就是这么产生的。

OEIS的核心价值：两个不同领域的问题若产生同一数列，往往暗示深层结构联系。「查表撞车」由此成为发现跨领域联系的正规研究手段。

cardinality /ˌkɑːrdəˈnæləti/ *n.* （数学）基数，集合元素的个数

05 科学计算与SAT求解器

7:24

Tables are one of the earliest ways we've been using computers. The most famous when you think of using computers to do mathematics you think of number crunching. The formal name for this is scientific computation. You want to do a very big calculation and you just do lots and lots of arithmetic – you source it out to a computer. We've been doing that since the 1920s. Maybe the first person to really do scientific computation was Hendrik Lorentz. He was tasked by the Dutch to figure out what's going to happen – they wanted to build a really giant dyke and they wanted to know what happened to the water flow and so they had to model some fluid equations. He used a whole bunch of human computers actually to work this out. He had to invent floating point arithmetic to do this. He realized that if you wanted to get a lot of people to do a lot of calculations very quickly you should represent lots of numbers of different magnitudes as floating points.

表格是我们使用计算机最早的方式之一。说到用计算机做数学，最有名的就是数值计算（number crunching）。它的正式名称叫科学计算。你想做一个非常大的计算，就是做大量大量的算术——把它外包给计算机。从1920年代起我们就在这么做了。也许第一个真正做科学计算的人是亨德里克·洛伦兹（Hendrik Lorentz）。荷兰政府委托他去搞清楚会发生什么——他们想建一座非常巨大的堤坝，想知道水流会怎么变化，所以必须对一些流体方程建模。他实际上动用了一大批“人肉计算员”来完成这项工作。为此他不得不发明了浮点算术。他意识到，如果你要让很多人非常快地做大量计算，就应该把不同数量级的数表示成浮点数。

洛伦兹（1853–1928）是1902年诺贝尔物理奖得主。荷兰须德海拦海工程（1932年建成的Afsluitdijk大坝）需预测潮流变化，他组织人力计算并因此引入浮点表示。

number crunching *phr.*
大规模数值计算（口语，crunch numbers 处理大量数据）

dyke /daɪk/ *n.*
堤坝，拦海坝（亦拼作 dike）

floating point arithmetic *phr.*
浮点算术（用科学记数法形式表示不同数量级的数）

magnitudes /ˈmægnətuːdz/ *n.*
数量级；大小（orders of magnitude 数量级）

8:24

We now use computers to model all kinds of things. If you're solving lots of linear equations or partial differential equations or want to do some combinatorial calculations, you can also solve algebra problems. In principle, many of the geometry questions you see at olympiads can be solved in principle by scientific computation. There are these algebra packages that can solve – you can turn any geometry problem say involving 10 points and some lines and circles into a system of equations of 20 real variables in some 20 unknowns and just whack it into Sage or Maple or something.

我们现在用计算机对各种各样的东西建模。如果你要解大量线性方程、偏微分方程，或者想做一些组合计算，你也可以解代数问题。原则上，你在奥林匹克竞赛上看到的很多几何题都可以用科学计算来解。有那些代数软件包可以做到——你可以把任何一道几何题，比如涉及10个点和一些直线、圆的题目，转化成一个含20个实变量、20个未知数的方程组，然后直接扔进Sage或Maple之类的软件里。

任何奥赛几何题都可用坐标法转化为多项式方程组，交给Sage、Maple等计算机代数系统——原则上「可判定」，这是下一段讨论其复杂度瓶颈的铺垫。

combinatorial /ˌkɑːmbəˈnɔːriəl/ *adj.*
组合的，组合数学的

whack /wæk/ *v.*
（口语）随手扔进、塞进；原义猛击

9:06

Unfortunately, once it gets beyond a certain size the complexity becomes exponential or even double exponential. So until recently it was not really feasible to just brute force these problems with just standard computer algebra packages but now with AI assistance maybe it's more promising. You heard a talk about that this morning. Another type of scientific computation that has become quite powerful is what are called SAT solvers – satisfiability solvers. These are meant to solve kind of logic puzzles.

不幸的是，一旦规模超过某个界限，复杂度就变成指数级甚至双指数级。所以直到最近，用标准的计算机代数软件包去暴力破解这些问题都不太可行，但现在有了AI的辅助，也许更有希望了。今天上午你们听过一个相关的报告。另一类变得相当强大的科学计算是所谓的SAT求解器——可满足性求解器。它们是用来解某种逻辑谜题的。

代数方法的瓶颈：复杂度随规模呈指数甚至双指数增长。SAT（布尔可满足性）是第一个被证明的NP完全问题（Cook, 1971），暴力求解的代价在理论上不可避免。

feasible /ˈfiːzəbl/ *adj.*
可行的，行得通的

brute force *phr.*
暴力穷举（计算机术语）；此处作动词：靠蛮力算出

satisfiability /ˌsætɪsˌfaɪəˈbɪləti/ *n.*
（逻辑）可满足性，SAT 即其缩写

9:43

Like if you have 10 statements that are true or false – maybe a thousand statements that are true or false – and you know that maybe if the third statement is true and the sixth statement is true then the seventh statement must be false. If you're given a whole bunch of constraints like that, a SAT solver will try to take all this information and conclude – can you prove a certain combination of these sentences or not? There's a more fancy version of a SAT solver called an SMT solver – satisfiability modulo theories – where you also have some variables x, y and z and you assume some laws like maybe there's an addition operation and addition is commutative and associative. You plug in these laws as well as some other facts and you try to just brute force – can you deduce some conclusion out of some finite set of hypotheses?

比如你有10个陈述，每个非真即假——也许有一千个这样的陈述——而且你知道，如果第三个陈述为真、第六个陈述为真，那么第七个陈述必然为假。如果给你一大堆这样的约束，SAT求解器会试着把所有这些信息综合起来，然后得出结论——你能否证明这些句子的某种组合成立？还有一种更高级的SAT求解器，叫做SMT求解器——“可满足性模理论”——在那里你还有一些变量x、y、z，并且假设一些定律，比如有一个加法运算，加法满足交换律和结合律。你把这些定律以及其他一些事实输进去，然后试着暴力求解——你能否从有限的一组假设中推出某个结论？

SMT（可满足性模理论）在布尔逻辑之外加入等式、算术等「理论」，是现代程序验证的核心引擎，Z3等求解器已广泛用于工业界。

commutative /kəˈmjuːtətɪv/ *adj.*
（数学）可交换的，满足交换律的

associative /əˈsoʊʃiətɪv/ *adj.*
（数学）满足结合律的

deduce /dɪˈduːs/ *v.*
演绎，推导出

06 毕达哥拉斯三元组：200TB证明

10:27

Those are quite powerful but unfortunately they also don't scale well at all. Again the time complexity to solve them grows exponentially and so once you get past a thousand or so propositions it becomes really hard for these solvers to run in any reasonable amount of time. But they can actually solve some problems. One recent success for example here's a problem that probably will only ever be solved by computer – this will not be possible to solve by a human I think unassisted. It considers what was called the Pythagorean triple problem which was unsolved until this big computer SAT solver calculation.

这些工具相当强大，但不幸的是它们的可扩展性也非常差。同样地，求解的时间复杂度呈指数增长，所以一旦超过一千个左右的命题，这些求解器就很难在合理时间内跑完了。但它们确实能解决一些问题。比如最近的一个成功案例：这里有一个问题，很可能只有计算机才解得出来——我认为人类在没有辅助的情况下是不可能解出来的。它考察的是所谓的毕达哥拉斯三元组问题，在这次大规模计算机SAT求解之前一直是未解决的。

布尔毕达哥拉斯三元组问题由Heule、Kullmann、Marek于2016年用SAT求解器解决，动用德州超级计算机800核并行计算约两天完成。

propositions /ˌprɑːpəˈzɪʃnz/ *n.* （逻辑）命题

11:15

The question is: You take the natural numbers and you color them two colors red or blue. Is it true that no matter how you color the natural numbers, one of the colors must contain a Pythagorean triple – three numbers which form the sides of a right triangle like 3, 4, 5? This was not known to be true. We have no sort of human proof of this. But we have a computer proof. It is now known that in fact you don't need all the natural numbers – you just need to go up to 7,824. No matter how many ways you can color 7,824 into two color classes, one of them will contain a Pythagorean triple.

问题是：你取自然数，用红蓝两种颜色去染色。是否无论你怎么染，总有一种颜色里必然包含一个毕达哥拉斯三元组——也就是构成直角三角形三边的三个数，比如3、4、5？这个结论此前并不知道是否成立。我们没有任何人类可读的证明。但我们有一个计算机证明。现在已经知道，其实你不需要用到所有自然数——只需要取到7824就够了。无论你用多少种方式把7824个数分成两个颜色类，其中一类必定包含一个毕达哥拉斯三元组。

注意结论的形式：一个关于全体自然数的无穷染色问题被压缩为有限阈值7825。这是拉姆齐理论的典型现象——足够大的结构中必然出现局部有序。

11:54

Now there's 2 to the 7824 such classes – you can't do it by brute force. So you have to be somewhat clever. But it is possible. Once you have 7,825, you must have a Pythagorean triple. There was an example of 7,824 where there's no Pythagorean triple in either class. That's provable. It actually I think it was the world's longest proof ever at the time. I think now it's the second longest proof. The proof required a few years of computation and it generated a proof certificate. The actual proof is 200 terabytes long although it's since been compressed to a mere 86 gigabytes.

而这样的分法有2的7824次方种——你没法靠暴力枚举。所以你必须得聪明一点。但这是可以做到的。一旦到了7825，你就必定有一个毕达哥拉斯三元组。而对7824，存在一个例子使得两类中都没有毕达哥拉斯三元组。这是可证明的。我记得当时它是世界上最长的证明。我想现在它是第二长的。这个证明耗费了几年的计算，并生成了一份证明凭证。实际的证明有200TB那么长，不过后来已经被压缩到区区86GB。

200TB的证明当时创「最长证明」纪录。计算机输出的「证明凭证」可由独立程序机械核验，这一点直接回应了「计算机证明可信吗」的经典质疑。

a mere *phr.*
仅仅，区区（表示数量小得惊人，常带反讽：86GB仍很大）

12:39

This is one way in which we can use computers just to do enormous case analysis. That's sort of a fairly obvious way to use computers. But in recent years we've begun to use computers in more creative ways. There are three ways in which computers are being used to do mathematics which I think are really exciting, particularly when they are combined with each other and with more classical databases, tables, and symbolic computation/scientific computation. First of all, we're using machine learning neural networks to discover new connections and find out ways in which different types of mathematics are correlated in ways that you would not see as a human or are unlikely to see as a human.

这是我们用计算机做巨量情形分析的一种方式。那算是使用计算机的一种比较显而易见的方式。但近些年来，我们开始以更有创造性的方式使用计算机。有三种用计算机做数学的方式我认为非常令人兴奋，尤其是当它们彼此结合，并与更经典的数据库、表格和符号计算/科学计算结合起来的时候。首先，我们正在用机器学习神经网络去发现新的联系，找出不同数学分支之间以某种方式相关联的规律，而这些是作为人类你看不到、或者不太可能看到的。

全篇转折点：从「显而易见」的算力辅助转向三种创造性用法——机器学习找联系、大语言模型生成思路、形式化证明助手验证。三者结合是陶的核心展望。

correlated /ˈkɔːrəleɪtɪd/ *adj.*
相关联的，有相关性的

13:26

Most splashy are the large language models which are in some sense very large versions of machine learning algorithms, which can take natural language – like ChatGPT and Claude and so forth – and they can sometimes generate possible approaches to problems which sometimes work, sometimes don't. You'll see more examples of this actually in the talk after mine. There's also another technology which is just becoming usable by the everyday mathematician which are called formal proof assistants. These are languages – so you know languages, computer languages you use to write executable code, programs that do things – formal proof assistants are languages that you write to check things, to check whether a certain argument is actually true and actually gives you the conclusion from the data. These have been fairly annoying to use until very recently and they're becoming now somewhat easier to use and they are facilitating a lot of interesting math projects that wouldn't have been possible without these

最引人注目的是大语言模型，它们在某种意义上是机器学习算法的超大版本，可以处理自然语言——比如ChatGPT、Claude等等——它们有时能生成解决问题的可能思路，有时管用，有时不管用。在我之后的那场报告里，你们会看到更多这方面的例子。还有另一项技术，正开始变得让普通数学家也能上手使用，那就是形式化证明助手。它们是一些语言——你知道，计算机语言是用来写可执行代码、写能做事的程序的——而形式化证明助手是你用来检查东西的语言，检查某个论证是否真的成立、是否真的能从给定的数据推出结论。直到不久前，它们用起来都相当烦人，而现在正变得比较好用了，并且正在促成很多有趣的数学项目，没有这些证明助手的话它们是不可能完成的。将来它们会和我这里讲的其他工具很好地结合起来。

形式化证明助手与编程语言的关键区别：程序语言描述「做什么」，证明语言描述「为何成立」——编译通过即逻辑无漏洞。

splashy /ˈsplæʃi/ *adj.*
抢眼的，引人注目的，大张旗鼓的

executable /ɪgˈzekjətəbl/ *adj.*
可执行的（executable code 可执行代码）

facilitating /fəˈsɪlətetɪŋ/ *v.*
促成，使便利（facilitate）

07 四色定理与开普勒猜想

14:32

Proof assistants. They will combine very well in the future with the other tools I described here. So I want to talk about these more modern ways to use machines and computers to do mathematics. I think I'll start with proof assistants. The first really computer-assisted proof maybe in history was the proof of the Four Color Theorem – every planar map can be colored using only four colors. That was proven in 1976. This was before proof assistants – it wouldn't really be called a computer proof nowadays. It was a proof which was a massive computation, like half of which was done by computer and half of which was done by humans.

