苏菲拉底 THE SOPHIE LAB

本期追问

当通用模型持续变强，创业者还能在哪里建立持久优势?

能耗与数据搬运如何暗中决定 AI 产品的可能性边界?

当代码都交给智能体来写，人的稀缺能力还剩什么?

视频精读 VIDEO STUDY

Jeff Dean: The 1% Rule for Building in AI

来源：YouTube 视频（https://www.youtube.com/watch?v=CxXgV54KzpQ）

节目发布：2026-07-30 · 全文 63 段 · 页边注释 61 条 · 生词词组 112 条

好视频，值得精读

目录 CONTENTS

导读　视频总结

本期讲者

Jeff Dean　Google 首席科学家，1999 年加入当时仅 20 人的 Google，是 MapReduce、Bigtable、TensorFlow、TPU 等奠基性系统的核心设计者，现领导 Google DeepMind 的 Gemini 模型研发。

主持人　本场大会的访谈主持人，面向约 6000 名创业者与工程师听众，围绕 Jeff Dean 的过往系统与当下 AI 判断展开提问。

第一部分　全文对照 · 附页边注释与生词标注

01　开场：AI 已是初级工程师?

0:07 主持人

All right. Should we go Should we get started, Jeff? >> Sure. Sounds great. >> All right. Jeff, welcome. And again, thank you so much for being here. Especially I just got a cold and thank you for being here. >> Yeah, I'm afraid I've lost my voice. I don't normally sound quite like this, but we'll do what we can. >> So, you built map reduce, big table, tensorflow, the TPU, Gemini. We could spend a whole hour on all the things you've done, but what I love is that you're still making bold predictions in public. Last year, yes, last year in May 2025 at AI Ascent, you said that AI is at the level of a junior engineer.

好的。我们要不要开始了，Jeff？是的，听起来不错。好的。Jeff，欢迎你。再次感谢你能来。尤其是我刚感冒了，谢谢你能来。是啊，我这嗓子恐怕是哑了。我平时说话不是这个声音，不过我们尽力吧。那么，你做过 MapReduce、Bigtable、TensorFlow、TPU、Gemini。光是你做过的这些事，我们就能聊上一整个小时，但我特别喜欢的一点是，你到现在还敢在公开场合做大胆的预测。去年，对，就是去年，2025 年 5 月在 AI Ascent 大会上，你说 AI 已经达到了初级工程师的水平。

Jeff Dean 1999 年加入 Google，是 MapReduce（分布式计算框架）、Bigtable（分布式数据库）、TensorFlow（深度学习框架）、TPU（AI 专用芯片）的核心设计者，现为 Google 首席科学家。AI Ascent 是红杉资本主办的年度 AI 峰会。

0:50 Jeff Dean

That was about a year ago. It's been How close are we to that prediction? Yeah, I mean I feel like the models have been getting a lot better at sort of agent-based longer running coding tasks and it seems pretty clear that they are now actually pretty capable and depending on exactly your definition of junior engineer it seems pretty spot-on I would say. >> What did you underestimate from that prediction? I mean I think the ability to do more and more complex tasks has been growing faster than I thought. and I also think outside of coding these agent-based systems are really starting to shine in other domains and I think you know that's going to be an important trend in the future.

那大概是一年前了。到现在，我们离那个预测有多近了？是啊，我觉得这些模型在那种基于 agent 的、长时间运行的编程任务上进步了很多，而且现在看来相当明确，它们确实已经挺有能力了。当然这取决于你对“初级工程师”的具体定义，但我觉得那个说法基本上是相当准确的。那个预测里，你低估了什么？嗯，我觉得处理越来越复杂任务的能力，增长得比我想象的要快。另外我还觉得，在编程之外，这些基于 agent 的系统在其他领域也真的开始大放异彩了，我觉得……知道吧，呃，这在未来会是一个很重要的趋势。

「AI 达到初级工程师水平」是 2025 年业内争议较大的判断。Jeff 一年后自评「基本准确」，且承认低估了两点：复杂任务能力的增速，以及智能体在编程之外领域的扩散。

spot-on /ˌspɑːt ˈɑːn/ *adj.*
完全准确的，一语中的的（英式口语色彩，作表语）

1:44

>> So [snorts] give us another bold prediction. What do you think is going to be the 2027 edition? >> I think you will see a lot more automation of ML systems themselves. basically getting ML systems to improve their capabilities by running lots of experiments, breaking things down into subpros, you know, running those subpros in a tight automatic experimentation loop, putting the results together and being able to then you know, get some improved system out from that sort of fully automated problem decomposition and automated experimentation that I think that's going to be really exciting. M >> I think that also applies not just to ML but also to other fields of science and engineering. basically anything where you can have a measurable objective I think you can actually make a lot of progress these days.

那么 [吸鼻子] 再给我们一个大胆的预测吧。你觉得 2027 版本会是什么样？呃，我觉得你会看到 ML 系统本身会有更多的自动化。嗯，基本上就是让 ML 系统通过运行大量实验来提升自己的能力，把问题拆解成子问题，你懂的，把这些子问题放进一个紧凑的自动实验循环里跑，再把结果汇总起来，然后就能，呃，你懂的，得到一个改进后的系统，呃，从那种完全自动化的问题拆解和自动化实验中得出结果，我觉得这会非常令人兴奋。嗯，我觉得这不仅适用于 ML，也适用于科学和工程的其他领域。嗯，基本上任何有可衡量目标的领域，呃，我我觉得如今你都可以，呃，取得很大的进展。

这是全场的核心论题之一：让 ML 系统自动跑实验来改进 ML 系统本身。关键前提是「可衡量的目标」——后文的评估器、AlphaEvolve、递归自我改进都围绕这一条件展开。

decomposition /ˌdiːˌkɑːmpəˈzɪʃən/ *n.*
分解，拆解（problem decomposition 问题拆解）

02 「装进内存」时刻与推理硬件

2:40 主持人

>> Now let's go back to a little bit in history. Back in way back in 2001 Google search used to run on hard drives. >> Yep. And you and Sanjay did the math and realized that at some point the whole search index would finally fit in all of the RAM of all the computers you had running and you made that radical realization and you basically in few days with Sanjay shipped in production a whole new search version that worked in RAM rather than hard drive and that was the thing that got Google to be so fast. Google searches.

那我们回顾一下历史。早在 2001 年，Google 搜索还是跑在硬盘上的。是的。你和 Sanjay 算了一笔账，意识到到某个时候，整个搜索索引最终可以完全装进你们所有在运行的机器的内存里，你们做出了那个激进的判断，然后基本上在几天之内就和Sanjay 一起把一个全新的搜索版本推上了生产环境，它跑在内存里而不是硬盘上，正是这件事让 Google 搜索变得那么快。

Sanjay Ghemawat 是 Jeff 二十多年的结对编程搭档，两人合著了 MapReduce、Bigtable 等论文。2001 年把整个搜索索引从硬盘搬进内存，是 Google 搜索速度优势的起点。

shipped /ʃɪpt/ *v.*
上线，发布（科技行业用法：ship in production 推上生产环境）

3:21 Jeff Dean

>> So history tends to remix. What is the it fits the memory moment right now in 2026 that everyone in this room is still should be thinking about and designing? >> Yeah. Yeah, I mean it's a little different, but I think you're going to see more and more high performance and low energy inference hardware systems because I think everyone is now realizing that inference is the key to making you know these agent-based systems be available to more and more people and that latency is really important and that specialization of the hardware is a really key way you can make things that are more energy efficient and lower latency than more general purpose computational devices like say GPUs or TPUs >> because I think all everyone here is used to waiting for responses on models. [clears throat] So >> waiting is no fun >> master speed.

所以历史往往会重新混搭上演。现在是 2026 年，什么才是当下真正契合这个时刻的记忆（memory）问题，是在座各位仍然应该去思考、去设计的？>> 是的。是的，我是说这个话题稍微有点不一样，但我觉得你会看到越来越多高性能、低能耗的推理硬件系统，因为我觉得现在大家都意识到，推理才是让这些基于智能体的系统能够被越来越多的人用上的关键，而且延迟真的非常重要，硬件的专用化是一个非常关键的手段，能让你做出比通用计算设备更节能、延迟更低的东西，比如说 GPU 或者 TPU 这类更通用的计算设备。>> 因为我觉得在座的各位都已经习惯了等待模型的响应。[清嗓子] 所以 >> 等待可不好玩 >> 速度为王。

Jeff 给出的 2026 年「装进内存时刻」：推理（inference）专用硬件。逻辑链是：智能体普及→推理量爆炸→延迟和能耗成为瓶颈→硬件专用化是数量级改进的来源。

inference /ˈɪnfərəns/ *n.*
推理（ML 术语：模型部署后的运行阶段，与训练相对）

latency /ˈleɪtənsi/ *n.*
延迟，时延（系统响应所需时间）

general purpose /ˌdʒenrəl ˈpɜːrpəs/ *adj.*
通用的（与 specialized 专用的相对）

4:27

So you're saying what if we don't have to wait anymore? >> Yeah. I mean, I think we'll imagine what you could do with something where the latency is, you know, 50x better. >> Interesting thought. Now, what's one assumption that perhaps 6,000 people in this room hold that's already false about AI? >> Yeah. that's a good question. I mean I think probably one thing is people don't quite realize how possible it is to have you know agent-based systems that can run not just for an hour or two hours on a problem you care about but for some problem domains and with highly capable models underlying them you can get them to run for days or weeks and do really complicated tasks and I think that's you know starting some people are starting to see inklings of this but I don't think everyone has really internalized this and that's going to be really a pretty big deal.

所以你的意思是，如果我们不再需要等待了会怎样？>> 是的。我是说，想象一下如果延迟能好上 50 倍，你能用它做些什么。>> 有意思的想法。那么，这个房间里大概 6000 人所持有的、但其实关于 AI 已经不成立的一个假设是什么？>> 嗯，这是个好问题。我觉得，可能有一点是：大家还没完全意识到，基于智能体的系统其实已经可以运行得不只是在你关心的问题上跑一两个小时，而是在某些问题领域里，配上能力足够强的底层模型，你可以让它们连续跑上几天甚至几周，去完成非常非常复杂的任务。我觉得现在已经有一些人开始看到这方面的苗头了，但我不认为所有人都真正消化了这一点，而这将会是一件相当大的事。

「50 倍延迟改进」不是渐进优化，而是会改变产品形态的假设。第二个反共识判断：智能体已能连续运行数天甚至数周，多数人尚未消化这一点对任务复杂度上限的含义。

inklings /ˈɪŋklɪŋz/ *n.*
端倪，模糊的迹象（see inklings of 看到…的苗头）

internalized /ɪnˈtɜːrnəlaɪzd/ *v.*
内化，真正消化理解（把认识变成自己思维的一部分）

5:28

>> What's a particular task that you have run that has run for weeks? What was it? What did the tell what did you tell the agents to solve? Yeah, I mean I think you can tell agents to go off and implement you know completely new versions of software in different programming languages that might be you know have better safety properties or better performance properties that and then they can go off and actually do that in a you know pretty serious way. >> That's pretty cool.

>> 有没有哪个具体的任务是你跑过、而且跑了好几周的？是什么任务？你当时让智能体去解决什么问题？是的，我觉得你可以让智能体去用不同的编程语言，把某个软件完全重新实现一个新版本，可能具备更好的安全特性或者更好的性能特性，然后它们就真的可以相当认真地把这件事做出来。>> 那挺酷的。

跑数周的任务实例：用另一种语言完整重写软件（如换成内存安全或性能更好的语言）。这类任务成立的原因在第 38 段揭晓——原有代码和测试本身就是完备的规格说明。

03 餐巾纸计算与 TPU 的起源

6:00 主持人

[clears throat] Now, one thing that you've been very well known for is you're really good at napkin math. Sounds funny. So, one of the stories about you is that back in 2013 when speech recognition started to work at Google, you did the nap napkin math where if every Google user used their phone and talked to it and used the speech recognition system for three minute just three minutes a day, you found that the system requires a Google server. you would have to double the fleet which would be really expensive just to do speech translation.