所以我想谈谈这些更现代的、用机器和计算机做数学的方式。我想我先从证明助手讲起。历史上第一个真正意义上的计算机辅助证明，也许是四色定理的证明——每张平面地图只用四种颜色就能着色。那是在1976年被证明的。那时还没有证明助手——按今天的标准它其实算不上计算机证明。那是一个包含大量计算的证明，大概一半由计算机完成，一半由人完成。

四色定理由Appel与Haken于1976年证明，是历史上第一个主要依赖计算机的重大定理，当时引发「这还算数学证明吗」的哲学争论。

planar /ˈpleɪnər/ *adj.*
平面的（planar map 平面地图/平面图）

15:18

The way they proved the Four Color Theorem is that you basically induct on the number of countries. You show that if you have a massive map, there's some subgraph of countries – there was a produced list of about 1,000–2,000 special subgraphs – and every big graph of countries had to contain in some sense one of these subgraphs. That was one thing they had to check. Then they had to check that every time you had a subgraph you could replace that subgraph with something simpler and if you could four-color the simpler thing you could color the main thing.

他们证明四色定理的方法，基本上是对国家的数量作归纳。你要证明，如果你有一张很大的地图，其中必然存在某个国家构成的子图——他们列出了大约1000到2000个特殊子图——任何大的国家图都必然在某种意义上包含其中一个子图。这是他们需要验证的一件事。然后他们还得验证：每当出现某个子图时，你都可以把它替换成更简单的东西，而如果更简单的图能四染色，原来的图也能。

证明思路是标准的「最小反例+归纳」：先列出一组不可避免的构形，再证每个构形都可约化为更小情形，从而不存在最小反例。

induct /ɪnˈdʌkt/ *v.*
（数学）作归纳；induct on n 对 n 作归纳法

15:46

They had to check these properties – I think they're called dischargeability and reducibility – for each of these 10,000 or so subgraphs. I think one of these tasks they could do by computer although this was an early computer – I think they had to enter in each graph by hand into this one program and check it. The other task was actually done by a human computer – one of the daughters of one of the authors actually had to spend hours and hours just manually checking this reducibility thing. It was very tedious.

他们必须对这一万个左右的子图逐一验证这些性质——我记得叫做“可放电性”和“可约性”。我想其中一项任务他们可以用计算机来做，不过那是早期的计算机——我记得他们得把每个图手工输入到那个程序里再做检验。另一项任务实际上是由一位“人肉计算员”完成的——其中一位作者的女儿不得不花上几个小时又几个小时，纯靠手工去核对这个可约性。非常枯燥。

耐人寻味的细节：1976年的「计算机证明」有一半靠人工验算——作者之一的女儿手工核对可约性数月。人机分工的边界在不同时代完全不同。

tedious /ˈtiːdiəs/ *adj.*
冗长乏味的，枯燥繁琐的

16:21

The process was not perfect – there were lots of little mistakes and they had to update the table. So it was not by modern standards a computer proof or computer-verifiable proof. That only came much later in the '90s where there was a simpler proof using a mere 700 or so graphs. But now all the things that need to be checked – there was a very precise well-defined list of properties and you could write code in your favorite computer language, C or Python or something, and you can check it in a couple pages and a couple hundred lines of code in a few minutes with a modern computer.

这个过程并不完美——出现了很多小错误，他们得不断更新表格。所以按现代标准，它算不上计算机证明或计算机可验证的证明。那要到很久以后的90年代才实现，当时出现了一个更简单的证明，只用了700个左右的图。而现在，所有需要验证的东西——都有了一份非常精确、定义明确的性质清单，你可以用你喜欢的计算机语言写代码，比如C或Python之类，用几页纸、几百行代码，在现代计算机上几分钟内就能验证完。

1997年Robertson、Sanders、Seymour、Thomas给出仅633个构形的简化证明，验证程序几百行代码几分钟跑完——问题没变，变的是计算基础设施。

16:59

To actually check that is completely – to write a proof that goes all the way down to the axioms of mathematics – that was done in 2005 using a proof assistant language called Coq. I think it's now renamed to Rooster. That was one of the first proofs and you see there's a huge gap between sort of when the proof first appeared and then when we actually could completely verify it by computer. Another famous example is the Kepler conjecture for sphere packing. This is a really old conjecture – Kepler from the 17th century. It's very easy to state: You take a whole bunch of unit spheres and you want to cover three-dimensional space as efficiently as possible.

而要真正把这件事彻底验证——写出一个一路回溯到数学公理的证明——那是在2005年完成的，用的是一种叫Coq的证明助手语言。我记得它现在改名叫Rocq（“公鸡”）了。那是最早的几个这类证明之一，你能看到，从证明最初出现，到我们真正能用计算机完全验证它，中间隔了很大一段时间。另一个著名的例子是关于球堆积的开普勒猜想。这是一个非常古老的猜想——开普勒是17世纪的人。它非常容易叙述：你取一大堆单位球，想尽可能高效地填满三维空间。

Coq诞生于1989年的法国INRIA，2024年更名Rocq。Gonthier于2005年在Coq中完全形式化四色定理，与原证明相隔近30年——「验证滞后」是常态。

axioms /ˈæksiəmz/ *n.*
公理

17:41

There's sort of an obvious way to try to pack spheres – it's a triangular packing. Like the way you pack oranges at a grocery store. There's also a dual packing called the cubic packing which has the same density. There's a density of about 74% which is the obvious packing and the question is – is that the best possible? This turns out to be a surprisingly hard problem. In two dimensions, the hexagonal packing is not too hard to show it's the best. Only very recently in 8 and 24 dimensions do we know the answer – great work of Viazovska, maybe she talked about it yesterday. But three was the only other case that we know except for one which is trivial.

有一种很显然的堆球方式——三角形堆积。就像杂货店里码橙子的方式。还有一种对偶的堆积叫立方堆积，密度是一样的。密度大约是74%，这就是那个显然的堆法，问题是——它是不是最优的？事实证明这是一个出乎意料地困难的问题。在二维中，六边形堆积不太难证明是最优的。直到最近，我们才知道8维和24维的答案——维亚佐夫斯卡（Viazovska）的杰出工作，也许她昨天讲过这个。但除了平凡的一维之外，三维是我们知道答案的唯一另一种情形。

Viazovska 2016年用模形式解决8维球堆积（E8格），随后与合作者解决24维（Leech格），2022年获菲尔兹奖，是史上第二位获此奖的女性。

dual /ˈduːəl/ *adj.*
（数学）对偶的；双重的

hexagonal /hekˈsægənl/ *adj.*
六边形的，六角的

trivial /ˈtrɪviəl/ *adj.*
（数学）平凡的，显然的；日常义为琐碎的

18:23

Surprisingly difficult to prove. Again there was no completely human-readable proof of this conjecture. There is a strategy – so of course the problem is that there are infinitely many of these spheres and the density is an asymptotic thing so it's not a priori a finite problem that you can just throw at a computer. But you can try to reduce it to a finite problem. There's a strategy proposed by Toth in the 50s. Every time you have a packing, it subdivides space into these polyhedra called Voronoi regions. The Voronoi polytope of a sphere is just all the points which are closer to the center of that sphere than to all the other spheres. So you can sort of split up space into all these polytopes and these polytopes have certain volumes. You can also count their faces and the surface areas and so forth and so they all have these statistics.

证明起来出人意料地困难。同样，这个猜想也没有完全由人类可读的证明。有一套策略——当然，问题在于这些球有无穷多个，而密度是一个渐近的概念，所以它先天不是一个可以直接扔给计算机的有限问题。但你可以设法把它化归为一个有限问题。50年代托特（Tóth）提出过一套策略。每当你有一个堆积，它就把空间分割成一些多面体，叫做Voronoi区域。一个球的Voronoi多胞形，就是所有离该球球心比离其他任何球球心都更近的点的集合。所以你可以把空间切分成这些多胞形，而这些多胞形有各自的体积。你还可以数它们的面、算表面积等等，于是它们都有这些统计量。

化归思想是关键：无穷的渐近密度问题→局部Voronoi胞腔的有限统计问题。Fejes Tóth于1950年代提出此策略，并预言计算机将参与完成证明。

asymptotic /ˌæsɪmpˈtɑːtɪk/ *adj.*
（数学）渐近的

a priori /ˌeɪ praɪˈɔːraɪ/ *phr.*
先验地，事先看来（拉丁语，学术写作常用）

polyhedra /ˌpɑːliˈhiːdrə/ *n.*
多面体（polyhedron 的复数）

19:19

The volumes – like the packing density is very closely related to sort of the average volume of these regions. So if you can say something about kind of how these volumes of these polytopes behave on average, then you could get at least maybe some upper bound onto how efficient these packings can be. You can try to make relations between these polytopes – like if one polytope is very big maybe it forces the nearby ones to be very small and so maybe you can try to find some inequalities connecting the volume of one polytope to another. So maybe you should just collect lots and lots of these inequalities and then do some linear programming or something and hopefully you can just derive the right bound of this magic π/√18 which is the right density from all these inequalities.

这些体积——堆积密度和这些区域的平均体积密切相关。所以如果你能说清楚这些多胞形的体积平均而言是怎么表现的，那你至少也许能得到堆积效率的某个上界。你可以试着在这些多胞形之间建立关系——比如如果一个多胞形很大，也许它就迫使附近的那些变得很小，于是也许你可以试着找出一些把一个多胞形的体积和另一个联系起来的不等式。所以也许你应该收集大量大量这样的不等式，然后做某种线性规划之类的运算，希望能从所有这些不等式中推出那个正确的界，也就是π/√18这个神奇的数，那才是正确的密度。

π/√18≈0.7405即面心立方堆积的密度。策略本质是收集一族线性不等式做线性规划——把几何问题转成优化问题，为后文「得分函数」的灵活性埋下伏笔。

upper bound *phr.*
（数学）上界

linear programming *phr.*
线性规划（运筹学优化方法）

derive /dɪˈraɪv/ *v.*
推导出，导出（derive A from B）

20:12

People tried this. There were many attempts – some even claimed success. But none have been accepted as actual proofs.

有人试过这条路。有过很多次尝试——有些人甚至宣称成功了。但都没有被接受为真正的证明。

最著名的争议来自数学家项武义1990年代宣称的证明，因细节无法核实而未被学界接受——这正是Hales后来决心彻底形式化的历史背景。

20:23

The problem was eventually solved first by Thomas Hales and his co-authors. He did basically the same strategy but with lots of technical tweaks. He changed the cells from Voronoi cells to slightly more fancy cells. Instead of taking the volume he invented this called the score that he assigned to each of these – it's a volume plus or minus lots of little ad hoc adjustments. But again with the aim of trying to create all these linear inequalities between these different scores and to eventually get upper bounds of the density and to hopefully hit exactly the optimal density.

这个问题最终首先由托马斯·黑尔斯（Thomas Hales）和他的合作者解决。他基本上用的是同样的策略，只是加了大量技术性的调整。他把胞腔从Voronoi胞腔换成了稍微复杂一点的胞腔。他不用体积，而是发明了一个所谓的“得分”，赋给每一个胞腔——它是体积加上或减去大量临时性的小修正。但目的还是一样的：试图在这些不同的得分之间建立起所有这些线性不等式，最终得到密度的上界，并希望正好命中最优密度。

Hales与学生Ferguson于1998年宣布证明。「得分」是在体积上叠加人为修正项的拼凑量，没有唯一自然的定义——灵活既是力量也是隐患。

tweaks /twiːks/ *n.*
微调，小改动（动词 tweak 亦常用）

ad hoc /ˌæd ˈhɑːk/ *adj.*
临时拼凑的，为特定目的而设的（拉丁语）

21:08

It's a very flexible method – actually it's too flexible because there are too many things you can try. There are so many ways you can set up the score and so forth. So there's a quote here: "Sam Ferguson realized that every time he encountered problems in trying to minimize his functional and so forth he could just change the score and try it again." But then all the things that they checked already they had to redo. And so the scoring function became more and more complicated. You know they worked on this for almost a decade I think. It became more and more complicated but each change we cut months through years from my work. This incessant fiddling was unpopular with our colleagues.