[清嗓子] 你有一点特别出名，就是你非常擅长「餐巾纸计算」（napkin math）。听起来挺有意思。关于你的一个故事是，早在 2013 年，当语音识别在Google 开始真正跑通的时候，你做了个餐巾纸计算：如果每个 Google 用户都用手机跟它说话、每天使用语音识别系统哪怕只有三分钟，你发现这个系统需要的 Google 服务器数量，会让你不得不把整个服务器集群翻一倍，光是为了做语音识别就贵得离谱。

napkin math（餐巾纸计算）指用粗略数字快速估算量级。2013 年的估算：每用户每天 3 分钟语音识别，就要把 Google 整个服务器集群翻倍——这笔账直接催生了 TPU。

napkin math *phr.*
餐巾纸计算，粗略的量级估算

fleet /fli:t/ *n.*
机群，（服务器）集群（原义为舰队）

6:39 Jeff Dean

>> Yeah. >> And instead you basically built a custom ship and that was the origin story of the TPU. >> Yeah. Yeah. I mean I sort of had done you know we were starting to see really good quality results on this the sort of deep learning based speech systems speech models we were training. but they were computationally expensive compared to the old speech system but they haved the error rate. So that was like the equivalent of 20 years of advances in speech recognition in just a few months of like fiddling with the model and getting scaling it up a bit and getting better data. And so we started to get worried that if speech worked a lot better, people would use it more. And so that back of the envelope calculation was really about that like well what if people start to use speech recognition more to dictate emails or to talk to their phone or whatever. and yeah it turned out that we realized that we needed some better solution than running on CPUs at the time. And so we came up with TPUs

>> 是的。>> 于是你基本上做了一颗定制芯片，这就是 TPU 的起源故事。>> 是的，是的。我是说，当时我们开始在这些基于深度学习的语音系统、我们训练的语音模型上，看到质量非常好的结果。但跟老的语音系统相比，它们的计算开销很大，不过错误率降了一半。所以那基本上相当于语音识别领域 20 年的进展，却只用了几个月——就是摆弄一下模型、把规模稍微扩大一点、再拿到更好的数据。于是我们开始担心，如果语音效果好这么多，大家就会用得更多。所以那个粗略估算其实就是在算：如果大家开始更多地用语音识别来口述邮件、跟手机说话之类的，会怎么样。结果我们意识到，当时在 CPU 上跑确实需要一个更好的方案。于是我们搞出了 TPU，它本质上是高度专门化地

关键数据：深度学习语音模型把错误率减半，相当于压缩了 20 年的领域进展。「效果变好→使用量暴增→算力成本失控」的推演，是从质量突破倒推基础设施需求的经典案例。

halved /hævd/ *v.*
使减半（halve the error rate 把错误率降低一半）

fiddling with *phr.*
摆弄，反复调试（fiddle with the model 调模型）

back of the envelope calculation *phr.*
信封背面式估算，粗略推算（与 napkin math 同义）

dictate /ˈdɪkteɪt/ *v.*
口述（dictate emails 用语音口述邮件）

7:47

Which are sort of very specialized for essentially low precision dense linear algebra which is at the heart of nearly all of the modern machine learning algorithms we use today. And if you build a specialized chip for low precision dense linear algebra and can't do anything else that turns out to be really useful for machine learning inference even though it can't run Chrome or Word or whatever. and so that system produced a chip a couple years later that was 30 to 80 times more energy efficient than CPUs and GPUs of the day and also much lower latency like 20 to 30x lower latency >> which is incredible what the foundation that TPU has become today. No way you would have predicted that TPU would be so foundational now with transformer architecture which was invented way later before you actually invented the TPU. Yeah, I mean that's sort of why we built a general purpose linear algebra system, which is what a TPU is really.

做低精度稠密线性代数，而这正是我们今天使用的几乎所有现代机器学习算法的核心。如果你造一颗只能做低精度稠密线性代数、别的什么都干不了的专用芯片，结果它对机器学习推理特别有用——尽管它跑不了 Chrome 或者 Word 之类的东西。所以那个项目在几年后产出了一颗芯片，能效比当时的 CPU 和 GPU 高出 30 到 80 倍，延迟也低得多，大概低 20 到 30 倍。>> 这太不可思议了，看看 TPU 今天已经成为多么重要的基础设施。你当时绝不可能预料到，TPU 在 Transformer架构出现之后会如此关键——而这个架构是在你发明 TPU 很久之后才被提出来的。是的，这某种程度上就是我们为什么造了一个通用的线性代数系统，TPU 其实就是这么个东西。

TPU 的设计取舍：只做低精度稠密线性代数、其他什么都不做，换来对当时 CPU/GPU 30~80 倍的能效和 20~30 倍的延迟优势。专用化牺牲通用性换数量级收益。

at the heart of *phr.*
处于…的核心

low precision *phr.*
低精度（ML 硬件术语：用更少比特表示数值以换取速度和能效）

8:48

Because we knew ML algorithms were still evolving and you didn't want to over specialize, but you wanted to specialize enough that you got the dramatic performance benefits of we could have very big multiplier units. we could have you know high-speed memory we could have high-speed interconnect or later TPUs that like brought many chips to bear on the same problem efficiently and u you know we've continued to scale those up and improve their performance for over many generations now >> incredible napkin math so >> what's good >> napkins are good >> so actually what's a good napkin math that everyone here who wants to be a future founder should run tonight to potentially build something as consequential as the TPU.

因为我们知道机器学习算法还在演进，你不想过度专用化，但又想专用到足以获得显著的性能收益——我们可以放很大的乘法单元，可以有高速内存，可以有高速互连；后来的 TPU 还能把非常多的芯片高效地投入到同一个问题上。我们也一直在把它们的规模往上扩、把性能往上提，到现在已经很多很多代了。>> 了不起的餐巾纸计算。>> 挺好的 >> 餐巾纸是个好东西 >> 那么实际上，对在座每一位想成为未来创业者的人来说，今晚可以做一个什么样的餐巾纸计算，去有可能造出像 TPU 那样有分量的东西?

> 「适度专用化」是核心工程哲学：算法还在演进，过度专用会押错方向；但要专用到能放大乘法单元、配高速内存和互连。TPU 押注的是「线性代数」这一层抽象，而非某个具体算法——所以 Transformer 后来才能跑在它上面。

interconnect /ˌɪntərkəˈnekt/ *n.*
互连（芯片间的高速连接通道）

brought many many chips to bear on *phr.*
bring…to bear on：把（资源、力量）投入运用于

consequential /ˌkɑːnsəˈkwenʃəl/ *adj.*
影响重大的，举足轻重的

9:39

>> Yeah, I mean, it's always hard to say. [clears throat] I think, think about what problems you see in whatever it is you're thinking about, what bottlenecks you see, and are there very different ways of thinking of the solutions to some of those problems that would get you, you know, an order of magnitude or two orders of magnitude better performance or capability or whatever it is. you know, because sometimes if you just squint at a problem and you think about not necessarily being anchored on exactly how that problem is solved today, but how you would solve it from first principles, you can come up with really good ideas that are, you know, maybe not what other people are thinking about.

>> 嗯，这个总是很难说。我觉得，想一想在你正在琢磨的那个领域里，你看到了哪些问题、哪些瓶颈，以及对其中某些问题，有没有非常不一样的解决思路，能让你在性能、能力或者别的什么指标上，拿到一个数量级甚至两个数量级的提升。因为有时候，如果你眯着眼睛看一个问题，不一定被今天这个问题是怎么解决的所束缚，而是思考如果从第一性原理出发你会怎么解，你就能想出一些非常好的点子，而这些可能正是别人没在想的。

第一性原理方法论：不被「今天这个问题是怎么解的」锚定，寻找能带来一到两个数量级改进的不同解法。「数量级」是 Jeff 判断一个方向是否值得做的反复出现的标尺。

bottlenecks /ˈbɑːtlneks/ *n.*
瓶颈

order of magnitude *phr.*
数量级（十倍为一个数量级）

squint at /skwɪnt/ *phr.*
眯起眼看；引申为换个角度、模糊细节地重新审视问题

anchored on /ˈæŋkərd/ *phr.*
被…锚定、束缚（认知科学中的锚定效应）

first principles *phr.*
第一性原理（抛开现有方案，从基本事实出发推导）

04 新版必知数字：以能量为单位

10:25 主持人

>> That's a good tip. [clears throat] >> No. for everyone here who doesn't know, years ago, Jeff wrote a very famous list called the latency numbers. every engineer should know and these are numbers around for example how long a cache miss takes disk seek a network package traveling let's say from California to Netherlands lots of numbers like this about distributed systems and systems engineering [clears throat] >> and it's been sort of taped and become the bible for a lot of distributed systems engineers >> okay yeah >> now fast forward that list is up for an update give us the AI edition for now 2026.

>> 这是个好建议。[清嗓子] >> 对了，可能有人不知道，很多年前 Jeff 写过一份非常有名的清单，叫做「每个工程师都该知道的延迟数字」。这些数字包括比如说一次缓存未命中要多久、磁盘寻道要多久、一个网络数据包从加州传到荷兰要多久，诸如此类关于分布式系统和系统工程的很多数字。[清嗓子]>> 这份清单基本上被贴在墙上，成了很多分布式系统工程师的圣经。>> 好的，是的 >> 现在快进到今天，这份清单该更新了，给我们讲讲 2026 年的 AI 版本吧。

《Latency Numbers Every Programmer Should Know》是 Jeff 广为流传的清单，列出缓存未命中（约纳秒级）、磁盘寻道（毫秒级）、跨洲网络往返等关键数字，是分布式系统工程师的量级直觉基础。

cache miss *phr.*
缓存未命中（数据不在高速缓存中、须访问更慢存储的情形）

disk seek *phr.*
磁盘寻道（磁头移动到目标数据位置的操作，毫秒级耗时）

11:09 Jeff Dean

>> Yeah, I mean I think if you looked at what is important in AI systems these days, you would want to know things like the bandwidth between you know your main memory system on your accelerator to the onchip memory to the you know the multiplier unit or whatever. You want to know how much energy does it take to do a single multiplier operation. you know what is the interconnect bandwidth between chips and how much does that how many chips can you connect with that bandwidth and then if you go beyond that domain like what is the fall off in network bandwidth when you need to talk to 10,000 strips instead of instead of 500 or something I think these are all really important numbers to learn and really affect how you think about solving particular kinds the problems.

>> 是的，我觉得如果你看看当下 AI 系统里什么是重要的，你会想知道这些数字：比如加速器上主存到片上内存、再到乘法单元之间的带宽是多少。你会想知道做一次乘法操作要消耗多少能量。还有芯片之间的互连带宽是多少，用这个带宽你能连接多少颗芯片；再往外走一层，当你需要跟 10,000 颗芯片通信而不是 500 颗时，网络带宽会掉到什么程度。我觉得这些都是非常重要、值得去掌握的数字，而且它们会实实在在地影响你思考如何解决某类问题。

AI 版「必知数字」：HBM 到片上内存的带宽、单次乘法的能耗、芯片间互连带宽、从 500 卡扩到 1 万卡时网络带宽的衰减。掌握这些数字才能对系统设计做出正确取舍。

bandwidth /ˈbændwɪdθ/ *n.*
带宽（单位时间可传输的数据量）

fall off *n./phr.*
衰减，下降幅度（the fall off in network bandwidth 带宽的衰减）

12:04 主持人

>> H [clears throat] and one interesting thing that I've heard you talk about is that nowadays the unit that you measure everything is energy. >> Yeah. >> You pointed out that doing a calculation or math costs about one pico. but moving the data and doing data IO costs thousand times that. >> Yeah. Just bringing it in from HPM on an accelerator into the processor so it can actually compute on it. Yep. That gap kind of quietly decides what products are possible and how these algorithms in AI are built. So what are the kinds of problems that founders keep calling model problems but are in fact actually energy or data IO problems?

>> 嗯 [清嗓子] 还有一件有意思的事，我听你讲过：现在你衡量一切的单位是能量。>> 是的。>> 你指出过，做一次计算大约消耗一皮焦（pico）。但搬运数据、做数据 IO 的成本是它的一千倍。>> 是的。就是把数据从加速器上的 HBM 搬进处理器，好让它真的能拿来计算。没错。这个差距悄无声息地决定了哪些产品是可行的、AI 里这些算法是怎么设计的。那么有哪些问题，是创业者一直称之为「模型问题」，但实际上是能耗或者数据 IO 问题?

全场最关键的数字：一次乘法约 1 皮焦，而把数据从 HBM（高带宽内存）搬进处理器的能耗是它的约一千倍。很多被称作「模型问题」的东西，本质是这个能耗差决定的系统问题。

12:51 Jeff Dean

Yeah, I mean I think the example you raised of a thousandx difference in bringing mo moving data versus actually computing on it in terms of energy is a pretty significant one and it shapes a lot of aspects of what we do in machine learning. because if you didn't have that thousandx difference then you know you wouldn't have to do batching but you have to do batching of you know many examples or maybe many tokens at once in order to amortize that data movement [clears throat] so that you can you know not pay a thousandx slowdown but pay a 1000x divided by batch size energy cost. and you know for really low latency batching is not really very good. so I think these [clears throat] kinds of things and the energy behind various decisions in the computer hardware we use really affects a lot of decisions we make in building higher level systems.

是的，我觉得你提到的那个例子——搬运数据和真正在数据上做计算之间在能耗上有一千倍的差距——是相当重要的一点，它塑造了我们在机器学习里做的很多事情。因为如果没有这一千倍的差距，你其实就不需要做批处理（batching）了；但现在你必须把很多样本、或者很多 token 一次性打成一批，才能摊薄那个数据搬运的开销 [清嗓子]，这样你付出的就不是一千倍的代价，而是一千倍除以批大小的能耗代价。而对于要求极低延迟的场景，批处理其实并不太好。所以我觉得这类事情，以及我们所用的计算硬件里各种取舍背后的能耗因素，确实会深刻影响我们在构建上层系统时做出的很多决策。

batching（批处理）存在的根本原因被揭示：把千倍的数据搬运开销摊到整个批次上，代价变成「一千除以批大小」。推论：低延迟场景下批处理天然吃亏——这正是推理硬件要重新设计的原因。

amortize /ˈæmərtaɪz/ *v.*
摊销，摊薄（把一次性开销分摊到多次操作上）

batching /ˈbætʃɪŋ/ *n.*
批处理（把多个样本打包一起计算以提高硬件效率）

13:51

>> A very concrete example is just how training models is done. There's this whole concept of batching the data sets and running epochs. That's basically people perhaps may confuse that as a model problem, but it's really a systems data IO problem, right? >> Yeah. Yeah. I mean, you have to assemble batches to get better efficiency in your hardware. You know, ideally you might do batch size one training, but you know, it's not as not as good in terms of efficiency. So people use re pretty large batches these days.

>> 一个非常具体的例子就是模型训练的做法。有一整套关于把数据集打成批次、跑若干个 epoch 的概念。人们可能会误以为那是个模型问题，但它其实是一个系统层面的数据 IO 问题，对吧？ >> 是的，是的。我是说，你必须把数据攒成批，才能让硬件效率更高。理想情况下你可能想做批大小为 1 的训练，但那在效率上就没那么好了。所以现在大家用的批次都挺大的。

反直觉之处：训练中的 batch、epoch 这些「ML 常识」并非算法本质要求，而是硬件效率的妥协。理想的批大小为 1 的训练，只是在当前能耗结构下不划算。

epochs /ˈepəks/ *n.*
轮次（ML 术语：完整过一遍训练数据集为一个 epoch）

14:26

>> Do you think it's possible for I know you're well known for taking off on a long week or weekend and coming up with this brilliant solution. Is there such things of Jeff going and working on it for a couple weeks and [clears throat] getting batch size equals one training done. >> Yeah, I've been thinking more about inference actually. So I think inference is a pretty interesting problem because you do want very low latency. You know training you don't necessarily need incredibly low latency. and I think there's a lot of room for specializing hardware more for inference than we are today.

>> 你觉得有没有可能……我知道你以出名的一点是，你会抽出一个长周末或者一周跑去琢磨，然后想出某个绝妙的解法。会不会出现 Jeff 花几个星期，[清嗓子]把批大小等于 1 的训练给搞定了？ >> 嗯，其实我最近想得更多的是推理。我觉得推理是个相当有意思的问题，因为你确实需要非常低的延迟。而训练不一定需要极低的延迟。我觉得在为推理做更多硬件专用化这件事上，还有很大的空间，比我们今天做的要多。

Jeff 当前的关注点从训练转向推理：训练可以容忍延迟，推理不能。这与第 4 段的预测呼应——推理硬件专用化的空间远未被挖掘。

15:02

>> What are some of those interesting things that are on inference that you're really thinking a lot about? >> I mean just trying to minimize data movement. trying to think about incredibly low precision operations and maybe not supporting lots and lots of different kinds of precisions. if you feel like you have a good answer for what kinds of precision you need, maybe just build that into the hardware and not much else. which I think it brings down to a core [clears throat] analogy I heard from famous computer scientists that really the whole process of u AI is a big compression problem because in order to have the data to be f fully lossy and compress it and then restore it you basically need to understand it. Yeah, I mean if you truly understand the data, you should be able to compress it really well >> and now transformer architecture is basically one of the ways that has turned out to work really well.

>> 在推理方面，有哪些有意思的东西是你现在思考特别多的？>> 嗯，就是尽量减少数据搬运。还有考虑极低精度的运算，以及也许不去支持特别多种不同的精度格式。如果你觉得自己对「需要哪些精度」已经有了好答案，那也许就把它直接做进硬件里，别的就不要了。这让我想到一个核心的类比，我从一些著名计算机科学家那里听来的：其实整个 AI 的过程就是一个巨大的压缩问题，因为要把数据完全（有损地）压缩、再还原出来，你基本上必须理解它。是的，我是说如果你真正理解了数据，你就应该能把它压缩得非常好。>> 而现在 Transformer 架构基本上是被证明效果非常好的方式之一。

「AI 即压缩」是压缩理论中的经典观点：能把数据无损/有损地压缩再还原，等价于理解了数据。Transformer 可以被看作一种被证明极其有效的压缩机制。

lossy /ˈlɔːsi/ *adj.*
有损的（lossy compression 有损压缩，与 lossless 无损相对）

analogy /əˈnælədʒi/ *n.*
类比（a core analogy 一个核心类比）

05 上下文工程：人人可用的杠杆

16:07 主持人

>> Yeah. Yeah, I would say >> working pretty well so far. >> Good work by my colleagues. [laughter] >> Yes. Now let's zoom out a bit. AI progress used to mean just better models. You could had more data trainer models with bigger parameters. But increasingly in the last years or so, it's everything around the model. Not just the model size and number of parameters or more data. It's everything around things like retrieval tools, memory, agent tools, and it might kind of get consolidated into what people call context engineering, right?

>> 是的，是的，我会说 >> 到目前为止表现相当不错。>> 这得归功于我同事们的出色工作。[笑声] >> 是的。现在我们把视角拉远一点。以前 AI 的进步基本就意味着更好的模型：更多数据、更大参数量的模型。但在过去这一两年里，越来越多的进步来自模型周边的一切，而不只是模型大小、参数量或者更多数据。是围绕模型的那一整圈东西，比如检索工具、记忆、智能体工具，这些可能会被整合成人们所说的「上下文工程」（context engineering），对吧？

话题转向：AI 进步的来源从「更大的模型、更多数据」转移到模型周边的系统——检索、记忆、工具调用，统称「上下文工程」（context engineering）。

retrieval /rɪˈtriːvəl/ *n.*
检索（RAG 中的 R，即从外部知识库取回相关信息）

consolidated /kənˈsɑːlɪdeɪtɪd/ *v.*
整合，归并（get consolidated into 被整合为）

16:42 Jeff Dean

[clears throat] >> Yeah. I mean I think the model is really only one piece of what you're trying to do which is build an overall system that can solve really interesting problems and that involves you know a model that knows how to use various tools. It maybe knows how to retrieve relevant information, maybe has a, you know, a history of other information that it has retrieved for past problems and it can put information into the context of the of the model. And the nice thing about that is that information is really clear to the model, unlike the training data the model was trained on where it's all kind of like trillions of tokens stirred together into a soup of hundreds of billions or trillions of parameters, but it's all less clear than the actual context that the model sees directly for this particular problem or uses use case. And then I think being able to understand what tools are available, which ones are going to help me solve the help the model solve this next you

[清嗓子] >> 是的。我觉得模型其实只是你想做的事情中的一块，你真正要做的是构建一个整体的系统，能解决非常有意思的问题；这里面涉及一个知道如何使用各种工具的模型。它可能知道怎么去检索相关信息，可能保存着以往为其他问题检索过的信息的历史记录，然后它可以把信息放进模型的上下文里。而这样做的好处是，这些信息对模型来说非常清晰明确，不像模型训练时用的训练数据那样，是几万亿个 token 被搅在一起，糊成了几千亿甚至上万亿参数的一锅汤，但是这一切都不如模型针对这个特定问题或用例直接看到的实际上下文那么清晰。然后我认为，能够理解有哪些工具可用，哪些工具能帮我解决——帮模型解决问题的下一个阶段，如何把一个问题拆解成一连串的

关键对比：训练数据是「几万亿 token 搅成几千亿参数的一锅汤」，模糊而间接；而放进上下文的信息对模型清晰直接。这解释了为什么检索和上下文管理能显著提升表现。

stirred together into a soup

phr.

搅成一锅汤（比喻信息混杂交融、无法分辨来源）

17:47

Know phase of the problem, how to decompose a problem into a sequence of tool calls. Maybe trying multiple approaches to solve the problem and seeing which ones work and being able to evaluate that. you know this is the whole you know orchestration of complex agent and multi-agent systems that I think is going to be more and more important and super exciting times I would say >> and I think the fun thing about this particular problem domain set is actually something that everyone in this room can actually do because before to train a model you needed incredible amount of resources incredible amount of access of to GPUs and data but for context engineering everyone here could do you have you just need the API to something like Gemini and then work on your own setup for your own retrieval your own tool calls and etc. So how does what are some tips for everyone here? How does everyone get better at and become exceptional at context engineering? Yeah, I mean I think [clears throat]

工具调用。也许尝试多种方法来解决问题，看看哪些行得通，并且能够对此做出评估。你知道，这就是复杂智能体和多智能体系统的整体编排，我认为它会越来越重要，可以说是非常令人兴奋的时代。>> 我觉得这个特定问题领域有意思的一点是，它其实是在座每个人都能做的事，因为过去要训练一个模型，你需要难以想象的资源、难以想象的 GPU 和数据的获取渠道，但对于上下文工程，在座的每个人都能做——你只需要拿到像 Gemini 这样的 API，然后去打磨你自己的一套东西，你自己的检索、你自己的工具调用等等。那么，对在座各位有什么建议？大家怎样才能变得更擅长，甚至在上下文工程上做到出类拔萃？是啊，我想（清嗓）一个很好的办法就是去用

对创业者的民主化意义：训练模型需要巨额算力和数据，而上下文工程只需要一个 API。这是普通团队今天就能参与的杠杆点。

orchestration /ˌɔːrkɪˈstreɪʃən/ *n.* 编排，协调调度（原指管弦乐配器，引申为对多组件系统的统筹）

decompose /ˌdiːkəmˈpoʊz/ *v.* 分解，拆解（decompose a problem into… 把问题拆成…）

18:50

A really good way to do it is to use these models and sort of harnesses and tools and so on to try to solve problems and then some sometimes you can actually see where the models are failing. And often you can actually make the model work better and succeed at that kind of problem by not just adjusting the model parameters which is hard to do from the outside but from you know creating better guidelines for the model you know writing skills for the model to know how to use different tools that would be incredibly useful for solving this particular class of problem. And I think as you do that, you end up on this kind of improving self-improving of the setup that you're trying to use to solve things. and you know that's a really good way to get better at understanding what additional information the model would want in order to become more capable.