这是一个非常灵活的方法——其实是太灵活了，因为你能尝试的东西太多了。设置得分的方式实在太多。所以这里有一段引文：“萨姆·弗格森（Sam Ferguson）意识到，每当他在试图最小化他的泛函等等时遇到问题，他只要改一下得分函数再试一次就行了。”但这样一来，他们已经验证过的所有东西都得重做。于是得分函数变得越来越复杂。你知道，他们在这上面干了差不多十年吧。它变得越来越复杂，但每一次改动都为我的工作省下了几个月到几年的时间。这种没完没了的反复修改在同行中很不受欢迎。

引文出自Hales团队的回忆：得分函数可以随时改，但每次改动都要重做已验证的部分。「太灵活」的方法让同行根本无法跟踪验证。

functional /ˈfʌŋkʃənl/ *n.*
（数学）泛函；注意此处是名词，不是形容词「功能性的」

incessant /ɪnˈsesnt/ *adj.*
不间断的，没完没了的（多含贬义）

fiddling /ˈfɪdlɪŋ/ *n.*
反复摆弄，小修小改（fiddle with 瞎鼓捣）

21:46

Every time I presented my work in progress at a conference I was minimizing a different function. Even worse the function was mildly incompatible with what I did in earlier papers and this required going back and patching the earlier papers. But eventually they did it. In 1998 they announced that they had finally found a score which obeyed a whole bunch of linear inequalities in 150 variables which they did minimize and got their thing. Initially they did not plan to make this a computer-assisted proof but as the project became more and more complicated it was inevitable they had to use more and more computers.

每次我在会议上介绍我的进展，我最小化的都是一个不同的函数。更糟的是，这个函数和我在早先论文里做的东西还稍微不兼容，这就得回头去给早先的论文打补丁。但他们最终还是做到了。1998年他们宣布，终于找到了一个得分函数，它满足一大堆含150个变量的线性不等式，他们把它最小化，得到了想要的结果。一开始他们并没有打算把它做成一个计算机辅助的证明，但随着项目越来越复杂，越来越多地依赖计算机就成了必然。

注意科研的真实面貌：每次会议报告最小化的函数都不同，还要回头给旧论文打补丁。计算机的深度介入不是预先设计的，而是被复杂度逼出来的。

incompatible /ˌɪnkəmˈpætəbl/ *adj.*
不兼容的，相抵触的（incompatible with）

inevitable /ɪnˈevɪtəbl/ *adj.*
不可避免的，必然的

22:23

The proof was enormous by the standards of 1998. It was 250 pages of notes and three gigabytes of computer programs and data. It actually had a very tough time getting refereed. It got sent to the top journal Annals of Mathematics and it took four years to referee with a panel of 12 referees. At the end they said they were 99% certain of the correctness of the proof but they could not certify the correctness of the computer calculations. They did a very unusual thing actually – they published the paper with a little caveat from the editors saying this. They have since removed that caveat actually. At the time there was a lot more controversy as to whether a computer-assisted proof qualified as an actual proof. Now I think we are much more comfortable with it.

以1998年的标准看，这个证明是庞大的。它有250页的笔记，外加三GB的计算机程序和数据。它的审稿过程也非常艰难。它被投到顶级期刊《数学年刊》（Annals of Mathematics），审了四年，动用了12位审稿人组成的评审组。最后他们说，他们对证明的正确性有99%的把握，但无法认证计算机计算部分的正确性。他们实际上做了一件非常不寻常的事——他们发表了这篇论文，但编辑加了一小段这样的免责说明。后来他们把那段说明撤掉了。在当时，关于计算机辅助的证明算不算真正的证明，争议要大得多。现在我想我们已经自在得多了。

《数学年刊》12人审稿4年，结论是「99%确信但无法认证计算部分」——人类审稿制度在计算机证明面前触到极限，这是形式化运动的直接导火索。

refereed /ˌrefəˈriːd/ *v.*
审稿（referee 学术审稿人/审稿，非「裁判」义）

caveat /ˈkæviˌæt/ *n.*
警示说明，附加限制条件（学术与法律常用词）

controversy /ˈkɑːntrəvɜːrsi/ *n.*
争议，论战

certify /ˈsɜːrtəfaɪ/ *v.*
认证，证明属实

23:11

But even after it was published there was doubt about whether it was really a proof. So this was maybe the first major high-profile problem where there was really a big incentive to really formalize this completely all the way down to first principles in a formal proof language. So Hales in fact created a language – well a modification of existing languages to do this. He called it the Flyspeck project. He estimated it would take 20 years to formalize his proof but actually with the help of 21 collaborators he actually finished in a mere 12 years. It finally appeared in 2014.

但即便在发表之后，人们仍然怀疑它是不是真的算一个证明。所以这也许是第一个重大的、高关注度的问题，让人们真正有强烈的动机去把它彻底形式化，一路追溯到第一性原理，用形式化证明语言写出来。于是黑尔斯实际上创造了一种语言——嗯，是对已有语言做了改造来做这件事。他把这个项目称为Flyspeck项目。他估计形式化他的证明需要20年，但实际上在21位合作者的帮助下，他只用了12年就完成了。它最终在2014年问世。

Flyspeck项目2003年启动，2014年在HOL Light与Isabelle中完成，21人参与、耗时12年，比Hales预估的20年短。「确定性」的成本第一次被具体量化。

high-profile /ˌhaɪ ˈproʊfaɪl/ *adj.*
备受瞩目的，高关注度的

first principles *phr.*
第一性原理，最基本原理（from first principles 从头推起）

08 Lean形式化：从Scholze到PFR

24:00

So we now have sort of complete confidence in this particular result but it was quite a painful thing to do. Moving now to the last few years – we've now figured out sort of a better workflow for how to formalize. It's still tedious but it is getting better. Peter Scholze, who is a very prominent young mathematician – Fields Medalist for instance – he's famous for many things but he created this amazingly promising area of mathematics. It's called condensed mathematics. It deploys the power of algebra, category theory and all the tools from algebra to apply to functional analysis – the theory of function spaces like Banach spaces and so forth which in analysis has really been resistant to the methods of algebra.

所以我们现在对这个特定结果有了完全的信心，但做这件事的过程相当痛苦。再讲到最近这几年——我们现在摸索出了一套更好的形式化工作流程。它仍然很枯燥，但正在变好。彼得·舒尔茨（Peter Scholze），一位非常杰出的年轻数学家——比如说菲尔兹奖得主——他因为很多很多事情而出名，但他开创了一个极有前景的数学领域。叫做凝聚态数学（condensed mathematics）。它调动代数、范畴论以及代数中的各种工具的力量，来处理泛函分析——函数空间的理论，比如巴拿赫空间等等，而分析学中这些东西一直很抗拒代数方法。

Scholze生于1987年，24岁任波恩大学教授，2018年获菲尔兹奖。凝聚数学与Clausen合作提出，目标是让代数工具适用于泛函分析等长期「抗拒代数」的领域。

prominent /ˈprɑːmənənt/ *adj.*
杰出的，声名显赫的

condensed /kənˈdenst/ *adj.*
凝聚的；此处为术语 condensed mathematics（凝聚数学）

24:51

But this area of mathematics in principle could allow one to solve questions in at least functional analysis – certain types of questions – with algebraic methods. So he set up this whole category of these things called condensed abelian groups and condensed vector spaces. I won't take long to explain what condensed means. His thesis is that all our categories of function spaces that we learn in our graduate classes are incorrect or they're not the natural ones – there are ones with better properties.

但这个数学领域原则上可以让人用代数方法去解决至少是泛函分析中的某些问题——某些类型的问题。所以他建立起这一整套范畴，叫做凝聚交换群和凝聚向量空间。我不会花时间解释“凝聚”是什么意思。他的论点是，我们在研究生课程里学到的所有函数空间范畴都是不对的，或者说它们不是自然的那一个——存在性质更好的范畴。

25:27

So he set up this theory but there was this one very important vanishing theorem which he needed to prove. I stated it here but I'm not going to explain what any of these words mean or symbols mean. But there was a very technical vanishing of a certain category theoretic group he needed to compute. Without this the whole theory doesn't have any interesting consequences. This was sort of the foundation of this theory. So he wrote a blog post about this result. He said he spent a whole year getting obsessed with the proof of this theorem, going almost crazy over it. In the end we were able to get an argument put down on paper but no one has dared to look at the details of this so I still have lingering doubts.

于是他建立起这套理论，但有一个非常重要的消没定理是他必须证明的。我把它写在这里了，但我不打算解释这些词或符号是什么意思。总之，他需要计算某个范畴论意义上的群，并证明它的一个非常技术性的消没结果。没有这个，整套理论就没有任何有意思的推论。这算是这套理论的地基。于是他就这个结果写了一篇博客文章。他说他花了整整一年痴迷于这个定理的证明，几乎为它疯掉。最后我们总算把一个论证凑了出来写在纸上了，但没有人敢去细看其中的细节，所以我心里仍然有挥之不去的疑虑。

即著名的「液体张量实验」（2020年12月Scholze在博客发起的形式化挑战）：连作者本人都不敢完全相信的证明，正是形式化最有价值的对象。

lingering /ˈlɪŋgərɪŋ/ *adj.*
挥之不去的，迟迟不散的
（lingering doubts 萦绕心头的疑虑）

26:16

"With this theorem the hope that this condensed formalism can be fruitfully applied to functional analysis stands or falls. This theorem is of the utmost foundational importance so being 99.9% sure is not enough." He said he was happy to see many study groups on condensed mathematics throughout the world but they all stopped short of the proof of this theorem. "This proof is not much fun." So he says "This may be my most important result to date – better be sure it's correct." So he was also very incentivized to formalize this theorem now in a more modern proof assistant language called Lean.

“有了这个定理，'凝聚'这套形式体系能否被富有成效地应用到泛函分析上，就成败在此一举。这个定理在基础性上极其重要，所以有99.9%的把握是不够的。”他说他很高兴看到世界各地有很多研究凝聚数学的讨论班，但他们都止步于这个定理的证明。“这个证明并不有趣。”所以他说：“这可能是我到目前为止最重要的结果——最好确保它是对的。”所以他也非常有动力，要用一种更现代的证明助手语言 Lean 把这个定理形式化。

「99.9%的把握是不够的」——因为整座理论大厦压在这一个定理上。此处道出形式化的经济学：验证投入应与结果的基础性成正比。

stands or falls *phr.*
成败系于（sth stands or falls with/on X：X成则成，X败则败）

utmost /ˈʌtməʊst/ *adj.*
极度的，最大限度的（of the utmost importance 至关重要）

incentivized /ɪnˈsentəvaɪzd/ *v.*
被激励，有强烈动机做（incentivize 给予激励）

26:48

Lean is a language that has been developed quite a lot in recent years. It comes with a crowdsourced effort to develop this massive math library. Rather than deriving everything from the axioms of mathematics which becomes very tedious the more advanced you go – and this type of mathematics is very advanced – this central math library in Lean has already proved lots of intermediate results like the type of things you would see in say undergraduate math courses like calculus or basic theorems of group theory or topology and so forth.

Lean 是近年来发展了很多的一门语言。它还伴随着一项众包工作，来构建这个庞大的数学库。与其把所有东西都推导出来——从数学公理出发，越往高深处走就越是繁琐——而这类数学是非常高深的——Lean 里的这个核心数学库已经证明了很多中间结果，就是你在本科数学课程里会看到的那类东西，比如微积分，或者群论、拓扑学的基本定理等等。

Lean由微软研究院的Leonardo de Moura于2013年创建；其社区库mathlib由数百人众包维护，已超百万行，覆盖本科到部分研究生水平的数学。

crowdsourced /ˈkraʊdsɔːrst/ *adj.*
众包的，由大众协作完成的

27:22

These have already been formalized and so you have a standing base – you're not starting from the axioms, you're starting from roughly a graduate level math education. Still a big gap to where you need to go but it helps. But in order to formalize this they had to add many extra things. The math library was not complete – it's still not complete. There's lots of areas of mathematics like homological algebra, sheaf theory, topos theory that needed to be added to the library. But in a mere 18 months they were able to formalize this theorem.