这些模型，以及各种脚手架（harness）、工具等等，去尝试解决问题，然后有时候你能真切地看到模型是在哪里失败的。而且往往你其实可以让模型表现更好、在那类问题上取得成功，做法不是去调整模型参数——那从外部很难做到——而是为模型编写更好的指引，为模型编写 skills，让它知道如何使用各种工具，这些对解决这一类特定问题会非常有用。我认为随着你这样做，你最终会进入一种不断改进、自我改进的循环，改进你用来解决问题的这套配置。呃，而且你知道，这真的是一个很好的途径，让你更好地理解模型还想要什么额外信息才能变得更有能力。

「skills」指为模型编写的结构化指引文档，教它如何用工具、按什么流程做事。改进路径不是调模型参数（外部做不到），而是观察失败点、迭代指引——形成自我强化的改进循环。

harnesses /ˈhɑːrnɪsɪz/ *n.*
（智能体的）运行脚手架、驾驭框架（原义为马具，AI 行业指包裹模型的执行环境）

19:46

>> Can you give an example of some context engineering you personally have done? I don't know skills you wrote tools that really made a huge different in your in your workflow. Yeah, I mean I guess Sanjay and I were working a few weeks ago and we you know we often do some amount of like performance improvement for very low-level libraries and we have a microbenchmark library we've written at Google where you can write microbenchmarks of how long different kinds of operations take or how long does it take to populate this data structure whatever and sometimes those data structures are used on millions of processes across Google. So, it's actually pretty important to make sure they're high performance. And so, you can write microbenchmarks. but then without an agent-based system, what you usually do is you measure what the current performance is on some benchmarks you care about. You make some modifications to improve the performance you hope. Then you rerun the benchmarks, see where things improved.

>> 你能举个例子吗，说说你个人做过的上下文工程？比如你写过的 skills、真正对你的工作流产生巨大影响的工具。是啊，我想想，几周前 Sanjay 和我在做一些工作，你知道我们经常会为一些非常底层的库做性能优化，我们在 Google 写了一个微基准测试库，你可以用它来编写微基准，测量各种操作耗时多久，或者填充某个数据结构要多久，诸如此类，而有时候这些数据结构会被 Google 内部数百万个进程使用。所以确保它们的高性能其实相当重要。因此你可以写微基准。嗯，但如果没有基于智能体的系统，你通常的做法是：先测量你关心的那些基准当前的性能，然后做一些你希望能提升性能的修改，接着重新跑基准，看看哪里有改善。

微基准测试（microbenchmark）测量单个操作的耗时。Google 的底层数据结构运行在数百万个进程上，微小的性能差异会被放大成巨大的资源成本，所以值得极致优化。

microbenchmark /ˌmaɪkroʊ ˈbentʃmɑːrk/ *n.*
微基准测试（测量单个小操作性能的测试）

populate /ˈpɑːpjuleɪt/ *v.*
填充（populate a data structure 向数据结构填入数据）

20:48

You run a maybe a broader set of benchmarks, measure the cache footprint of things. And so we wrote a skill that basically taught the model how to do most of those things in var in various sequences so that it could actually you know do self-improving benchmark measurement benchmark improve you know code changes measure the performance improvement and then iterate on that and that seemed to work pretty well for some kinds of problems. And it really just is us giving the approach we would use as people to the model in a form that it could use.

嗯，你可能会跑一套更广泛的基准，测量缓存占用之类的。所以我们写了一个 skill，基本上就是教模型如何按各种顺序完成这些事情，这样它就能真正做到自我改进——跑基准测量、改进基准、做代码修改、测量性能提升，然后不断迭代，而这在某些类型的问题上效果似乎相当不错。这本质上就是我们把人会采用的方法，以模型能用的形式交给它。

把「测基准→改代码→再测→迭代」这套人类专家工作流写成 skill 交给智能体，本质是把隐性经验显式化。这是第 2 段「自动化实验循环」的一个已实现的小型样本。

cache footprint *phr.*
缓存占用（程序运行时占用高速缓存的量）

iterate /ˈɪtəreɪt/ *v.*
迭代，反复改进（iterate on that 在此基础上不断迭代）

21:23

>> Wow, that seems very impressive. So you're saying you have this skill that if someone got access to it, it could do perform optimizations like Jeff Dean. Seems like the world would love this and is worth infinite amount of money to someone have access to this. >> Oh. we actually published a document maybe a few months ago called performance hints that Sanjay and I wrote that's like a 30-page document about you know various kinds of performance tricks and some people have taken that and then given it in summarized form to various models and seen that they that model can now get you know better at per reasoning about performance issues in code.

>> 哇，这听起来太厉害了。所以你是说，你有这么一个 skill，如果有人能拿到它，就能像 Jeff Dean 一样做性能优化。感觉全世界都会想要这个，能拿到它对某些人来说值无限多的钱。>> 哦。其实我们几个月前发布过一份文档，叫《performance hints》，是 Sanjay 和我写的，大概 30 页，讲的是各种各样的性能技巧，有些人把它拿去做成摘要形式喂给各种模型，然后发现模型在推理代码中的性能问题上确实变得更强了。

《Performance Hints》是 Jeff 和 Sanjay 公开发布的约 30 页性能优化经验文档。有人将其摘要后喂给模型，模型推理代码性能问题的能力确实提升——印证了 skills 的价值。

06 长程智能体为何跑偏与解法

22:01 主持人

>> So you heard it all here. You could actually get your own optimize your own code like Jeff Dean if you take this paper that you published when performance hints. Yep. >> It's all free available, so you should all try it. >> Very cool. >> Yeah. >> Now, you're talking about agents. everyone here is probably building one or built one at some point. And I'm sure everyone has seen your agent go off the rail at perhaps like step 30 or 40. Like agents are great for like up to step, I don't know, 10 or something and then gets shaky at step 50. What do you think is the constraint today? Is it like context evaluators or just errors that compound because it's basically a openloop system?

>> 所以大家都听到了。如果你拿上这份你们发布的《performance hints》，你其实真的可以像 Jeff Dean 一样优化自己的代码。没错。>> 而且都是免费公开的，所以大家都该去试试。>> 非常酷。>> 是的。>> 那我们来聊聊智能体。在座各位大概都在做智能体，或者曾经做过。我相信每个人都见过自己的智能体在大概第 30 步或第 40 步的时候跑偏。就是说智能体在大概前十步左右表现很好，到第 50 步就开始不稳了。你认为今天的瓶颈是什么？是上下文、评估器，还是因为它本质上是个开环系统而导致误差不断累积？

主持人点出智能体的普遍痛点：前 10 步可靠，到三五十步就跑偏。候选原因：上下文不足、缺评估器、开环系统（无反馈校正）导致误差复利式累积。

go off the rail *phr.*
（原为 go off the rails）脱轨，失控跑偏

shaky /ˈʃeɪki/ *adj.*
不稳的，摇摇欲坠的（gets shaky 开始不稳）

compound /kəmˈpaʊnd/ *v.*
（误差、利息）复利式累积放大（errors that compound 不断累积的误差）

openloop /ˈoʊpən luːp/ *adj.*
（open-loop）开环的（控制论术语：无反馈校正的系统）

22:41 Jeff Dean

>> Yeah, I mean obviously we want agents to be able to run for very long periods of time because that's how they're going to solve more and more complicated problems. but as you as you observe today, you know, they sometimes stop working after, you know, 10 10 interactions with the tools and so on. and sometimes that's because the model is trying to do something it doesn't have a lot of experience doing. So it's been trained on a whole set of things and as soon as you get a little bit off the distribution of things it knows how to do then like most machine learning models it will you know its performance will suddenly will start to degrade and the farther you get off the comfort zone of what it knows how to do the more likely it is to not work as well. so there's a bunch of things you can do. So one is you know give the model skills and hints that kind of tend to keep it in the sort of more brightly lit path of things it does know how to do. I think you know having

>> 是啊，显然我们希望智能体能长时间持续运行，因为只有这样它们才能解决越来越复杂的问题。但正如你今天观察到的，它们有时候在和工具交互十来次之后就不工作了。嗯，有时候是因为模型在尝试做一件它没什么经验的事。它在一整套事情上被训练过，而一旦你稍微偏离它知道怎么做的那个分布，那么就像大多数机器学习模型一样，它的表现会突然开始下降，而且你越是偏离它熟悉的舒适区，它就越可能表现不佳。嗯，所以你可以做很多事情。一是给模型 skills 和提示，让它倾向于待在它确实知道怎么做的那条“灯光明亮的路”上。嗯，我觉得，

Jeff 的诊断：偏离训练分布（off the distribution）是 ML 模型的通病，偏得越远退化越快。对策一是用 skills 把智能体约束在「灯光明亮的路」——它确实会做的事情范围内。

off the distribution *phr.*
偏离（训练数据的）分布（ML 术语，即 out-of-distribution）

degrade /dɪˈgreɪd/ *v.*
退化，性能下降

23:43

Multi– aent systems where you have multiple agents trying different approaches and you can evaluate you have maybe another model or another agent that's evaluating which ones of those seem promising is another way to kind of in some sense search the path of pos search the space of possible solutions and stick to the ones that seem most promising and discard the ones that didn't seem to work or maybe that went off the rails. or whatever. and that's a very useful general technique is you know inference time compute to perform search over plausible ways of solving the problem that can get much higher performance or much more reliability in longunning agent flows.

采用多智能体系统，让多个智能体尝试不同的方法，然后你可以评估——也许有另一个模型或另一个智能体来评估其中哪些看起来有希望——这也是一种在某种意义上搜索可能解法空间的方式，坚持那些看起来最有希望的，丢掉那些行不通的，或者跑偏了之类的。嗯，这是一个非常非常有用的通用技巧，也就是用推理时的算力去搜索各种可能的解题路径，这能带来高得多的性能，或者在长时间运行的智能体流程中带来高得多的可靠性。

对策二：多智能体尝试不同路径，另设评估者筛选——本质是用推理时算力（inference-time compute）对解空间做搜索。这是提升长程任务可靠性的通用技术，与 o1/深度思考类方法同源。

plausible /ˈplɔːzəbl/ *adj.*
貌似可行的，说得通的（plausible ways 可能可行的路径）

discard /dɪˈskɑːrd/ *v.*
丢弃，舍弃

24:30

>> How are some ways you implemented this particular workflow for your agents internally? Yeah, I mean we have you know harnesses and then we have a whole set of skills particularly in the internal Google development environment. We have skills so that the agents can know how to use lots of our internal tooling for coding or for code reviews or for you know measuring performance or you know fetching log files. And those are just skills that you can add to make the base model more capable even though it hasn't necessarily been trained on exactly the way that you know Google internal engineers would fetch log files from our you know proprietary system with the right kind of skill definition you can actually get it to work >> and that improves the usefulness of the agents. Now let's talk about where startups can win. This section is one that I personally care a lot about because also everyone here in this room needs to decide what to build in the future of your future founder. So

>> 你们内部是通过哪些方式为自己的智能体实现这套工作流的？是啊，我们有脚手架，然后我们有一整套 skills，尤其是在 Google 内部的开发环境里。我们有 skills，让智能体知道如何使用我们大量的内部工具，比如写代码、做代码评审，或者测量性能、抓取日志文件。而这些就是你可以加上去的 skills，让基础模型变得更有能力，即使它并不一定被专门训练过 Google 内部工程师是怎么从我们的专有系统里抓取日志文件的——只要有合适的 skill定义，你其实就能让它跑起来。>> 这也提升了智能体的实用性。那我们来聊聊创业公司能在哪里取胜。这一部分我个人非常关心，因为在座的各位都需要决定未来要做什么，作为未来的创业者。Google 的特点是，你们从处理器到产品

落地细节：Google 内部为代码评审、性能测量、抓取日志等专有工具写了成套 skills。基础模型没见过这些专有系统，但有了 skill 定义就能用——说明能力缺口可以靠文档补齐。

proprietary /prəˈpraɪəteri/ *adj.*
专有的，私有的（proprietary system 公司内部专有系统）

tooling /ˈtuːlɪŋ/ *n.*
工具链，成套工具（internal tooling 内部工具）

07 通用模型阴影下的创业机会

25:37

The thing about Google is you co-design everything on the system from the processors to the products. which are the layers that someone like Google would keep building and compounding being better and where does a two three person team can still win? >> Yeah, I mean I think obviously Google and our Gemini models and our hardware infrastructure are really trying to build very general models that can do almost anything. But in a lot of cases that means that we don't have a lot of attention on particular domains where perhaps a really well-designed surface that and maybe a model and set of skills or maybe a specialized model that isn't in sort of a general mix of things that our models do well can actually have a significant advantage because you can build something delightful and you know really high accuracy. really high quality for a domain that you are really passionate about. And I think that's where you know the two or three people in a room building that they're

对整个系统做协同设计。那么，哪些层是像 Google 这样的公司会持续构建、不断复利式变强的，而两三个人的小团队又能在哪里取胜？>> 是啊，我觉得显然 Google、我们的 Gemini 模型以及我们的硬件基础设施，都在努力构建能做几乎所有事情的非常通用的模型。但在很多情况下这也意味着我们不会在某些特定领域投入很多注意力，而在那些领域里，一个设计得非常好的交互界面，加上也许一个模型和一套skills，或者一个专门的模型——那种不在我们模型擅长的通用范畴里的——其实可以有显著的优势，因为你可以为一个你真正热爱的领域做出一个令人愉悦、准确率非常高、质量非常高的东西。我认为那就是两三个人在一个屋子里、做他们真正兴奋的事情能有优势的地方。嗯，但我也要提醒一句，通用模型确实在

创业机会的定位逻辑：Google 做「几乎什么都能做」的通用模型，必然在特定领域投入的注意力有限。精心设计的产品界面＋领域 skills 或专用模型，就是小团队的立足点。

compounding /kəmˈpaʊndɪŋ/ *v.*
复利式积累，不断叠加增强

surface /ˈsɜːrfɪs/ *n.*
（产品的）交互界面、承载面（行业用法：product surface）

delightful /dɪˈlaɪtfl/ *adj.*
令人愉悦的（硅谷产品话语：delightful product 体验极佳的产品）

26:49

Really excited about can have an advantage. but I would also caution that the general models are definitely getting better at a broader and broader range of things. So you have to figure out, you know, is that thing you're working on, is that going to be a durable thing or do you think the models at the forefront are going to get better at that in the next six months or 12 months or is it something they're not going to be able to do for a couple years or three years? And you know, you want to weigh that as you're as you're deciding what to work on.

越来越广的范围上变得更强。所以你得想清楚，你正在做的这件事，它是不是一件能长期站得住的事，还是说你觉得最前沿的模型在未来六个月或十二个月内就会在这件事上变强，又或者这是它们两三年内都做不到的事？在决定要做什么的时候，你需要权衡这一点。

关键的风险提示：通用模型的能力边界在持续外扩。选方向前要判断——这件事是能长期站住（durable），还是前沿模型 6~12 个月内就会追上？

caution /ˈkɔːʃən/ *v.*
告诫，提醒注意（I would caution that… 我要提醒的是…）

durable /ˈdʊrəbl/ *adj.*
持久的，经得起时间考验的（a durable thing 能长期站得住的事）

at the forefront *phr.*
处于最前沿（models at the forefront 前沿模型）

weigh /weɪ/ *v.*
权衡，掂量（weigh that 权衡这一点）

27:22

>> So let's dive deeper into this. So the general models of course you're going to keep working on and keep making them all better. And how should the audience reason about what are those areas that it doesn't I mean h how should founder think about things to pick on and work on. >> Yeah. I mean I mean the most important thing is to pick something you're super excited about and want to build and you think would be useful in the world, right? So if you do that's you're already way ahead than if you wake up and you're like oh I don't really want to do this or whatever or you're going to build something that is actually not that useful to the world or to many people. so I think that's the number one selection criteria I try to apply for what problem should I work on next. second, I think you want to look at what the current more general models can do in that problem domain, right? You can you can test them with like, are they able to do this thing very well? And if they're

>> 那我们再深入一点。通用模型你们当然会继续做，继续让它们全面变强。那么听众该如何判断哪些是它做不到的领域？我是说，创业者该如何思考该挑选并投入哪些方向？>> 是啊。我觉得最重要的是挑一件你特别兴奋、特别想做，并且你认为对世界有用的事，对吧？如果你做到了这一点，你就已经领先很多了，比起你早上醒来想“唉我其实不太想做这个”之类的，或者你要做的东西其实对世界、对很多人都没什么用。嗯，所以我认为这是我在选择下一个该做什么问题时会用的第一条标准。第二，我认为你要看看当前更通用的模型在那个问题领域能做到什么，对吧？你可以去测试它们，比如它们能不能把这件事做得很好？如果它们完全做不到，那大概是个好信号。如果它们能做一部分但做得不太好，那

28:26

Completely failing, that's probably a good sign. If they're kind of able to do some of it but not very well, that's maybe not a great sign because that's a probably a sign that the capability is starting to be present in those models and with more training data or larger scale models or whatever it's likely to get better. So you know look for something where the model succeeds 0% or 1% of the time not 20%. >> How do you find those? I mean are those things effectively out of distribution from the training set and what exactly is the problem shape that fits that?

可能不是什么好信号，因为这大概说明这项能力已经开始在这些模型里出现了，随着更多训练数据、更大规模的模型等等，它很可能会变好。所以，你要找的是模型成功率为 0% 或 1% 的事情，而不是 20%。>> 那怎么找到这些？我是说，这些东西是不是本质上处于训练集分布之外？到底什么样的问题形态符合这一点？

反直觉的筛选标准：找通用模型成功率 0%~1% 的问题，而不是 20%。20% 意味着能力已开始涌现，会随规模扩大被碾平；完全失败反而说明存在结构性缺口。

29:04

>> Yeah, I mean I think sometimes it's a product that you build that might have access to particular kind of data that the underlying model might not the a general model. So it might be you're building something to help users organize all their own personal information and the model won't necessarily have access to that. And so there you can have a big advantage because all of a sudden your model has visibility or your product has visibility into important data. it could be some incredibly hard problem where if you get the right training data and you can train a more specific model than a general purpose one, you can actually do that in a very affordable way. you maybe it doesn't take that much compute to train a niche model for this particular problem, but you can get something that's highly accurate. That can sometimes be a really good building block for solving a important problem that is maybe not handled very well by the general model.

>> 是啊，我想有时候是这样：你做的产品可能能接触到某种特定的数据，而底层模型、通用模型接触不到。比如说，你在做一个帮助用户整理他们全部个人信息的产品，而模型不一定能访问这些数据。那样你就能有很大的优势，因为突然之间你的模型、或者说你的产品，能看到重要的数据。嗯，也可能是某个极其困难的问题，如果你能拿到合适的训练数据，训练一个比通用模型更专门的模型，你其实能以非常低的成本做到这一点。也许为这个特定问题训练一个小众模型并不需要那么多算力，但你能得到一个准确率非常高的东西。这有时候会成为一个非常好的构建模块，用来解决通用模型可能处理得不太好的重要问题。

两条护城河：一是数据访问权——你的产品能看到通用模型看不到的私有数据；二是用合适的训练数据低成本训练小型专用模型，在窄域达到通用模型达不到的准确率。

niche /nɪtʃ/ *adj./n.*
小众的（细分市场）；a niche model 小众专用模型

building block *phr.*
构建模块，基础组件

30:04

>> I think that's interesting. I think there are basically two paths. The first path is a little bit funny is you guys are organizing the world's information. >> Yeah, >> that's probably kind of well covered. >> Yeah. but organizing your personal information that's open which is funny. >> Yeah. >> And then the second path you talked about more specialized models in certain domains. Can you tell us more about what are some of these domains? >> Yeah. I mean I think like if you look at my colleagues work on say alpha fold that was a very specific model for protein folding and it was highly successful and was able to really handle that domain quite well so that all of a sudden you now have this amazing tool and model that can give you answers to questions about proteins and their structure really effectively but it's not a general model it's a very specific one and there are other I domains where that kind of approach can work really well. maybe in material science or chip design or things like

>> 我觉得这很有意思。我觉得基本上有两条路。第一条路有点好笑，就是你们已经在整合全世界的信息了。>> 是的。>> 那块大概已经被覆盖得差不多了。>> 对。但整理你的个人信息这块还是开放的，这挺有意思的。>> 是的。>> 然后第二条路，你提到了某些领域里更专门的模型。你能多讲讲有哪些这样的领域吗？>> 好的。我觉得，比如你看我同事们做的 AlphaFold，那就是一个非常专门的模型，用于蛋白质折叠，它极为成功，能够把那个领域处理得相当好，以至于突然之间突然之间，你就有了这样一个了不起的工具和模型，能非常有效地回答关于蛋白质及其结构的问题。但它不是一个通用模型，而是一个非常专门的模型。还有其他一些领域，这种思路也能发挥得很好。比如材料科学、芯片设计之类的领域，它能让你借助

AlphaFold 是 DeepMind 的蛋白质结构预测模型，2020 年基本解决了困扰生物学 50 年的蛋白质折叠问题，其负责人 Hassabis 和 Jumper 因此获 2024 年诺贝尔化学奖。它是「窄而深」路线的标杆。

protein folding *phr.*
蛋白质折叠（由氨基酸序列预测三维结构的生物学难题）

08 写清规格与养成选题品味

31:07 主持人

That will enable you to leverage the capabilities of a very accurate but niche model to do things that are hard today. >> That's a good example. So if some of you find a problem that's similar shape like alpha fold could be a good problem to work on. Now let's assume you found a problem to work on. We're going to talk a bit about how do you become a AI native founder? How do you really become good at it? you in the past said that managing a fleet of agents, it's like 50 or 100 agents is all about writing really good crisp design docs or specs.

一个非常精准但很小众的模型的能力，去做今天很难做到的事。>> 这是个好例子。所以如果你们中有人找到形态类似的问题，像 AlphaFold 这样的，可能就是个值得投入的好问题。那现在假设你已经找到了要做的问题。我们来聊聊，怎么才能成为一个 AI原生的创始人？怎么才能真正做得好？你以前说过，管理一支智能体大军，比如 50 个或 100 个智能体，关键就在于写出非常好、非常清晰的设计文档或规格说明。

话题转向个人能力：管理 50~100 个智能体的关键，是写出清晰的设计文档和规格说明（spec）——这是主持人引用 Jeff 此前的观点。

crisp /krɪsp/ *adj.*
清晰利落的（crisp design docs 简明清晰的设计文档）

specs /speks/ *n.*
规格说明（specification 的缩写，软件行业常用）

31:45 Jeff Dean

>> And how do people get good at that? What do those look like? Yeah, I mean I think it's you'll have a lot more success when working with your virtual agents if you can clearly specify what it is you want. And the clearer you are on what it is you want, the more the agent will have sort of guidelines and sort of rules of, you know, an outline of what it is trying to accomplish. whereas if you don't specify very much stuff, the agent has to sort of infer what it is you meant. And in many cases, it might infer things that are different than what you imagined. So we've always told computer scientists from the very beginning that really it's really important to specify what it is, what's the software that you're writing is trying to accomplish before then going and writing it. And so now we actually have agent-based systems that can do the writing, but the importance of specifying what it is you want has actually gone up because before you'd be handing it off to a very intelligent human who maybe has context

>> 那大家怎么才能把这件事做好？这些文档长什么样？是的，我觉得，你在跟虚拟智能体协作时，如果你能清楚地说明自己想要什么，成功率会高得多。你对自己想要什么说得越清楚，智能体就越能有一套指引和规则，也就是它要完成的目标的大致轮廓。而如果你什么都不怎么说明，智能体就得去猜你到底是什么意思。很多情况下，它猜出来的东西跟你设想的不一样。所以我们从一开始就一直告诉计算机科学家，非常重要的一点是，在动手写之前先明确说清楚：你要写的这个软件到底要完成什么。而现在我们确实有了能替我们写代码的智能体系统，但把你想要什么说清楚的重要性其实是上升了。因为以前你是把活交给一个非常聪明的人，他可能了解背景，

反直觉之处：AI 会写代码了，「说清楚你要什么」的重要性反而上升了。以前的接收方是有上下文、会追问的人类；智能体只能从你的描述里推断，说得越含糊，猜错越多。

infer /ɪnˈfɜːr/ *v.*
推断，推测（infer what it is you meant 推断你的本意）

32:50

Or can ask you follow–up questions. and agents can sometimes do that, but I think clear specifications is a really good idea. and to give you an example of a use of a coding agent that works extremely well is you can ask today's models to translate software from one computer language to another very effectively because in that case you actually have a incredibly detailed specification. You have the whole software that says what the system is supposed to do. And so if you have a Python implementation of something and you want a Go implementation of it, you know, that is something that the models seem incredibly capable at doing these days because you can it can sort of take all the tests that are in Python, make sure they pass in the Go version, translate the tests to Go, you know, compare behavioral differences between the implementations until there aren't any and be you know, highly effective because that spec is so clear.