这些都已经形式化好了，所以你有一个现成的基础——你不是从公理开始，而是大致从研究生水平的数学教育开始。离你需要到达的地方还有很大差距，但这确实有帮助。但为了把这个形式化，他们必须补充很多额外的东西。数学库当时并不完整——现在仍然不完整。有很多数学领域，比如同调代数、层论、拓扑斯理论，都需要加进这个库里。但仅仅用了 18 个月，他们就把这个定理形式化了。

液体张量实验由Johan Commelin等人主导，仅18个月完成核心定理的形式化，远快于预期——此事让主流数学界开始正视证明助手的实用性。

27:58

The proof was basically correct. There were some minor technical issues but nothing really major was discovered. They found some nice simplifications. There were some technical steps that were just too hard to formalize and so they were forced to find some shortcuts. But actually the value of this project was more indirect. Firstly, they greatly added to Lean's math library. So now this math library can handle lots of abstract algebra to a much greater extent than it could before. But also there were other supporting software that got set up that future formalization projects have started using, including some that I did.

这个证明基本上是正确的。有一些小的技术问题，但没有发现真正重大的问题。他们还找到了一些不错的简化。有一些技术步骤实在太难形式化了，所以他们不得不去找一些捷径。但其实这个项目的价值更多是间接的。首先，他们极大地扩充了 Lean 的数学库。所以现在这个数学库处理抽象代数的能力比以前强得多。但同时还搭建起了一些其他的配套软件，后来的形式化项目都开始用它们，其中也包括我做的一些项目。

反直觉的结论：形式化没有发现重大错误，最大价值反而是「副产品」——扩充数学库、沉淀工具链。基础设施的复利效应是本段重点。

28:36

For example, one tool that was set up in the course of this project was what's called a blueprint. Taking a huge 50 page proof and trying to directly formalize it is really painful. You have to keep the whole proof in your head. But what we've realized is the right workflow is that you take a big proof and you first write what's called a blueprint which sort of breaks up this proof into like hundreds of tiny little steps. Each step you can formalize separately and then you just put them all together.

比如，在这个项目过程中搭建的一个工具叫做“蓝图”（blueprint）。拿一个 50 页的大证明直接去形式化是真的很痛苦。你得把整个证明都装在脑子里。但我们意识到，正确的工作流程是：你拿到一个大证明，先写一份所谓的“蓝图”，把这个证明拆成几百个很小的步骤。每个步骤你可以分别形式化，然后把它们拼起来就行了。

blueprint工具由法国数学家Patrick Massot开发：把大证明拆成带依赖关系的小引理网络，供多人并行认领——相当于数学证明的「工程化分解」。

blueprint /ˈbluːprɪnt/ *n.*
蓝图，详细规划；此处为Lean形式化专用工具名

29:08

So you try to break up a huge argument into lots of little pieces. You write that first and then different people in your team can formalize different parts of different steps of your argument. So they also as a byproduct of this formalization also produced this very nice blueprint. This is probably the – if you actually want to read the proof as a human – the blueprint is probably the best place to go to now. Another spin-off of this – so there's now also this formal proof which is tens of thousands of lines long but now there are efforts to try to convert that back to a human readable proof.

所以你要设法把一个庞大的论证拆成很多小块。你先把这个写出来，然后你团队里不同的人就可以去形式化你论证中不同步骤的不同部分。所以作为这次形式化的副产品，他们还产出了一份非常漂亮的蓝图。这大概就是——如果你真的想以人的方式去读这个证明——那这份蓝图大概是现在最好的去处。这件事的另一个衍生成果是——现在有了这个形式化证明，它有好几万行长，但现在有人在努力把它再转换回人类可读的证明。

注意双向管道：人写的证明→形式化→再自动生成可交互的人类可读证明。形式化不是终点，而是一种新出版形态的中间表示。

byproduct /ˈbaɪˌprɑːdʌkt/ *n.*
副产品，附带成果

29:42

So another thing that's been developed is that there are now tools – you can take a proof that's been written in say this language Lean – like here's an example where there's a proof written of a topological problem and they converted it back to a human readable proof. So all this text here is a proof that is computer generated from a formal proof. It looks like a human proof. It uses the same sort of math language but it's much more interactive. You can click on any location – I can't do it because it's a static PDF – but you can click on any location here and it will tell you wherever you are what the hypothesis is, what you're trying to prove, what the variables are. If there's a step that is too short you can expand it and it will explain where it came from and you can go all the way down to the axioms if you want.

所以另一个被开发出来的东西是：现在有工具了——你可以拿一个用比如 Lean 这门语言写的证明——比如这里有个例子，是一个拓扑问题的证明，他们把它转换回了人类可读的证明。所以这里的所有文字，都是由形式化证明自动生成的证明。它看起来就像人写的证明。它用的是同样的数学语言，但交互性强得多。你可以点击任何一个位置——我现在做不到，因为这是静态的 PDF——但你可以点这里的任何位置，它会告诉你在你所处的地方，假设是什么、你要证明什么、变量是什么。如果某一步跳得太快，你可以把它展开，它会解释这一步是从哪来的，而且你愿意的话可以一路追溯到公理。

30:23

I think this is great. I think in the future textbooks will be written in this interactive style. You formalize them first and then you can have much more interactive textbooks than currently. Inspired by this, I myself started a project to formalize – so I recently last year I solved a problem in combinatorics with several people including Tim Gowers who's here in the audience. It's a problem in combinatorics – it's not super important what the problem is. There's a subset of Z mod 2 to the n, like what's called the Hamming cube, and it obeys a property called small doubling. Then there's a certain limit to how big it can be. But it doesn't really matter what the statement is.

我觉得这非常棒。我认为将来教科书会以这种交互式的风格来写。你先把它们形式化，然后你就能拥有比现在交互性强得多的教科书。受此启发，我自己也开始了一个形式化项目——我最近，去年，和好几个人一起解决了一个组合数学问题，其中包括 Tim Gowers，他就在台下的听众里。这是一个组合数学问题——具体是什么问题并不太重要。有 Z mod 2 的 n 次方的一个子集，也就是所谓的汉明立方体，它满足一个叫“小倍增”的性质。那么它能有多大就存在某个上限。不过这个陈述具体是什么其实无所谓。

PFR即多项式Freiman–Ruzsa猜想，加性组合领域的核心问题，2023年11月由Gowers、Green、Manners、陶哲轩四人解决。Gowers是1998年菲尔兹奖得主。

combinatorics /ˌkɑːmbənəˈtɔːrɪks/ *n.*
组合数学，组合学

31:09

We proved it. The proof is about 33 pages. We formalized it in relatively record time – actually probably still the fastest formalized actual research paper. In 3 weeks, in a group project of about 20 people using all this blueprint machinery that had been developed in Scholze's project. It makes the task of proving things much more open and collaborative. You get all these nice visualizations. As I said, the first thing you do is that you take your big theorem and you break it up into lots of little pieces.

我们证明了它。证明大约 33 页。我们以相对创纪录的速度把它形式化了——实际上可能仍然是形式化一篇真实研究论文最快的一次。用了 3 周，一个大约20 人的集体项目，用的全是 Scholze 项目里开发出来的这套蓝图机制。它让证明这件事变得开放和协作得多。你还能得到各种漂亮的可视化。就像我说的，你要做的第一件事是把你的大定理拆成很多小块。

33页的研究论文3周内由约20人完成形式化，创当时纪录——对比Flyspeck的12年，十年间工具链进步的幅度一目了然。

machinery /məˈʃiːnəri/ *n.*
（喻）一整套机制、方法体系；原义机器设备

31:50

The theorem that we have is we call it PFR – won't explain why. That corresponds to this little bubble at the bottom of this graph here and then we introduce all these other statements. The proof of PFR has to depend on several other previous statements. These ones depend on previous statements as well. So there's this dependency graph and they have different colors depending on whether you've formalized them or not. A green bubble is a statement that you've already formally proven in your language. A blue bubble is one that hasn't yet been formalized but it's ready to be formalized – like all the definitions are in place, someone needs to actually go ahead and do it. A white bubble – even the statement hasn't been formalized yet, someone has to write in the statement.

我们的这个定理，我们叫它 PFR——就不解释为什么了。它对应这张图底部的这个小气泡，然后我们引入了所有这些其他的命题。PFR 的证明必须依赖于前面的好几个命题。而这些命题又依赖于更前面的命题。所以就有了这张依赖关系图，它们有不同的颜色，取决于你有没有把它们形式化。绿色气泡表示这个命题你已经在这门语言里形式化证明了。蓝色气泡表示它还没被形式化，但已经可以着手了——所有定义都到位了，只需要有人真的去动手做。白色气泡——连命题本身都还没形式化，得有人先把命题写进去。

> 依赖图的绿/蓝/白三色状态实质是任务看板：形式化把「理解整个证明」的高门槛，替换成「完成一个局部节点」的低门槛。

32:32

So you get this tree of tasks. The beauty of this project is that you can get all these people to collaborate on these different pieces of this graph independently. Every little bubble corresponds to some statement and you don't need to understand the whole proof in order to just work on your little piece. This was a problem in combinatorics but the people who contributed – there were people from probability, there were people who were not even mathematicians, they were computer programmers but they were just very good at sort of doing these little mini puzzle type things. Everyone just sort of picked one bubble that they think they could do and they did it. In three weeks we did the whole thing. It was a really exciting project.

于是你就得到了这样一棵任务树。这个项目的美妙之处在于，你可以让所有这些人在这张图的不同部分上协作，而且是各自独立地做。每个小气泡都对应某个命题，你不需要理解整个证明，就能做你自己那一小块。这是一个组合数学问题，但参与贡献的人里——有搞概率的，也有根本不是数学家的人，他们是程序员，但他们非常擅长做这种小谜题式的东西。每个人就挑一个自己觉得能做的气泡，然后就把它做出来了。三周我们就把整件事做完了。这真是一个特别令人兴奋的项目。

33:21

In mathematics we don't normally collaborate with this many people. Maybe five people is the most I've normally seen because when you collaborate on a big project you have to trust that everyone's math is correct and past a certain size this is just not feasible. But with a project like this the Lean compiler automatically checks – you cannot upload anything that doesn't compile, it will get rejected. So you can collaborate with people that you never met before. I met a lot of people actually – wrote a lot of letters of recommendation actually coming out of this project.

在数学里我们通常不会这么多人一起合作。我平时见过最多大概五个人，因为当你在一个大项目上合作时，你必须信任每个人的数学都是对的，超过一定规模这就根本不可行了。但在这样的项目里，Lean 编译器会自动检查——你上传不了任何编译不通过的东西，它会被拒掉。所以你可以和素未谋面的人合作。我其实因此认识了很多人——这个项目下来我还写了不少推荐信。

核心洞察：数学协作的规模瓶颈是信任，而Lean编译器把信任外包给机器——「编译不过就无法提交」使陌生人大规模协作成为可能，类似开源软件模式。

compiler /kəmˈpaɪlər/ *n.*
编译器

33:59

This is an example – like this is one little piece of the proof. This is what a proof looks like in Lean. It's not exactly – I mean if you know the language it's human readable but it looks a little bit unusual. It really sort of decouples the task of proving things into many different sort of disjoint skills. You can have some people who see the big picture and organize things into little pieces and then you have people who don't necessarily know all the mathematics but can just work on little pieces at a time.

这是一个例子——这是证明中的一小块。这就是 Lean 里的证明长什么样。它不完全是——我是说，如果你懂这门语言，它是人类可读的，但看起来有点不太一样。它真的把“证明”这件事解耦成了许多种不同的、互不重叠的技能。你可以有一些人看到大局，把事情组织成一小块一小块，然后另一些人不一定懂全部的数学，但可以一次做一小块。

「解耦」是关键词：证明工作被拆成「看大局的架构师」与「解局部谜题的贡献者」两种技能，不懂全部数学也能做出实质贡献。

decouples /diːˈkʌplz/ *v.*
解耦，使分离（decouple A into B 把A拆解为B）

disjoint /dɪsˈdʒɔɪnt/ *adj.*
（数学）不相交的；互不重叠的

34:32

I think this will be a more and more common way of doing mathematics going forward. It still is painful to do – like the tools are not really – they're getting better and user–friendlier but you still need to have some expertise in programming. I would say it takes maybe 10 times longer to formalize a proof than to write it by hand. On the other hand if you want to change a proof – so for example there was a 12 that showed up in this theorem, we later improved this 12 to an 11. We got a slightly stronger theorem. Normally if you do that you have to rewrite the whole proof or like you could maybe cut and paste 12 to 11 but then you have to check you didn't make any mistakes when you did that.

我认为今后这会成为越来越常见的做数学的方式。做起来仍然很痛苦——工具还不算真的——它们在变好、也变得更友好，但你仍然需要有一定的编程能力。我会说，形式化一个证明大概要比手写它多花 10 倍时间。但另一方面，如果你想改动一个证明——比如这个定理里出现过一个 12，我们后来把这个 12 改进成了 11。我们得到了一个稍微更强的定理。通常你这么做的话，你得把整个证明重写一遍，或者你也许可以把 12 都替换成 11，但接着你得检查替换时有没有出错。

「形式化比手写慢约10倍」是形式化界所谓de Bruijn因子的现实估计；早年这个倍数更高，工具进步正在持续压缩它。

35:15

But actually when we formalized this then we got the improvement – it only took a few days to change the theorem to the 11. We just changed the 12 to 11 somewhere and then the compiler complained like five different places now is certain this very specific part is not working and we could just do some targeted fixes. So in fact for some specific types of doing mathematics already the formal approach is actually faster. Now there are actually quite a few big proof formalization projects going on right now.

但实际上我们形式化之后，等到有了这个改进——只花了几天就把定理改成 11 了。我们就在某处把 12 改成 11，然后编译器就报错说，现在有五个不同的地方、某个非常具体的部分不成立了，我们只要做一些针对性的修补就行。所以事实上，对某些特定类型的数学工作来说，形式化的方式已经更快了。现在其实有相当多的大型证明形式化项目正在进行中。

把12改进为11只花几天：编译器精确指出五处失效位置。形式化证明像有完整测试覆盖的代码库——「重构」成本骤降，某些场景已比传统方式快。

09 机器学习发现纽结的秘密

35:46

The biggest is Kevin Buzzard's project – he's just got a big grant to formalize Fermat's Last Theorem in Lean. He says it will take five years to do the most important parts of this proof. He doesn't claim to do the whole thing in five years but the interesting part is already on the way actually. So that is formal proof assistants. I'll talk about machine learning. Machine learning – these are using neural networks to predict answers to various questions that you can use in many ways. I think I'll skip the first way I was discussing which is to use neural networks to guess solutions to differential equations which is a very exciting new tool in PDEs but I will skip it.

最大的是 Kevin Buzzard 的项目——他刚拿到一笔大额资助，要在 Lean 里形式化费马大定理。他说完成这个证明中最重要的部分需要五年。他并不声称五年内把整件事都做完，但其实有趣的那部分已经在路上了。以上就是形式化证明助手。下面我要讲机器学习。机器学习——就是用神经网络来预测各种问题的答案，用法有很多。我想我会跳过我原本要讲的第一种用法，就是用神经网络去猜微分方程的解，这在偏微分方程里是个非常令人兴奋的新工具，但我就跳过了。

Kevin Buzzard是伦敦帝国理工教授、Lean布道者，2024年获英国资助启动费马大定理形式化项目，目标五年内完成Wiles证明的最核心部分。

36:36

I'll talk about another application of machine learning to knot theory. Knot theory is quite a fun area of mathematics. It's an interesting area that brings together many different fields of mathematics and they don't really talk to each other. A knot is just a loop of string or a curve really in space that is closed. Two knots are equivalent if there's some way to continuously deform one knot to another in a way in which you're not allowed to cross the string – the string is not allowed to cross itself.

我要讲机器学习的另一个应用——纽结理论。纽结理论是数学里相当有趣的一个领域。它是个把很多东西汇聚到一起的有趣领域，汇聚了许多不同的数学分支，而它们之间其实并不怎么互相交流。纽结就是一根绳圈，或者说空间中一条闭合的曲线。如果有办法把一个纽结连续地变形成另一个，而且过程中不允许绳子穿过自己——绳子是不允许自我相交的——那这两个纽结就是等价的。

纽结理论连接拓扑、代数、几何乃至量子物理。「不同分支各自为政、互不交流」正是下文机器学习发现跨域联系的舞台。

deform /dɪˈfɔːrm/ *v.*
使变形；（拓扑）连续形变

37:09

The basic questions in knot theory are: When are two knots equivalent? If I give you two knots is there some way to turn one into the other? The way you approach this question normally is that you develop these things called knot invariants. These are various numbers, sometimes also polynomials that you can attach to a knot and these numbers don't change no matter how you continuously deform the knot. So if two knots have different invariants they cannot be equivalent. There are many types of knot invariants. There's something called the signature which counts – you flatten the knot and you count crossings, whether the crossings go over or under and you create a certain matrix and so forth and you can get a certain integer called the signature. That is one type of knot invariant.

纽结理论的基本问题是：两个纽结什么时候是等价的？如果我给你两个纽结，有没有办法把其中一个变成另一个？通常处理这个问题的方式是，你去构造一些叫做纽结不变量的东西。它们是各种数，有时也是多项式，你可以把它们附着到一个纽结上，而且不管你怎么连续变形这个纽结，这些数都不会变。所以如果两个纽结的不变量不同，它们就不可能等价。纽结不变量有非常非常多种。有一个叫做“符号差”（signature）的东西，它是这样算的——你把纽结压平，然后数交叉点，看每个交叉是上穿还是下穿，再构造某个矩阵等等，你就能得到一个整数，叫做符号差。这就是一类纽结不变量。

不变量是判别等价的标准武器：连续变形不改变不变量，故不变量不同则纽结必不等价。符号差来自Seifert矩阵，是整数值的组合型不变量。

invariants /ɪnˈveriənts/ *n.*
（数学）不变量：变换下保持不变的量

37:55

There are some famous polynomials called the Jones polynomial and the Alexander polynomial which are connected to many areas of mathematics but I won't talk about that. Then there are these things called hyperbolic invariants which come from geometry. You can take the complement of the knot and that is actually what's called a hyperbolic space. It comes with a certain geometric structure, has a notion of distance and you can compute its volume and some other invariants. These are invariants that are real or complex numbers. So every knot comes with some combinatorial invariants like signatures and it comes with these geometric invariants like these hyperbolic invariants. Here is a whole list of knots with various hyperbolic invariants – there's something called the hyperbolic volume and the homological cusp shape and so forth. These are real or complex numbers but no one knew of any link between these two. There were these two separate ways to create statistics of knots and they didn't – there was no connection between them.

还有一些著名的多项式，叫琼斯多项式和亚历山大多项式，它们和数学的许多领域都有联系，不过我就不讲了。然后还有一类叫做双曲不变量的东西，它们来自几何。你可以取纽结的补集，而它实际上就是所谓的双曲空间。它带有某种几何结构，有距离的概念，你可以计算它的体积以及其他一些不变量。这些不变量是实数或复数。所以每个纽结都带有一些组合型的不变量，比如符号差，同时也带有这些几何型的不变量，比如这些双曲不变量。这里是一整张纽结的列表，配上各种双曲不变量——有一个叫双曲体积的，还有同调尖点形状等等。这些是实数或复数，但没有人知道这两类之间有任何联系。这是两套彼此独立的、生成纽结统计量的方式，而它们之间——完全没有联系。

琼斯多项式1984年由Vaughan Jones发现（他因此获1990年菲尔兹奖）；双曲不变量源自Thurston几何化纲领。组合与几何两套统计量长期没有已知联系。

complement /ˈkɑːmpləmənt/ *n.* （数学）补集，补空间；注意与compliment（恭维）拼写区别

hyperbolic /ˌhaɪpərˈbɑːlɪk/ *adj.* （数学）双曲的；日常义为夸张的

38:55

It was only very recently that people started using machine learning to attack this problem. They created databases of millions of knots which actually was already a slightly nontrivial task and they trained a neural network on this. They found that after training the neural network you could give it all the hyperbolic geometry invariants and like 90% of the time it will predict – it will guess the right signature. So it created this black box and it will tell you how the signature was somehow hidden somewhere in these geometric invariants but it didn't tell you how – it was this black box.

直到最近，人们才开始用机器学习来攻克这个问题。他们建了包含数百万个纽结的数据库，这本身就已经是个略微不平凡的任务了，然后他们在上面训练了一个神经网络。他们发现，训练之后，你可以把所有双曲几何不变量喂给这个神经网络，大概 90% 的情况下它能预测出来——它能猜对符号差。所以它造出了这么一个黑箱，这个黑箱告诉你符号差以某种方式藏在这些几何不变量里，但它没告诉你是怎么藏的——它就是个黑箱。

指DeepMind与牛津大学合作、2021年发表于《自然》的工作（Davies等）：神经网络以约90%准确率从几何不变量预测符号差，证明几何量里「藏着」组合信息。

nontrivial /ˌnɑːnˈtrɪviəl/ *adj.*
不平凡的，有实质难度的（数学家惯用的低调说法）

black box *phr.*
黑箱：只见输入输出、不知内部机制的系统

39:32

But that's still useful because once you have this black box you can just play with it. So what they did next is actually very simple analysis – this is what's called saliency analysis. What this black box does – it takes about 20 different inputs, one for each hyperbolic invariant, and which is one output – the signature. So once you have this black box you can just tweak each input. You just say what if I change one input, how likely is it to change the output? Of the 20 inputs that they found, they found that only three of them actually played a really major role in the output. The other 17 were barely relevant and it wasn't the three that they expected actually. They expected the volume for example to be very important and the volume turns out to be almost irrelevant.

但这仍然有用，因为一旦你有了这个黑箱，你就可以拿它做各种试探。所以他们接下来做的其实是非常简单的分析——就是所谓的显著性分析。这个黑箱做的事情是——它接收大约 20 个输入，每个双曲不变量对应一个，而输出只有一个——符号差。所以一旦你有了这个黑箱，你就可以逐个去调整输入。你就问：如果我改变某一个输入，输出有多大可能会变？在他们考察的这 20 个输入里，他们发现只有其中三个真正对输出起了重要作用。另外 17 个几乎无关紧要，而且起作用的那三个还不是他们预期的那三个。比如他们本以为体积会非常重要，结果体积几乎无关紧要。

显著性分析即逐个扰动输入、观察输出变化——技术上极其简单。反直觉之处：20个输入中仅3个起作用，且专家预期最重要的双曲体积几乎无关。

saliency /ˈseɪliənsi/ *n.*
显著性；saliency analysis 显著性分析（机器学习可解释性方法）

40:13

There were three – something called longitudinal translation and the real and complex parts of meridional translation – there were these three invariants that were the most important. So once they identified the ones that were most important, they could just plot directly the signature against those three particular inputs and then they could eyeball – rather than use neural network they use the human network – to then see oh okay there's some obvious patterns here. By staring at these graphs they could actually make conjectures as to what was actually going on. They made a conjecture based on this which turned out to be wrong actually. But they actually used the neural network to show that it was wrong. But then the way that it failed, they could correct it and they found a corrected version of the conjecture which actually did explain this phenomenon. Then once they found the right statement they were able to prove it. So they actually have a theoretical explanation of why the signature is so closely related to these particular statistics.

起作用的那三个是——一个叫纵向平移（longitudinal translation），还有经向平移（meridional translation）的实部和复部——就是这三个不变量最重要。所以一旦他们找出了最重要的那几个，他们就可以直接把符号差对着这三个特定输入画出来，然后用肉眼去看——不用神经网络，而是用人类的“神经网络”——然后就会发现，哦，好吧，这里有一些很明显的规律。通过盯着这些图看，他们真的能对背后到底在发生什么提出猜想。他们据此提出了一个猜想，结果这个猜想是错的。但他们实际上是用神经网络证明了它是错的。而从它失败的方式里，他们能把它修正，并找到了这个猜想的一个修正版本，这个版本确实解释了这个现象。然后一旦他们找到了正确的陈述，他们就能把它证明出来。所以他们其实有了一个理论上的解释，说明为什么符号差和这些特定的统计量关系如此紧密。

完整的人机回路：机器发现相关性→人提出猜想→机器给出反例→人修正→人完成证明。机器负责「在哪里找」，人负责「为什么成立」。

eyeball /ˈaɪbɔːl/ *v.*
（口语）用肉眼估看，目测

longitudinal /ˌlɑːndʒəˈtuːdnl/ *adj.*
纵向的（纽结理论术语：沿经线方向的）

meridional /məˈrɪdiənl/ *adj.*
经向的，子午线方向的

10 大语言模型：惊艳与翻车并存

41:15

This I think is a way in which machine learning is being increasingly used in mathematics. It doesn't directly solve the problem for you but it gives you all these really useful hints as to where the connections are and where to look at them, but you still need the human to actually make the connections. And then finally we have the large language models which are the most splashy and have made the most news. Neural networks have been around for 20 years but large language models have also been around for 5 or so years but they've only become sort of human level in output very recently.

我认为这就是机器学习在数学中被越来越多使用的一种方式。它并不直接替你解决问题，但它会给你各种非常有用的提示，告诉你联系可能在哪里、该往哪里看，但你仍然需要人来真正把这些联系建立起来。最后我们来谈大语言模型，它们最惹眼、也上了最多新闻。神经网络已经存在 20 年了，而大语言模型也已经存在五年左右了，但它们的输出直到最近才达到接近人类的水平。

> 时间线校准：神经网络已有数十年历史，LLM也有约五年，但「接近人类水平的输出」是最近一两年的事——能力跃迁远晚于技术出现。

41:53

You've all probably heard of GPT–4. This is ChatGPT's current model. Very famously when GPT–4 came out there was a paper describing its capabilities and they fed it basically a question from the 2022 IMO. It's a slightly simplified version – if you studied the 2022 IMO you'll probably notice it's not exactly the same form but it's a simplified form. For this particular question actually you give it the question and it actually gives a complete correct solution to this question. It actually solved an IMO question. Unfortunately this is an extremely cherry–picked example. I think out of the hundreds of IMO level questions they tested it on, they had a success rate of about 1%. So this particular problem they were able to solve, and they had to format the problem in the right way to get the solution, but still this is quite amazing.