也可以追问你问题。智能体有时候也能这么做，但我觉得清晰的规格说明真的很重要。举个编程智能体用得极好的例子：你可以让今天的模型把软件从一种编程语言翻译成另一种，效果非常好，因为在那种情况下你其实有一份极其详尽的规格说明。你有整套软件，它本身就说明了这个系统应该做什么。所以如果你有某个东西的 Python 实现，想要一个 Go 的实现，这就是现在的模型看起来极其擅长的事情，因为它可以把Python 里的所有测试拿过来，确保它们在 Go 版本里也能通过，把测试翻译成 Go，然后比较两个实现之间的行为差异，直到没有差异为止，而且效率非常高，因为那份规格说明太清晰了。

语言间翻译软件（如 Python 转 Go）为何是智能体的最佳用例：原有代码＋测试套件本身就是一份极其详尽的规格说明，模型可以通过测试和行为对比不断收敛到零差异。

handing it off *phr.*

hand off：移交，交接（把任务交给别人）

33:51

Hm. Now let's assume now every founder gets good at running hundreds of agents at the same time and all the code is written for them by the agents. What becomes the scarce skill? >> Yeah, I mean I think it's really having incredibly good taste in what you ask your agents to work on, right? That is the crux of you know from my background a research problem. You know, a researcher can have all the tools and all the techniques, but often most of the battle is what problem are you gonna spend your time on? And if you pick the problem well and you succeed in solving it, that's way better than if you, you know, delightfully execute a research investigation into a rather boring problem. And so that high level wisdom of what to work on, I think is incredibly important. And I think models are not necessarily going to be that good at it. So you're going to have people steering a lot of AI assisted computation in order to accomplish great things and more quickly. but that essence of

嗯。那现在假设每个创始人都很擅长同时驱动上百个智能体，所有代码都由智能体替他们写好了。这时候什么会成为稀缺的能力？>> 是的，我认为关键在于你对「让智能体去做什么」有极好的品味，对吧？这就是从我的背景来说，一个研究问题的核心。研究者可以拥有所有的工具和技术，但往往胜负手在于：你要把时间花在哪个问题上。如果你选对了问题并且成功解决了它，那比你把一个相当无聊的问题的研究过程执行得再漂亮都要好得多。所以「该做什么」这种高层次的智慧，我觉得极其重要。而且我觉得模型未必会特别擅长这一点。所以会是人来掌舵，去引导大量 AI 辅助的计算，从而更快地完成伟大的事情。但你希望模型去做什么，这个本质

当执行被智能体接管，稀缺技能变成「选题品味」（taste）。Jeff 以研究为例：工具和技术人人可得，胜负手在于把时间花在哪个问题上——漂亮地执行一个无聊的问题毫无价值。

scarce /skers/ *adj.*
稀缺的（the scarce skill 稀缺技能）

crux /krʌks/ *n.*
症结，最关键之处（the crux of… …的核心）

steering /ˈstɪrɪŋ/ *v.*
掌舵，引导方向

35:02

What it is you want your models to do is the key thing you should focus on. >> So let's talk a bit more about taste because it gets talked a lot about right now in this current era with agent coding. How do you exactly build taste and do that? I mean, yeah, that sounds so esoteric. How do you make it concrete? >> Yeah, I mean it is a difficult thing. It's not like there's a measurable objective of taste in a lot of cases. I think some of it is from experience. You know, working on a lot of different problems in the past kind of teaches you about what kinds of problems might be interesting in the future or what kinds of things might be just barely possible by cobbling together these previous approaches and then some open problems you might have to work on in order to get to something kind of magical or you know, highly useful. another way you can get more experience for yourself is to just write down a bunch of things you think might be important in the next 12 months. And

才是你该聚焦的关键。>> 那我们再多聊聊品味，因为在现在这个智能体编程的时代，大家特别爱谈这个。你到底要怎么培养品味、怎么做到？我是说，这听起来太玄了。怎么把它落到实处？>> 是的，这确实是件难事。很多情况下，品味并没有一个可衡量的目标函数。我觉得其中一部分来自经验。过去做过很多不同的问题，会让你多少懂得未来哪类问题可能有意思，或者哪些事情把以前的方法拼一拼就刚好可能做成，以及为了做出某种近乎神奇、或者说非常有用的东西，你还得攻克哪些开放问题。另一种给自己积累经验的方式是，把你认为未来 12 个月里可能重要的一堆事情写下来。也许你会挑其中一件去做，

品味培养法一：经验——做过足够多不同的问题，才知道哪些事「把旧方法拼一拼刚好能成」。法二：写下你认为未来 12 个月重要的事情清单。

esoteric /ˌesəˈterɪk/ *adj.*
玄奥的，只有内行才懂的

cobbling together /ˈkɑːblɪŋ/ *phr.*
cobble together：草草拼凑，攒出来

36:10

Maybe you pick one of them to work on, but go back and evaluate in 12 months of these other things, which ones actually seemed important or which ones did other people in the world go out and create and which ones did they did not seem to do yet. that can give you a lot more samples for your own sort of taste creation capability. and that's an important skill to have. >> I think a third way we were talking earlier was doing very crazy thought experiments. >> Oh yeah, that's another good way. I mean I think sometimes it's good to not take as a given things that most people seem to take as a as a given. so I was doing a crazy thought experiment with some colleagues the other day about you know for 60 years the whole silicon chip design industry design and fabrication industry have you know done tremendous work to make smaller and smaller scale transistors that are very low error rate right like because what the assumption that we want is that every chip we

但 12 个月后回头去评估其他那些事：哪些真的显得重要，哪些被世界上的其他人做出来了，哪些看起来还没人做。这样能给你的品味养成能力提供多得多的样本。这是一项很重要的技能。>> 我们之前聊到的第三种方式，是做非常疯狂的思想实验。>> 对，那也是个好办法。我觉得有时候，把大多数人视为理所当然的事情别当成理所当然，是很有价值的。前几天我就和几位同事做了一个疯狂的思想实验，过去 60 年里，整个硅芯片设计与制造行业做了大量的工作，把晶���管做得越来越小，而且错误率非常低。因为我们的假设是：同一设计制造出来的每一块芯片，都应该和其他芯片完全一样。

预测清单的用法：12 个月后回看，哪些真的重要、哪些被别人做出来了、哪些没人做——用真实世界给自己的判断力校准，扩充「品味」的训练样本。法三：疯狂的思想实验，即质疑人人视为理所当然的假设。

take as a given *phr.*
视为理所当然，当作默认前提

fabrication /ˌfæbrɪˈkeɪʃən/ *n.*
（芯片）制造（chip fabrication，简称 fab）

37:25

Manufacture of the same design should be identical to every other chip. >> You don't want any bits to flip. Everything >> no bits should flip. There's all kinds of things you there's all kinds of error margins built into you know memories have ECC memory these days. you know at the at the macro scale we don't make that assumption when we're building large scale distributed systems right we build reliable large scale distributed file systems out of unreliable parts right like individual discs can fail but your data should be safe and so we have mechanisms at a higher level to enable us to have you know three copies of the data on three different machines and three different racks so that if any rack switch or individual ual machine or disk fails, you still have your data. We have read Solomon encoding techniques. but we don't seem to do this at a really extreme level in the sort of transistor level scale of the technology we're working on. So what would h basically a interesting thought

>> 你不希望有任何比特翻转。所有东西 >> 不应该有比特翻转。当然也有各种误差余量设计，比如现在的内存有 ECC 内存。但在宏观尺度上，我们建大规模分布式系统时并不做这种假设，对吧？我们用不可靠的部件搭出可靠的大规模分布式文件系统，单块磁盘可以坏，但你的数据必须是安全的。所以我们在更高层次上有各种机制，让我们能把数据的三份副本放在三台不同机器、三个不同机架上，这样任何一个机架交换机、单台机器或磁盘挂了，你的数据还在。我们还有里德–所罗门编码之类的技术。但我们似乎没有在晶体管这个层级的尺度上，把这套思路做到很极端的程度。所以一个有意思的思想实验就是：如果你试图用那种一天可能出 20 次错的晶体管

背景：芯片行业 60 年的默认假设是晶体管必须几乎零错误。而分布式系统恰好相反——用会坏的磁盘搭出可靠的文件系统（三副本、ECC 内存、Reed–Solomon 纠删码），可靠性在更高层实现。

error margins *phr.*
误差余量（为容错预留的设计空间）

racks /ræks/ *n.*
机架（数据中心中安放服务器的柜架）

38:30

Experiment is what would happen if you tried to build a system out of transistors that might have you know 20 errors per day. >> Oh my god. rather than one every million years, right? That would be a very different design point and might be might enable you to do really interesting things in the fabrication side of things. You have very different kind of design methodologies because if you want to get a signal from here to there, you and you have these super unreliable transistors. You might have very different ways of signaling. You might send it along multiple redundant paths in order to make sure that it gets along one of them. and I think that would be a pretty interesting set of thought experiments. I'm not saying we should go do this, but you know, that's the kind of thing where you do want to, you know, occasionally question assumptions. Now, oftentimes these thought experiments don't work out because there are very good reasons that, you know, for the last 50 years, we've done this thing this way and not

来搭系统，会发生什么？>> 天哪。而不是一百万年才出一次错，对吧？那会是一个截然不同的设计点，也许能让你在制造这一端做出真正有意思的事情。你会有非常不同的设计方法论，因为如果你要把一个信号从这里传到那里，而你手上的晶体管极其不可靠，你可能就得用很不一样的信号传递方式。你可能会沿多条冗余路径同时发送，以确保它至少能从其中一条到达。我觉得那会是一组挺有意思的思想实验。我不是说我们真该去做这件事，但这就是那种你确实应该偶尔去质疑一下假设的例子。当然，这类思想实验很多时候是不成立的，因为过去 50 年我们之所以这么做而不是那么做，是有非常充分的理由的。但每隔一段时间重新审视一下总是好的。

思想实验本体：如果晶体管一天错 20 次而不是一百万年错一次，会怎样设计系统？答案可能是冗余路径传信号等全新方法论，并可能解放制造端。Jeff 强调多数思想实验会失败——现状往往有充分理由，但定期质疑假设本身有价值。

redundant /rɪˈdʌndənt/ *adj.*
冗余的（redundant paths 冗余路径；工程中冗余是容错手段）

question assumptions *phr.*
质疑假设（question 作动词：对…提出质疑）

39:28

That way. But it's good to kind of revisit those every so often. >> That is so wild. Well, I mean, it's starting to rhyme a lot with neuromorphic computing or the human brain and how nature works. >> I mean, exactly like signals in our brain are not especially reliable from getting one place to another. And so, I think in brains when there are really important things you need to get from one place to another, there are multiple pathways that enable you to sort of do that. >> What is in I mean, you have such an impressive career. What is one of these crazy assumptions that you threw out of the window that actually built a consequential system in the past?

>> 太疯狂了。我是说，这跟类脑计算、跟人脑、跟大自然的运作方式开始有很多相通之处了。>> 没错，我们大脑里的信号从一处传到另一处其实并不特别可靠。所以我觉得在大脑里，当有非常重要的信息需要从一处传到另一处时，会有多条通路来帮你完成这件事。>> 那么，你的职业生涯这么令人钦佩，有没有哪一个你曾经抛弃掉的疯狂假设，最后真的造出了一个影响深远的系统?