你们大概都听说过 GPT–4。这是 ChatGPT 目前的模型。很有名的是，GPT–4 发布时有一篇论文描述它的能力，他们基本上给它喂了一道2022 年 IMO 的题。这是个稍微简化过的版本——如果你研究过 2022 年 IMO，你大概会注意到它和原题形式不完全一样，但它是简化版。就这道特定的题目而言，你把题目给它，它真的给出了完整正确的解答。它真的解出了一道 IMO 题。不幸的是，这是一个极度精挑细选的例子。我记得在他们测试的几百道 IMO 级别的题里，成功率大概只有 1%。所以这道题他们能做出来，而且他们还得把题目按合适的方式排版才能得到解答，但即便如此，这也相当惊人了。

> 指微软2023年《Sparks of AGI》论文中的著名演示。1%的总体成功率与惊艳个例并存——评价AI能力时分布比单点重要，陶明确点破了cherry–picking。

> **cherry–picked** /ˈtʃeriˌpɪkt/ *adj.* 精心挑选的（只挑对自己有利的例子，含贬义）

42:46

On the other hand the funny thing about these tools is that things that humans find difficult, AI can do very easily sometimes, but things that humans find easy AI often struggles with. It is a very orthogonal way of solving problems. In the same related paper or presentation, they asked the same model to do a basic arithmetic computation: 7 * 4 + 8 * 8. The model, which is just guessing the most likely output based on the input, basically guessed the answer is 120. Then it paused and said okay maybe I should give an explanation why it's 120. So they did a step by step but when they did a step by step they actually arrived at the actual answer which is 92, not the answer that they started with. So then if you asked "Wait but you said that it was 120" and they said "Oh that was a typo, sorry the correct answer is 92."

另一方面，这些工具好笑的地方在于，人类觉得难的事情，AI 有时候能轻松搞定，而人类觉得容易的事情，AI 却常常做不好。这是一种非常“正交”的解决问题的方式。在同一篇相关论文或演示里，他们让同一个模型做一个基本的算术计算：7 * 4 + 8 * 8。这个模型，它只是在猜最可能的输出是基于输入生成的，基本上就是猜出答案是 120。然后它停了一下，说好吧，也许我该解释一下为什么是 120。于是它做了一步步的推导，但是当它一步步推导的时候，其实得到的是真正的答案 92，而不是它一开始给出的那个答案。所以如果你问它"等等，你刚才说是 120 啊"，它就会说"哦，那是个笔误，抱歉，正确答案是 92。"

LLM按「下一个词的概率」生成而非按逻辑推演，先给答案再解释时会自相矛盾。其难易剖面与人类正交：难题偶尔惊艳，简单算术反而翻车。

orthogonal /ɔːrˈθɑːgənl/ *adj.*
正交的；（喻）完全不同维度的、互不相干的

typo /ˈtaɪpoʊ/ *n.*
打字错误，笔误（typographical error 的缩略）

43:36

So you know they're not solving the problem from first principles, they're just guessing at each step of the output what is the most natural thing to say next. The amazing thing is sometimes that works, but often it doesn't. It's still ongoing how to sort of make it more accurate. People are trying all kinds of things. You can connect these models to other more reliable software. In fact you will see a presentation after where there's a large language model connected where you don't do the computation yourself, you outsource it to Python in that case.

所以你会发现，它们并不是从第一性原理出发去解题，而只是在输出的每一步猜测接下来最自然该说什么。神奇的是，有时候这样居然真的行得通，但更多时候不行。怎么让它更准确，目前仍在探索之中。人们在尝试各种各样的办法。你可以把这些模型连接到别的更可靠的软件上。事实上，等一下你会看到一个演讲，里面就有一个大语言模型被连了起来，你不自己做计算，而是把计算外包出去，那个例子里是外包给 Python。

「一步步推导反而算对了」印证了思维链（chain of thought）现象：让模型展开推理过程能显著提高准确率——这是提示工程的经典发现。

outsource /ˈaʊtsɔːrs/ *v.*
外包（outsource sth to sb/sth）

44:16

But another thing you can do is that you can force the language model to only produce correct answers by forcing the model to output in one of these proof assistant languages and if it doesn't compile you send it back to the AI and the AI has to try again. Or you can try to teach it directly the same problem solving techniques we use to solve IMO problems – you know, try simple examples, prove by contradiction, try to actually prove step by step and so forth. So people are trying all kinds of things. It's still nowhere near able to solve a large majority of say math olympiad problems, let alone math research problems, but we're making progress.

另一种做法是，你可以强迫语言模型只产生正确的答案——办法是强制模型用某种证明助手语言来输出，如果编译不通过，就把结果丢回给 AI，让 AI 重新再试一次。或者你也可以直接教它我们解 IMO 题目时用的那些解题技巧——比如先试简单的例子、反证法、真的试着一步一步地去证明，等等。所以人们在尝试各种各样的办法。目前它离能解出大多数奥数题还差得很远，更别说数学研究问题了，但我们确实在进步。

两条纠错路线：外接可靠工具（Python做计算、Lean验证证明），或注入人类解题启发式（试特例、反证法）。共同前提是不指望模型自身可靠。

let alone *phr.*
更不用说，遑论（用于否定语境后，引出更不可能的事）

45:01

Besides being able to actually solve problems directly, it's also useful just as a muse actually. I've also used these models myself. I've experimented with various problems – I had a combinatorics problem which I was trying a few things and they weren't working so I as an experiment I just tried asking GPT "What other techniques would you suggest to solve this question?" It gave me a list of 10 techniques of which like 5 I'd already tried or were obviously not helpful. But there was one technique I'd not tried which was to use generating functions for this particular question, which once it was suggested I realized it was the right approach but I had missed it.

除了能直接解题之外，它其实作为一个"缪斯"也挺有用的。我自己也用过这些模型。我做过各种实验——我当时有个组合数学的问题，试了几种方法都不奏效，于是我就当作实验，直接去问 GPT："你还建议用什么别的技巧来解这个问题？"它给了我 10 个技巧，其中大概 5 个我已经试过了，或者明显没什么帮助。但有一个技巧我没试过，就是针对这个具体问题去用母函数，一旦它提出来，我立刻意识到这才是对的思路，只是我之前漏掉了。

陶的亲身案例：GPT列出10个技巧，9个无用，但1个（母函数）恰是他漏掉的正解。AI作为「缪斯」的价值在于扩大搜索面，而非精准命中。

muse /mju:z/ *n.*
缪斯，灵感来源（源自希腊神话文艺女神）

generating functions *phr.*
（数学）母函数/生成函数：组合计数的核心工具

45:49

So just as someone to converse with, it is somewhat useful. It is not great right now but it is not completely useless. There's another type of AI assistance that's actually become very useful for proof assistants. As I said, writing formal proofs is a very tedious task. I mean it's like any really fussy computer language – you have to get the syntax exactly right. If you miss a step it doesn't compile. But there are tools – so I use something called GitHub Copilot where you can write down half of a proof and it will try to guess what the next line is. About 20% of the time it actually guesses something close to being correct and then you can just say I'll accept that and say okay.

所以单纯作为一个可以交流的对象，它还是有点用的。现在还称不上好，但也不是完全没用。还有另一类 AI 辅助，对证明助手来说已经变得非常实用了。就像我说的，写形式化证明是件非常繁琐的事。它就跟任何吹毛求疵的计算机语言一样——语法必须完全正确。少写一步就编译不过。但现在有些工具——我用的是一个叫 GitHub Copilot 的东西，你写下证明的一半，它会试着猜下一行是什么。大概有 20% 的时候它真的能猜出接近正确的东西，这时你就可以直接说"我接受"，搞定。

> Copilot约20%的建议接近正确——看似不高，但配合能即时判断对错的专家已能显著省时。AI辅助的收益大小取决于验证成本的高低。

> **fussy** /ˈfʌsi/ *adj.*
> 挑剔的，吹毛求疵的；此处形容语法要求苛刻的语言

> **syntax** /ˈsɪntæks/ *n.*
> （计算机/语言学）语法

46:37

So in this case I was trying to prove this statement here and the lines in gray are the ones that Copilot suggested. It turns out the first line is useless but the second line, which you can't quite see, actually did solve this particular problem. So you still have to – you can't just accept the input because it won't necessarily compile. But if you already know sort of how the code works it saves you a lot of time. These tools are getting better. So right now they can maybe – if a proof is one line or two lines long they can fill it in automatically. There are now experiments to sort of iterate an AI suggesting a proof and then you feed it back to the compiler and then if it compiles wrong you send the error message back. We're beginning to sort of prove things that are like 4 or 5 lines long – they can be done by this method.

比如这里我想证明这个命题，灰色的那几行就是 Copilot 建议的。结果第一行没什么用，但第二行——你可能看不太清——确实把这个具体问题给解决了。所以你还是得——你不能不加判断就接受它的输入，因为它未必能编译通过。但如果你本来就大致知道这些代码是怎么回事，它能帮你省下很多时间。这些工具在不断变好。现在的水平大概是——如果一段证明只有一两行长，它可以自动补全。现在还有一些实验，让 AI 提出一个证明，然后把它喂给编译器，如果编译出错就把错误信息再送回去，这样反复迭代。我们开始能证明大概四五行长的东西了——这些可以靠这种方法完成。

「AI提议→编译器验证→错误信息回传」的迭代循环，正是其后AlphaProof等神经定理证明系统的主流架构；2024年时能自动补全四五行的证明。

iterate /ˈɪtəreɪt/ *v.*
迭代，反复执行并逐步改进

11 展望：探索问题空间的新数学

47:29

Of course a big proof is like tens of thousands of lines so it's nowhere near the point where you can just instantly get your proof formalized immediately. But it is already a useful tool. Okay so where are we now? There are people who are hoping that in a few years we can use computers to actually solve math problems directly. I think we are still a long way away from that. For very narrowly focused problems you can sort of set up specialized AI to handle just a very narrow band of problems. But even then they're not fully reliable – they can be useful.

当然，一个大证明动辄上万行，所以离你能立刻把证明形式化还差得远。但它已经是个有用的工具了。好，那我们现在到哪一步了？有人希望再过几年我们就能用计算机直接解数学问题。我觉得我们离那一步还很远。对于非常狭窄、聚焦的问题，你可以搭建专门的 AI 来处理很窄的一类问题。但即便如此它们也不是完全可靠的——不过确实有用。

陶在2024年中的现状判断：距AI独立解决数学问题尚远，窄域专用AI可用但不可靠——与当时媒体的乐观叙事形成克制的对照。

nowhere near *phr.*
远远达不到，差得远（比 not near 语气强得多）

48:06

But still for the next few years at least, they're basically going to be really useful assistants beyond the sort of brute force computational assistance that we're already familiar with. People are trying all kinds of creative things. I think one direction which I find particularly exciting hasn't really been successful yet – hopefully AI will become very good at generating good conjectures. We already saw a little example of this with the knots where they could already sort of conjecture those connections between two different statistics. So you know there's just the hope that you just create these enormous datasets and feed them into an AI and they would just automatically generate lots of nice connections between different mathematical objects. We don't really know how to do this yet partly because we don't have these massive datasets but I think this is something that would eventually be possible.

但至少在未来几年内，它们基本上会是非常好用的助手，超出我们已经熟悉的那种暴力计算式的辅助。人们在尝试各种有创意的东西。我觉得有一个方向特别让我兴奋，虽然目前还没真正成功——希望 AI 将来能非常擅长提出好的猜想。我们已经在纽结那个例子里看到了一点苗头，它已经能猜出两个不同统计量之间的联系。所以人们就有这样一个期望：你造出这些巨大的数据集，喂给 AI，它就会自动生成大量不同数学对象之间的漂亮联系。我们现在还不太知道该怎么做，部分原因是我们还没有这些海量数据集，但我觉得这终究是可能实现的。

陶点出瓶颈不在算法而在数据：数学界还没有可供训练「猜想生成器」的大规模结构化数据集。这与前文OEIS、纽结数据库的成功案例形成呼应。

49:07

One thing I'm also excited about – this is a type of math that just doesn't exist yet. Right now because proving theorems is such a painful painstaking process we prove one theorem at a time or maybe two or three if you're efficient. But with AI you could imagine in the future instead of trying to prove one problem or solve one problem, you take a class of 1,000 similar problems and you say "Okay I'm going to tell your AI try to solve these 1,000 problems with this technique" and it will report back "Oh I could solve 35% of these problems with this technique. What about this technique? I can solve this percentage of problems. Well if I combine them I can do this." You could start exploring the space of problems rather than just each problem separately.

还有一件让我兴奋的事——这是一类目前根本还不存在的数学。现在因为证明定理是个如此痛苦、费力的过程，我们一次只证一个定理，效率高的话也许一次两三个。但有了 AI，你可以想象将来不是去证一个问题、解一个问题，而是拿一整类 1000 个类似的问题，你说"好，我让 AI 用这个技巧去试着解这 1000 个问题"，它会回报说"哦，用这个技巧我能解出其中 35%。那这个技巧呢？我能解出这么多百分比。如果把它们结合起来，我能做到这样。"你可以开始探索整个问题空间，而不只是一个个孤立地看问题。

全篇最著名的展望：「探索问题空间」——对1000个变体问题批量测试解题技巧并统计成功率，数学从逐个手工证明变成可做规模化实验的学科。

painstaking /ˈpeɪnzˌteɪkɪŋ/ *adj.*
费尽心力的，一丝不苟的

49:46

This is something that you just you either cannot do right now or you do over a process of decades with dozens and dozens of papers slowly figuring out what you can and can't do with various techniques. But with these tools you could really start doing mathematics on a scale which is really unprecedented. So the future is going to be really exciting Ithink. I mean they we will still also be proving theorems the old fashioned way. In fact we'll have to because we can't we won't be able to guide these AIs unless we also know how to do the things ourselves. But we'll be able to do lots of things that we can't do right now.

这是一件你现在要么根本做不到、要么得花上几十年、几十篇论文才能慢慢摸清楚的事——搞清楚各种技巧能做什么、不能做什么。但有了这些工具，你真的可以在前所未有的规模上做数学。所以我觉得未来会非常令人兴奋。当然我们还是会用老办法证明定理。事实上我们必须这么做，因为如果我们自己不会做这些事，我们就没法引导这些 AI。但我们将能做到很多现在做不到的事。

关键限定：AI不消解人类学习数学的必要性——自己不会做的人无法引导和验证AI。「老式证明」与规模化探索将长期并存。

unprecedented /ʌnˈpresədentɪd/ *adj.*
史无前例的，空前的

12 问答：数学基础与成长之路

50:28 陶哲轩 → 观众提问

Okay I think I will stop there. So thank you very much. Any questions? So we are on a tight schedule but I was told we have time for maybe three or so questions. So if people would raise hands and – there's someone over there. Thank you. Can you hear me? Thank you that was a beautiful talk. I particularly loved about formalizing mathematics but one thing that you didn't mention was Voevodsky who left algebraic geometry because he made a mistake and started formalizing homotopy type theory. I would be interested to know if you have studied this and have any comments on it.

好，我想我就讲到这里。非常感谢大家。有什么问题吗？我们的时间挺紧的，不过我被告知大概还有时间回答三个左右的问题。请大家举手——那边有位。谢谢。能听到我说话吗？谢谢您，这是一场很精彩的演讲。我特别喜欢关于数学形式化的部分，但有一点您没有提到，就是 Voevodsky——他离开了代数几何，因为他犯了个错误，然后开始做同伦类型论的形式化。我很想知道您有没有研究过这个，有没有什么评论。

Voevodsky（1966–2017）是2002年菲尔兹奖得主，因发现自己论文中的错误深受触动，晚年转向「单值基础」与同伦类型论（HoTT）的形式化事业。

51:32 陶哲轩

Right. For Voevodsky – yeah he was worried about a crisis in certain areas of mathematics including some that he created, that the proofs were so abstract and sophisticated that there was no way to verify that they were completely true. Yeah so he proposed changing the foundation of mathematics to homotopy type theory, which is more robust. Like if you change the underlying axioms of mathematics, a lot of what you prove in this theory is still true. There are proof assistant languages that are based on this sort of homotopy type theory. Lean is not actually, by design, because Lean wants to formalize a lot of traditional mathematics which is not written in this language.

是的。关于 Voevodsky——他确实担心某些数学领域会出现危机，包括他自己开创的一些领域，因为那些证明太抽象、太复杂，根本没办法验证它们是否完全正确。所以他提议把数学的基础改成同伦类型论，因为那更稳健。比如说，如果你改动数学的底层公理，在这套理论里证明的很多东西依然成立。确实有一些证明助手语言是基于这种同伦类型论的。Lean 并不是，这是有意为之，因为 Lean 想形式化大量传统数学，而传统数学并不是用这种语言写的。

同伦类型论以「类型即空间、相等即道路」为数学基础，对公理变动更稳健；Lean刻意不采用它，因为要兼容用传统语言写成的主流数学文献。

robust /roʊˈbʌst/ *adj.*
稳健的，健壮的（受扰动仍能保持性能）

52:20

I do hope in the future there will be multiple proof assistant languages with different strengths and weaknesses. One thing that we don't have right now is automatic ways to translate a proof in one language to another. That's actually one place where AI I think will be very useful. Once we have that then we can – if you have a different philosophy of your foundation of mathematics we could just hopefully translate a proof that's been formalized in one language to another and then everyone will be convinced, including Voevodsky hopefully.

我确实希望将来会有多种证明助手语言，各有各的长处和短处。我们现在缺的一样东西，就是自动把一种语言里的证明翻译成另一种语言的方法。我觉得这恰恰是 AI 会非常有用的地方。一旦有了这个，我们就可以——如果你对数学基础有不同的哲学立场，我们就有希望把一种语言里形式化好的证明翻译到另一种语言里，然后所有人都会被说服，希望也包括 Voevodsky。

陶指出的空白点：跨证明语言的自动翻译尚不存在，而这恰是AI的合适任务——持不同数学基础哲学的阵营可借此互认对方的成果。

52:54 陶哲轩 → 观众提问

Yeah so I mean there are multiple approaches to formalizing mathematics and we shouldn't just certainly fix on one particular standard just yet. Well I – okay. All right so a random process. Okay, well that will not be quite relevant to the topic of the talk, but I was recently applying for PhDs and the advice I was given by the professors was basically along the lines of "the longer the better". So it seems like there's kind of a general agreement that mathematicians need somehow to grow up to big ideas. So from that perspective, how do you think about your decision about going to university at such a young age? Did you think – how it influenced you as a mathematician and as a human being?

是的，我想说，数学形式化有多种路径，我们现在肯定还不该只固定在某一种标准上。呃，我——好的。那就随机点吧。好，我这个问题可能和演讲主题不太相关，但我最近在申请博士，教授们给我的建议基本上是"越久越好"。所以似乎大家普遍认为，数学家需要以某种方式慢慢成长到能驾驭大想法。那么从这个角度看，您怎么看待自己那么小就上大学这个决定？您觉得——它对您作为数学家和作为一个人有什么影响？

along the lines of *phr.*
大意是，类似于（引述大概内容时的常用表达）

54:03 陶哲轩

Well I was – yeah I had some very good advisers, both as a high school and undergraduate and graduate level. I mean I don't think it's a race. I mean you go to university when you're ready to go. You know you shouldn't go just because you were told you need X years or something to do this. I think it's different for different people. You know I mean – it was very important for me. I mean I went to undergraduate when I was 13 but it was at a university that was very close to where I lived so I lived with my parents and they drove quite a lot actually to the university for all my classes. If I didn't have that I don't think I would have had a good experience.

嗯，我——是的，我遇到过一些非常好的导师，高中、本科和研究生阶段都有。我觉得这不是一场竞赛。你觉得准备好了就去上大学。你不该因为别人告诉你必须花 X 年才能做这件事就去做。我觉得因人而异。对我来说这一点很重要。我 13 岁上的本科，但那所大学离我家很近，所以我住在家里，我父母其实开了很多次车送我去上所有的课。如果没有这个条件，我不觉得我会有一段好的经历。

陶13岁入读离家很近的弗林德斯大学，住在父母家由父母接送。他强调「这不是竞赛」：早慧路径成立依赖家庭支持等具体条件，不能推广为一般建议。

54:51 陶哲轩 → 观众提问 → 陶哲轩

So it really depends. I mean okay I did my university at a very young age – doesn't mean everybody should do that. Yeah it's – there's no single answer to this question. Thank you. Okay so another kind of more general question: Given that you've contributed to truly myriad mathematical fields, how do you go about choosing your next research topic and problem you want to solve? And also what's your Erdoś number? Okay well my Erdoś number is two, that's easy. But I actually don't know – I mean it – I mean early on in my career you know I had advisers who suggested problems to me. Nowadays it often just comes by serendipity. I mean math is a very social activity. You go to lots of events.

所以这真的看情况。我很小就上了大学，但这不意味着每个人都该这样。这个问题没有唯一答案。谢谢。好，那再问一个比较宽泛的问题：鉴于您在数不胜数的数学领域都有贡献，您是怎么选择下一个研究课题和想解决的问题的？另外，您的 Erdoś 数是多少？好，我的 Erdoś 数是 2，这个好回答。但我其实说不太清——我职业生涯早期是有导师给我出题。现在往往就是机缘巧合。我是说，数学是一项非常社交化的活动。你会去参加很多活动。

Erdoś数指与传奇数学家Erdoś（一生发表约1500篇论文、合作者逾500人）的合作距离。陶为2，即与Erdoś的直接合作者合写过论文。

myriad /ˈmɪriəd/ *adj.*
无数的，繁多的（书面语；亦作名词 a myriad of）

serendipity /ˌserənˈdɪpəti/ *n.*
机缘巧合，意外发现珍宝的运气

55:59 陶哲轩

So I mean after this I'm going to a math conference in Edinburgh and I'm going to talk to a lot of people in actually an area connected to this PFR conjecture thing actually. Likely I'll have some interesting math conversations and maybe some research questions come out of it. I do have some long-term projects that I would like to – you know something that I'd like to solve. But increasingly I find that it's the questions that just come up by conversation with other mathematicians that – yeah so you know I didn't know I'd be working – be talking so much about AI actually until about two years ago for instance.

比如这次之后我要去爱丁堡参加一个数学会议，而且我会和很多人聊到一个其实跟 PFR 猜想相关的领域。很可能会有一些有意思的数学对话，也许还能从中冒出一些研究问题。我确实有一些长期项目，是我想解决的东西。但我越来越发现，那些问题往往就是在和其他数学家聊天时冒出来的——所以你看，比如说直到大约两年前，我都不知道自己会这么多地谈论 AI。

PFR所属领域即加性组合。陶自述两年前还不知道自己会大谈AI——研究方向来自交谈与机缘，呼应他随后对「未来需要更多灵活性」的判断。

56:39

Yeah I think people should be sort of – you know the future is going to require more flexibility. I mean there will still be people who specialize in one topic and just one topic and be the world expert in X, but increasingly I think there'll be more and more people who will move around over time and just find interesting new mathematics every few years just by talking to other people. Okay. I think we have to move on actually. So the next speaker is Simon Coyle from XTX who will talk about the AI Math Olympiad.

是的，我觉得大家应该——未来会需要更多的灵活性。当然还是会有人专攻一个课题，只做一个课题，成为某个领域的世界级专家，但我觉得会有越来越多的人随着时间不断变换方向，每隔几年就通过和别人交流找到有意思的新数学。好。我想我们得继续往下走了。下一位演讲者是来自 XTX 的 Simon Coyle，他将谈谈 AI 数学奥林匹克。

XTX Markets是伦敦算法交易公司，2023年设立总额约1000万美元的AI数学奥林匹克奖（AIMO），悬赏达到IMO金牌水平的开源AI模型。

第二部分　核心句型 · 值得模仿的表达

1. Instead of doing X, you do Y

“Instead of having three hours to solve a problem you take months”

用 instead of + 动名词开头，一句话完成新旧对比。介绍变革、纠正误解时非常好用，比 not X but Y 更口语化流畅。

2. No matter how ..., one of ... must ...

“No matter how you color the natural numbers, one of the colors must contain a Pythagorean triple”

表达「无论怎样都必然」的全称断言，数学陈述和强调不可避免性时的标准句式。注意 no matter how 后接完整从句。

3. things that A find(s) X, B can do Y; things that A find(s) Y, B struggles with

“Things that humans find difficult, AI can do very easily sometimes, but things that humans find easy AI often struggles with”

宾语前置的平行对照结构，两个分句镜像排列，强烈突出反差。描述两种事物能力互补或错位时可直接套用。

4. With X, ... stands or falls

“With this theorem the hope that this condensed formalism can be fruitfully applied to functional analysis stands or falls.”