这一思路与神经形态计算（neuromorphic computing）相通：大脑的神经信号传递本身不可靠，重要信息靠多条通路冗余传输——自然界早已采用「不可靠部件+高层容错」的方案。

rhyme with /raɪm/ *phr.*
与…押韵；引申为与…惊人地相似（源自「历史不重复但押韵」）

neuromorphic /ˌnʊroʊˈmɔːrfɪk/ *adj.*
神经形态的（neuromorphic computing 模仿大脑结构的计算）

09 思想实验如何催生 MapReduce

40:09

>> Yeah, I mean I guess >> that worked out actually. >> Yeah, I mean I think well TPUs is a good example like being able to specialize hardware for a very niche >> problem domain before that problem domain seemed as important as it is today is one thought experiment. you know I think the origin of map produce is another good example. So we had worked the you know my Sanjay and myself and a number of other colleagues had worked on various iterations of the crawling and indexing system at Google and you know we'd sort of written lots of hand parallelized code with lots of checkpointing to make sure it would be robust and reliable if it was running on a 100 computers or a thousand computers and some of those died. but that code tended to be intermixed with the actually relatively simple thing you often were trying to do like I just want to like look at all the contents of all the web pages and then compute on the side a mapping from URL to you know what language is this page

>> 嗯，我想…… >> 而且真的成功了的。>> 是的，我觉得 TPU 就是个好例子——在某个问题领域还没有像今天这么重要之前，就能为这个非常小众的领域做专用硬件，这算是一个思想实验。另外我觉得 MapReduce 的由来也是个好例子。当时我、Sanjay 还有其他一些同事，做过 Google 抓取和索引系统的好几个版本迭代。我们写了大量手工并行化的代码，加上大量检查点，以确保在 100 台或 1000 台机器上跑、其中一些机器挂掉时，系统仍然稳健可靠。但那些代码往往和你真正想做的、其实相当简单的事情混在一起，比如我只是想看看所有网页的内容，然后顺带算出一个从 URL到「这个页面的正文是什么语言」的映射。而这些逻辑会被大量

MapReduce 起源：Google 早期爬取索引系统里，真正的业务逻辑（如判断网页语言）很简单，却被大量手写的并行化、检查点、容错代码淹没——问题在于两类代码纠缠在一起。

checkpointing /ˈtʃekpɔɪntɪŋ/ *n.* 检查点机制（定期保存状态以便故障后恢复）

intermixed /ˌɪntərˈmɪkst/ *v.* 混杂在一起（be intermixed with 与…交织混杂）

41:16

In the text of this page. and it would get obscured by all this kind of other code for parallelization and reliability. And so we sort of remembered our training in functional languages and realized we could squint at those problems and developed this map produce abstraction that you could have above this implementation and then below the implementation you could put all the checkpointing and reliability mechanisms into that lower level library that everything could then build on. And so that became a hugely successful way of dealing with very large scale computations at Google in a robust and reliable way. From that thought experiment of like well if we squint at it could we find lots of problems that fit into this abstraction.

用于并行化和可靠性的其他代码淹没。于是我们想起了自己在函数式语言方面的训练，意识到可以换个角度去看这些问题，就发展出了 MapReduce 这个抽象：在实现之上是这层抽象，而在实现之下，你可以把所有检查点和可靠性机制放进那个底层库里，让所有东西都构建在它之上。于是这就成了 Google 处理超大规模计算的一种极其成功的方式，既稳健又可靠。而这一切都来自那个思想实验：如果换个角度看，我们能不能找到很多适合这个抽象的问题。

解法来自函数式编程的 map/reduce 抽象：业务逻辑在抽象之上，容错机制沉到抽象之下的底层库。这是「关注点分离」的经典案例，也是思想实验（换个角度看能否覆盖大量问题）的直接产物。

obscured /əbˈskjʊrd/ *v.*
被遮蔽，被淹没（get obscured by 被…掩盖）

abstraction /æbˈstrækʃən/ *n.*
抽象（层）（计算机科学核心概念：隐藏底层细节的接口）

robust /roʊˈbʌst/ *adj.*
稳健的，健壮的（能承受故障和异常）

10 AI 造 AI：加速科学循环

42:03

>> That's impressive. So this thought experiment led you to create map reduce. >> Yeah. Awesome. >> Now let's go back to you talked a bit about your interest right now working on a lot of customized hardware. So right now alpha chip >> lays out chips. Now you also got alpha evolve that proposes solutions, >> evaluates them and keeps all the ones that work. Seems like you're starting to build all these system that can compound and build AI that builds AI. >> Yeah. I mean I think more generally there's a there's this sort of the foundation of the scientific method of you propose an experiment you implement what you need to run the experiment and you evaluate the experiment and then you get results from that and I think there are more and more problems that are now possible to implement where that whole loop of running you know not just a few experiments but running many experiments because you're able to automate that loop and make the latency of that loop extremely low is going to

>> 太厉害了。所以这个思想实验让你们创造了 MapReduce。>> 是的。太棒了。>> 我们回过头来。你刚才提到你现在对做大量定制化硬件很感兴趣。现在有 AlphaChip >> 来做芯片布局。你们还有 AlphaEvolve，它会提出解决方案、>> 对方案做评估，并保留所有有效的方案。看起来你们正在构建这样一整套能够复利叠加、用 AI 造 AI 的系统。>> 是的。我觉得更普遍地看，这其实是科学方法的基础：你提出一个实验，实现运行这个实验所需的东西，评估这个实验，然后得到结果。我觉得越来越多的问题现在可以这样来做了——整个循环不只是跑几个实验，而是跑非常非常多的实验，因为你能把这个循环自动化，并把循环的延迟压到极低，这会变得

AlphaChip 用强化学习做芯片布局（已用于 TPU 设计），AlphaEvolve 用进化式方法提出、评估并保留有效的算法方案。主持人点破其共性：AI 造 AI 的复利循环。Jeff 将其一般化为科学方法的自动化——提出实验、实现、评估、汇总。

lays out *phr.*

lay out：布局，排布（芯片设计中的布局规划）

43:08

Be really important. It's going to enable us to tackle you know lots of different problem domains in science and engineering and machine learning model design itself and also in engineering tasks like designing chips. And so if you can actually do those things in an automated way and have some orchestration framework that can take very high level objectives and break them down into subpros and each of those subpros can be one of these automated loop that is exploring the best way to solve that sub problem and then a orchestration framework that can put together subpros solutions into a you know the overall solution for the higher level problem that's going to be really impactful and it's really important and I think it'll enable us to do you know accelerate machine learning progress it'll enable us to accelerate science and enable us to accelerate engineering and I think that's going to be amazing >> that sounds awesome I mean it sounds like a lot of fields basically where you

非常非常重要。它会让我们能攻克科学、工程、机器学习模型设计本身，以及像芯片设计这样的工程任务中，许许多多不同的问题领域。所以如果你真能用自动化的方式做这些事，并且有某种编排框架，能接收非常高层次的目标，把它们拆解成子问题，而每个子问题都可以交给这样一个自动化循环去探索最佳解法，然后再由编排框架把子问题的解拼装成更高层问题的整体解决方案——那将会非常有影响力，也非常非常重要。我觉得它能让我们加速机器学习的进展，能让我们加速科学，也能让我们加速工程，我觉得那会非常了不起。 >> 听起来太棒了。我是说，听起来很多领域都是这样：只要你能有很好的评估器，或者接近那些可以被

> 完整愿景：编排框架接收高层目标→拆成子问题→每个子问题交给自动化实验循环→再拼装成整体方案。适用于科学、工程、ML模型设计、芯片设计等一切可评估的领域。

44:12

Can have very good evaluators and maybe adjacent to basically things that can be formally verified right those are ripe for AI systems that can self-improve Yeah, I think in a lot of cases sometimes your evaluators need to be made much faster. Mhm. >> So as an example, my colleagues did some work maybe a decade ago on some problems in quantum chemistry where you're trying to understand the properties of a particular molecule and you can you know generate some molecule configuration and then you want to understand what properties it has. And so you can run a very computationally intensive density functional theory simulator which is something that might take like a night of computation to tell you the answer for one thing. but what my colleagues did was take a bunch of output from those simulation runs the input molecule configurations and the outputs of the expensive simulator and then use it to train a neural approximation to the simulator. So this is now a validation device, but instead of it taking a

形式化验证的东西，那这些领域就非常适合能自我改进的 AI 系统。是的，我觉得很多情况下，你的评估器需要被做得快得多。嗯。>> 举个例子，大概十年前我的同事们做过一些量子化学方面的工作，你要理解某个特定分子的性质，你可以生成某种分子构型，然后想知道它有什么性质。于是你可以跑一个计算量非常大的密度泛函理论模拟器，可能要算一整晚才能告诉你一个结果。但我的同事们做的是，拿一批那些模拟运行的输出——输入的分子构型，以及那个昂贵模拟器的输出——然后用这些数据去训练一个神经网络近似模型，模拟器。所以这现在是一个验证工具，但它不需要花一整晚，他们做出来的东西快了 30 万倍。

评估器提速的经典案例：密度泛函理论（DFT）是量子化学的高精度模拟方法，算一个分子性质要一整晚。同事用昂贵模拟器的输入输出对训练神经网络近似模型。

computationally intensive *phr.*
计算密集的，算力开销极大的

neural approximation *phr.*
神经网络近似（用神经网络拟合昂贵计算过程的替代模型）

45:22

Night, they made something that was 300,000 times faster. >> Wow. >> And nearly as accurate as running the full scale simulator. So now that completely changes how you would do science, right? Because now you have 10 million things to screen. you know, you could do that while you go to lunch rather than it being a six-month endeavor where you could try to scrape together enough compute to run all these simulations. And I think there's a lot of room in a lot of domains for much faster validation models, possibly learned valu validation models that can you know get you a approximation to the true answer much more rapidly.

>> 哇。>> 而且几乎和跑完整规模的模拟器一样准确。所以这就彻底改变了你做科学研究的方式，对吧？因为现在你有 1000万个东西要筛选，你去吃个午饭的工夫就能搞定，而不是变成一个长达六个月的大工程，还得想办法东拼西凑足够的算力来跑完所有这些模拟。我觉得在很多领域里，都有很大的空间去做更快的验证模型，可能是学习出来的验证模型，能让你更快得到接近真实答案的近似结果。

结果：30 万倍加速、精度接近原模拟器。质变在于科研范式——筛选 1000 万个候选分子从「凑半年算力的大工程」变成「吃顿午饭的工夫」。快速评估器是自动化科学循环的前提。

screen /skriːn/ *v.*
筛选（药物/材料研发术语：screen candidates 筛选候选物）

scrape together *phr.*
东拼西凑，勉强凑齐（scrape together enough compute 凑够算力）

endeavor /ɪnˈdevər/ *n.*
（费力的）事业，大工程（a six-month endeavor 一项半年的大工程）

46:04

And that changes how those experimental loops can be thought of and how quickly you can go around those loops. >> What are some of the spaces and problems that you're super excited that this super sped up scientific method is going to solve or achieve? What particular problems or spaces? >> Yeah, I mean I think well clearly machine learning itself is one, right? So can we have a model that is able to recursively self-improve itself by running lots of experiments and you know if you think about how models are improved today in large research teams you know what usually happens is people think of some ideas they run a bunch of smallcale experiments they see if those small scale experiments worked out well if so they take the most promising ones of those they try them at larger scale and that gets then evaluated and then the results get integr ated together into you know a new recipe for your model. but I think there's no you know real impediment to making that be a much more automated loop where the model itself

这就改变了我们思考那些实验循环的方式，以及你能多快地跑完这些循环。>> 有哪些领域和问题是你特别期待这种大幅加速的科学方法能够解决或实现的？具体是哪些问题或领域？>> 是的，我是说，我觉得呃，很明显机器学习本身就是其中之一，对吧？我们能不能有一个模型，能够通过跑大量实验来递归地自我改进？你想想今天在大型研究团队里模型是怎么被改进的，通常的情况是，大家想出一些点子，跑一堆小规模实验，看看这些小规模实验效果好不好，如果好的话，就挑出其中最有希望的几个，在更大规模上试，然后进行评估，接着把结果整合到一起，形成你模型的新配方。嗯，但我觉得没有什么呃真正的障碍阻止我们把这变成一个自动化得多的循环——由模型自己决定去探索，或者说

递归自我改进的具体图景：今天大团队改进模型靠人想点子→小规模实验→放大验证→整合进配方。Jeff 认为这个循环没有本质障碍不能自动化，优化目标是「每单位算力的发现量」。

recursively /rɪˈkɜːrsɪvli/ *adv.*
递归地（recursively self-improve 递归自我改进）

impediment /ɪmˈpedɪmənt/ *n.*
障碍，阻碍（no real impediment to… 没有实质障碍）

recipe /ˈresəpi/ *n.*
配方；ML 行业指训练方案的完整配置组合

11 蒸馏论文被拒与坚持的教训

47:15 主持人

Decides it's going to explore or maybe with a nudge from some people at the various highest level like oh why don't you try some new ideas around model architectures that incorporate this and then it will go run lots of experiments see which ones work and then those will get incorporated at a much more rapid rate and you know effectively you want to optimize you know your discoveries per unit of compute input. >> Very cool. >> Yeah. >> Now going back to the room as all of you will become at some point founders or start your careers you will probably collect lots of rejections. That will happen. it has happened to you too Jeff. I mean there's a story that in 2014 you with Jeff Hinton and Oral Fin wrote a paper on distillation >> which has to do with taking a big teacher model to train a much smaller and more efficient model that's a lot cheaper to compute less model parameters and it has become a trick that everyone is using right now in industry. Yeah.