「成败在此一举」的书面表达，主语可以很长，谓语只有 stands or falls 两个词，形成头重脚轻的强调效果。

5. It's nowhere near ..., let alone ...

“It's still nowhere near able to solve a large majority of say math olympiad problems, let alone math research problems”

nowhere near 表「差得远」，let alone 再递进一层「更不用说」。评估现状、泼冷水降预期时的地道组合。

6. This turns out to be ...

“This turns out to be a surprisingly hard problem.”

turn out to be 表示「事实证明、结果发现」，暗含与预期相反。讲述探索过程中的意外结论时首选，常配 surprisingly 等副词。

7. in a mere + 数量

“He actually finished in a mere 12 years”

a mere 修饰数字表「区区、仅仅」，可正用（真的少）也可反讽（12年也叫少）。写作中替代 only 使数据更有态度。

8. Once you have X, you can just ...

“So once you have this black box you can just tweak each input.”

once 引导条件从句表「一旦具备…就能…」，just 强调操作简单。讲解方法论、工具价值时的高频句式。

第三部分 生词总表 · 复习用

词汇	音标	词性	释义	出现
a mere	—	phr.	仅仅，区区（表示数量小得惊人，常带反讽：86GB仍很大）	11:54
a priori	/ˌeɪ praɪˈɔːraɪ/	phr.	先验地，事先看来（拉丁语，学术写作常用）	18:23
abacus	/ˈæbəkəs/	n.	算盘	3:15
ad hoc	/ˌæd ˈhɑːk/	adj.	临时拼凑的，为特定目的而设的（拉丁语）	20:23
along the lines of	—	phr.	大意是，类似于（引述大概内容时的常用表达）	52:54
associative	/əˈsoʊʃiətɪv/	adj.	（数学）满足结合律的	9:43
asymptotic	/ˌæsɪmpˈtɑːtɪk/	adj.	（数学）渐近的	18:23
axioms	/ˈæksiəmz/	n.	公理	16:59
ballistics	/bəˈlɪstɪks/	n.	弹道学	3:56
black box	—	phr.	黑箱：只见输入输出、不知内部机制的系统	38:55
blueprint	/ˈbluːprɪnt/	n.	蓝图，详细规划；此处为Lean形式化专用工具名	28:36
brute force	—	phr.	暴力穷举（计算机术语）；此处作动词：靠蛮力算出	9:06
byproduct	/ˈbaɪˌprɑːdʌkt/	n.	副产品，附带成果	29:08
cardinality	/ˌkɑːrdəˈnæləti/	n.	（数学）基数，集合元素的个数	6:40
caveat	/ˈkæviˌæt/	n.	警示说明，附加限制条件（学术与法律常用词）	22:23
certify	/ˈsɜːrtəfaɪ/	v.	认证，证明属实	22:23
cherry-picked	/ˈtʃeriˌpɪkt/	adj.	精心挑选的（只挑对自己有利的例子，含贬义）	41:53
cluster	/ˈklʌstər/	n.	集群，一组；此处双关：既指人群也暗指计算机集群	3:56
combinatorial	/ˌkɑːmbənəˈtɔːriəl/	adj.	组合的，组合数学的	8:24
combinatorics	/ˌkɑːmbənəˈtɔːrɪks/	n.	组合数学，组合学	30:23
commutative	/kəˈmjuːtətɪv/	adj.	（数学）可交换的，满足交换律的	9:43
compiler	/kəmˈpaɪlər/	n.	编译器	33:21
complement	/ˈkɑːmpləmənt/	n.	（数学）补集，补空间；注意与 compliment（恭维）拼写区别	37:55
condensed	/kənˈdenst/	adj.	凝聚的；此处为术语 condensed mathematics（凝聚数学）	24:00
conjectured	/kənˈdʒektʃərd/	v.	猜想，推测（数学术语：提出猜想，conjecture 亦作名词）	4:40
continuity	/ˌkɑːntəˈnuːəti/	n.	连续性，延续性	2:31

controversy	/ˈkɑːntrəvɜːrsi/	n.	争议，论战	22:23
correlated	/ˈkɔːrəleɪtɪd/	adj.	相关联的，有相关性的	12:39
crowdsourced	/ˈkraʊdsɔːrst/	adj.	众包的，由大众协作完成的	26:48
decouples	/diːˈkʌplz/	v.	解耦，使分离（decouple A into B 把A拆解为B）	33:59
deduce	/dɪˈduːs/	v.	演绎，推导出	9:43
deform	/dɪˈfɔːrm/	v.	使变形；（拓扑）连续形变	36:36
derive	/dɪˈraɪv/	v.	推导出，导出（derive A from B）	19:19
disjoint	/dɪsˈdʒɔɪnt/	adj.	（数学）不相交的；互不重叠的	33:59
dual	/ˈduːəl/	adj.	（数学）对偶的；双重的	17:41
dyke	/daɪk/	n.	堤坝，拦海坝（亦拼作 dike）	7:24
executable	/ɪgˈzekjətəbl/	adj.	可执行的（executable code 可执行代码）	13:26
eyeball	/ˈaɪbɔːl/	v.	（口语）用肉眼估看，目测	40:13
facilitating	/fəˈsɪlətəɪtɪŋ/	v.	促成，使便利（facilitate）	13:26
feasible	/ˈfiːzəbl/	adj.	可行的，行得通的	9:06
fiddling	/ˈfɪdlɪŋ/	n.	反复摆弄，小修小改（fiddle with 瞎鼓捣）	21:08
first principles	—	phr.	第一性原理，最基本原理（from first principles 从头推起）	23:11
floating point arithmetic	—	phr.	浮点算术（用科学记数法形式表示不同数量级的数）	7:24
fondly	/ˈfɑːndli/	adv.	深情地，怀念地；look back on sth fondly 美好地回忆某事	1:41
functional	/ˈfʌŋkʃənl/	n.	（数学）泛函；注意此处是名词，不是形容词「功能性的」	21:08
fussy	/ˈfʌsi/	adj.	挑剔的，吹毛求疵的；此处形容语法要求苛刻的语言	45:49
generating functions	—	phr.	（数学）母函数/生成函数：组合计数的核心工具	45:01
hexagonal	/hekˈsægənl/	adj.	六边形的，六角的	17:41
high-profile	/ˌhaɪ ˈproʊfaɪl/	adj.	备受瞩目的，高关注度的	23:11
hyperbolic	/ˌhaɪpərˈbɑːlɪk/	adj.	（数学）双曲的；日常义为夸张的	37:55
incentivized	/ɪnˈsentəvaɪzd/	v.	被激励，有强烈动机做（incentivize 给予激励）	26:16
incessant	/ɪnˈsesnt/	adj.	不间断的，没完没了的（多含贬义）	21:08
incompatible	/ˌɪnkəmˈpætəbl/	adj.	不兼容的，相抵触的（incompatible with）	21:46
induct	/ɪnˈdʌkt/	v.	（数学）作归纳；induct on n 对 n 作归纳法	15:18
inevitable	/ɪnˈevɪtəbl/	adj.	不可避免的，必然的	21:46

invariants	/ɪnˈveriənts/	n.	（数学）不变量：变换下保持不变的量	37:09
iterate	/ˈɪtəreɪt/	v.	迭代，反复执行并逐步改进	46:37
let alone	—	phr.	更不用说，遑论（用于否定语境后，引出更不可能的事）	44:16
linear programming	—	phr.	线性规划（运筹学优化方法）	19:19
lingering	/ˈlɪŋgərɪŋ/	adj.	挥之不去的，迟迟不散的（lingering doubts 萦绕心头的疑虑）	25:27
longitudinal	/ˌlɑːndʒəˈtuːdnl/	adj.	纵向的（纽结理论术语：沿经线方向的）	40:13
machinery	/məˈʃiːnəri/	n.	（喻）一整套机制、方法体系；原义机器设备	31:09
magnitudes	/ˈmægnətuːdz/	n.	数量级；大小（orders of magnitude 数量级）	7:24
meridional	/məˈrɪdiənl/	adj.	经向的，子午线方向的	40:13
muse	/mjuːz/	n.	缪斯，灵感来源（源自希腊神话文艺女神）	45:01
myriad	/ˈmɪriəd/	adj.	无数的，繁多的（书面语；亦作名词 a myriad of）	54:51
nontrivial	/ˌnɑːnˈtrɪviəl/	adj.	不平凡的，有实质难度的（数学家惯用的低调说法）	38:55
nowhere near	—	phr.	远远达不到，差得远（比 not near 语气强得多）	47:29
number crunching	—	phr.	大规模数值计算（口语，crunch numbers 处理大量数据）	7:24
orthogonal	/ɔːrˈθɑːgənl/	adj.	正交的；（喻）完全不同维度的、互不相干的	42:46
outsource	/ˈaʊtsɔːrs/	v.	外包（outsource sth to sb/sth）	43:36
painstaking	/ˈpeɪnzˌteɪkɪŋ/	adj.	费尽心力的，一丝不苟的	49:07
phased out	—	phr.	被逐步淘汰（phase out 分阶段停用）	4:40
planar	/ˈpleɪnər/	adj.	平面的（planar map 平面地图/平面图）	14:32
polyhedra	/ˌpɑːliˈhiːdrə/	n.	多面体（polyhedron 的复数）	18:23
prominent	/ˈprɑːmənənt/	adj.	杰出的，声名显赫的	24:00
propositions	/ˌprɑːpəˈzɪʃnz/	n.	（逻辑）命题	10:27
refereed	/ˌrefəˈriːd/	v.	审稿（referee 学术审稿人/审稿，非「裁判」义）	22:23
robust	/roʊˈbʌst/	adj.	稳健的，健壮的（受扰动仍能保持性能）	51:32
saliency	/ˈseɪliənsi/	n.	显著性；saliency analysis 显著性分析（机器学习可解释性方法）	39:32
satisfiability	/ˌsætɪsˌfaɪəˈbɪləti/	n.	（逻辑）可满足性，SAT 即其缩写	9:06
serendipity	/ˌserənˈdɪpəti/	n.	机缘巧合，意外发现珍宝的运气	54:51
splashy	/ˈsplæʃi/	adj.	抢眼的，引人注目的，大张旗鼓的	13:26

stands or falls	—	phr.	成败系于（sth stands or falls with/on X：X成则成，X败则败）	26:16
syntax	/ˈsɪntæks/	n.	（计算机/语言学）语法	45:49
tedious	/ˈtiːdiəs/	adj.	冗长乏味的，枯燥繁琐的	15:46
transformative	/trænsˈfɔːrmətɪv/	adj.	带来变革的，颠覆性的	2:31
trivial	/ˈtrɪviəl/	adj.	（数学）平凡的，显然的；日常义为琐碎的	17:41
tweaks	/twiːks/	n.	微调，小改动（动词 tweak 亦常用）	20:23
typo	/ˈtaɪpoʊ/	n.	打字错误，笔误（typographical error 的缩略）	42:46
unprecedented	/ʌnˈpresədentɪd/	adj.	史无前例的，空前的	49:46
upper bound	—	phr.	（数学）上界	19:19
utmost	/ˈʌtmoʊst/	adj.	极度的，最大限度的（of the utmost importance 至关重要）	26:16
whack	/wæk/	v.	（口语）随手扔进、塞进；原义猛击	8:24

第四部分　理解自测 · 8 题

1. 陶哲轩在IMO上创下了什么纪录?
2. 「kilogirl」这个计量单位指什么？它反映了怎样的历史?
3. 布尔毕达哥拉斯三元组问题的结论中，关键数字7825意味着什么?
4. 开普勒猜想的证明在审稿时遇到了什么困境？这导致了什么?
5. PFR形式化项目为何能让20个陌生人在3周内高效协作?
6. 在纽结理论的案例中，机器学习和人类分别做了什么?
7. 陶哲轩描述的「探索问题空间」是怎样一种尚不存在的数学?
8. 对于「AI是否会取代数学家」，陶哲轩的立场是什么?

参考答案

1. 13岁获得金牌，是IMO史上最年轻的金牌得主（第0段），纪录至今未破。
2. 指1000名女性计算员工作一小时的计算量（第4段）。二战时computer是职业而非机器，弹道计算由大批女性用加法机完成。
3. 只要把1到7825的自然数任意分成两个颜色类，必有一类包含毕达哥拉斯三元组；而7824存在避开的染色方案（第13–14段）。
4. 《数学年刊》12位审稿人审了4年，只能给出「99%确信」、无法认证计算部分（第29段）。这促使Hales发起Flyspeck项目，用12年完成完全形式化。
5. 蓝图工具把证明拆成独立小气泡，每人只需做局部；Lean编译器自动验证，编译不过就无法提交，用机器验证替代了人际信任（第42–45段）。
6. 神经网络发现符号差可由双曲不变量预测，显著性分析锁定3个关键输入；人类据此提猜想、被机器反驳后修正，最终给出证明（第53–55段）。机器找方向，人类建立并证明联系。
7. 让AI对1000个类似问题批量尝试各种技巧并统计成功率（如「此法解决35%」），从逐个证明定理转向整体研究问题空间（第66段）。
8. 未来几年AI主要是强大的助手而非替代者；人类仍须自己掌握证明能力，否则无法引导和验证AI（第64、67段）。