在最高层面上得到人的一点提示，比如说，你不如试试围绕模型架构、结合这个东西的一些新想法，然后它就会去跑大量实验，呃看看哪些管用，然后这些就会以快得多的速度被整合进来。呃，你知道，本质上你想优化的是每单位算力投入所带来的发现量。>> 非常酷。>> 是的。>> 那我们回到在座各位——你们中的一些人以后会成为创业者，或者刚开始职业生涯，你们大概会收到很多拒信。这是会发生的。呃，你也经历过，Jeff。我是说，有个故事是，2014 年你和 Geoff Hinton、Oriol Vinyals 一起写了一篇关于蒸馏（distillation）的论文 >> 就是用一个大的教师模型去训练一个小得多、更高效的模型，计算成本低很多，模型参数也更少，而它现在已经成了业界人人都在用的技巧。是的。

蒸馏（distillation）：2014 年 Jeff 与 Geoffrey Hinton（深度学习先驱、图灵奖得主）、Oriol Vinyals 提出，用大「教师」模型的输出训练小「学生」模型，以低算力保留大部分能力，现已是全行业标配技术。

nudge /nʌdʒ/ *n.*
轻推，小提示（with a nudge from some people 在人的一点提示下）

distillation /ˌdɪstɪˈleɪʃən/ *n.*
蒸馏（ML 术语：用大模型的输出训练小模型的技术）

48:30 Jeff Dean

>> And the thing is this paper got rejected at Europe. >> Yeah. I mean Yeah. I mean I think I don't fault the program committee because you know a lot of times a paper gets three reviews and someone will look at one of the reviewers will look at it and in this case they said oh it's unlikely to have significant impact. >> Unlikely to have significant impact. But you know I think you know when we wrote the paper we actually saw this was a super important problem because we knew making cheaper highly capable models from larger scale models was something we desperately wanted to do because we wanted to serve models to more and more people in many different domains like speech or vision. but you know sometimes the reviewer maybe didn't have that experience because maybe they're not thinking about you know largecale AI services and are thinking about you know is this a fundamental advance so you know it gets rejected every so often that's fine we put it on archive people read it people

>> 问题是，这篇论文被 NeurIPS 拒了。>> 是的。我是说，是的。我觉得我不怪程序委员会，因为你知道，很多时候一篇论文会拿到三份评审，有人会看一眼——其中一位审稿人看了之后，这次他们说，哦，这不太可能产生显著影响。>> 不太可能产生显著影响。但你知道，我觉得我们写这篇论文的时候，其实是看到这是一个超级重要的问题，因为我们知道，从大规模模型里做出更便宜但能力很强的模型，是我们非常迫切想做的事，因为我们想把模型服务提供给越来越多的人，在很多不同领域，比如语音或视觉。嗯，但你知道，有时候审稿人可能没有那样的经历，因为也许他们没在想呃大规模 AI 服务这件事，而是在想呃这是不是一项根本性的突破。嗯，所以你知道，隔三差五被拒一次，没关系，我们把它放到 arXiv 上，大家读了，大家用了，都挺好。呃，你知道，我们确实在做 Flash 模型时用到了它，比如

这篇日后影响深远的论文被 NeurIPS 拒稿，理由是「不太可能有显著影响」。Jeff 的复盘很克制：审稿人没有运营大规模 AI 服务的经验，看不到「更便宜的高能力模型」的工业价值——评审视角决定了判断盲区。

fault /fɔːlt/ *v.*
指责，挑错（I don't fault them 我不怪他们）

49:31

Use it's all good and you know we do use it in making our flash models for example from our larger scale pro model that's partly why our flash models for example in Gemini are so capable relative to their size and speed. >> They're some of the best in the benchmark for their model size class. Yeah. Just impressive. And I think part of the lesson is that even if you get rejected, keep going. >> Yeah. That's the lesson I would distill from that. [laughter] >> no, I think the fun thing is that you basically join when you when you join Google as a 20 person startup back in 1999.

从我们更大规模的 Pro 模型蒸馏出来，这也是为什么我们 Gemini 里的 Flash 模型相对于它们的体量和速度来说，能力这么强的部分原因。>> 在同等模型规模级别的基准测试里，它们属于最好的一批。是的，真的很厉害。我觉得其中一部分教训是，就算被拒了，也要坚持下去。>> 是的。这就是我会从中"蒸馏"出来的教训。[笑声] >> 嗯，不过我觉得有意思的是，你基本上是在 Google 还是一家 20 人的创业公司时就加入了，那是在 1999 年。

蒸馏的现实成果：Gemini Flash 系列由更大的 Pro 模型蒸馏而来，这是它在同规模级别基准表现突出的部分原因。被拒的应对：放上 arXiv，让使用者验证价值。

distill /dɪˈstɪl/ *v.*
提炼，萃取（此处双关：从教训中「蒸馏」出结论）

12 职业选择与留给后来者的问题

50:09 主持人

Now, if you were to take the young Jeff Dean from way back then to teleransport him to now today. >> Yeah. >> In this era with your skills. >> I'm feeling so vigorous and young now. >> what would you do? Do you join a frontier lab, start a company? I don't know what would you do? The c the Jeff theme today 25-year-old Jeff theme. >> Yeah. I mean, it's always hard to say and it's a very personal choice of what it is you want to spend your time on. to me, some of the most important questions are, are you going to work on something you really care about, will you're working on that? And if you're able to make progress on it with a bunch of colleagues you like working with if you're able to make collectively solve it or make progress on it, will that make a difference in the world in some positive way, right? Like will you suddenly be able to do something and offer that service to you know partically help biochemists or something or maybe it's a broader thing. It'll help programmers or it will help all

那么，如果把当年那个年轻的 Jeff Dean 瞬移到今天。>> 是的。>> 在这个时代，带着你的技能。>> 我现在感觉浑身是劲、特别年轻。>> 嗯，你会做什么？你会加入一个前沿实验室，还是自己创业？我不知道，你会怎么做？今天这个 25 岁的 Jeff Dean。>> 是的。我是说，这always很难说，而且这是一个非常个人化的选择，取决于你想把时间花在什么上。嗯，对我来说，有些最重要的问题之一是，你会去做一件你真正在乎的事情吗？你会投入其中吗？如果你能和一群你喜欢共事的同事一起在这件事上取得进展，如果你们能共同解决它或者推进它，那这会不会在某种正面的意义上改变世界？比如说，你会不会突然能做成某件事，并把这项服务提供出去，比如切实地帮助到生物化学家，或者也许是更广泛的事情。它能帮到程序员，或者能帮到所有消费者，帮到互联网上的人，等等。嗯，你应该努力去做的，是在世界上产生

「25 岁的 Jeff 今天会做什么」——Jeff 拒绝给标准答案，转而给出判断框架：是否真正在乎这件事、能否与喜欢的同事取得进展、成功后是否对世界有正面影响。

vigorous /ˈvɪgərəs/ *adj.*
精力充沛的，生机勃勃的

frontier lab *phr.*
前沿实验室（指 OpenAI、DeepMind、Anthropic 等顶级 AI 研究机构）

51:24 Jeff Dean

Consumers. on the internet or other things. what you know what you should strive to do is to have impact in the world that is positive and to work with people you enjoy working with and to you know work hard and do your best. so in terms of say the particular trade-off you offered joining a frontier lab versus say starting a company with just one or two or three of you and your close friends. I think those are different experiences, right? In a in a large established organization, you have some structure. You have lots and lots of amazing colleagues who know lots of things you don't. you have lots of interesting problems that you can work on and h you already have a platform for impact by your work, you know, influencing lots and lots of people in the world already. and then as a very small startup, you know, you have to have something you're passionate about and there's a lot of risk in taking on, you know, working on that particular problem in a

正面的影响，和你喜欢共事的人一起工作，然后努力工作，尽你所能。嗯，所以说到你提出的那个具体权衡——加入一个前沿实验室，还是和一两个或三个你和你的好朋友一起创业。嗯，我觉得这是两种不同的体验，对吧？在一个大的、成熟的组织里，你有一定的架构。你有非常非常多了不起的同事，他们懂很多你不懂的东西。嗯，你有很多有意思的问题可以去做，而且你的工作本身已经有了一个产生影响的平台，已经能影响到世界上非常非常多的人。嗯，而作为一家很小的初创公司，你必须得有一件你充满热情的事情，而且要承担很大的风险，去用某种方式攻克那个特定的问题，让自己能成功，能把这份

大组织与小创业的诚实对比：前者有结构、有懂你不懂的东西的同事、有现成的影响力平台；后者需要真正的热情、承担更大风险，但回报感可能极强——没有普适的对错，只有个人偏好。

strive /straɪv/ *v.*
力求，奋力追求（strive to do 努力做到）

trade-off /ˈtreɪd ɔːf/ *n.*
权衡取舍（两者不可兼得时的抉择）

52:36

Way that you're going to succeed and you're going to grow a, you know, an endeavor in order to do that. But that can also be incredibly rewarding, I would imagine. So I think you know it's really up to personal taste but at the very least regardless of what path you take ask yourself if I work on this problem and the best possible outcome happens you know will the world be a lot better in some way or will the world go eh that's kind of cool but whatever. >> that's not the kind of thing you should spend your time on.

事业做大做起来。但我想，那同样可能是极其有成就感的。所以我觉得，嗯，这真的取决于个人的偏好。但至少，不管你选哪条路，都问问自己：如果我做这个问题，而且发生了最好的结果，那世界会不会在某种意义上变得好很多？还是说世界只会说，嗯，挺酷的，但也就那样。>> 呃，那就不是你该把时间花在上面的事情。

可操作的选题测试：想象这个问题的最好结果发生了，世界会「好很多」，还是只会说「挺酷的，但也就那样」？后者不值得投入时间。这与第 39 段的「选题品味」呼应。

rewarding /rɪˈwɔːrdɪŋ/ *adj.*
有成就感的，值得付出的

53:10

Now let's talk a bit about more about that second path of working with people that you really like in a small team. You've been able to be an incredible mentor and manager to many engineers and you've been able to build huge systems and what are some of the lessons for everyone here on how to get the most and how to work with smart people or find smart people? Yeah, I mean, you always want to find people who have really good skills in some area that's needed in, you know, a team you're trying to form, whether that's inside a company or starting a company. but you also want to find people that are people you delight being around, right? because you're going to spend a lot of time around people working on really hard problems and you want people who are low ego that are team players that you know have complimentary skills to your own perhaps I always find working in a small team where people know things that I don't know and where maybe I have some skills that other people don't have as much of

现在我们再多聊聊第二条路——和你真正喜欢的人在一个小团队里共事。你一直是许许多多工程师非常出色的导师和管理者，你也构建过庞大的系统。对于在座的各位，关于如何收获最多、如何与聪明的人共事或者找到聪明的人，你有什么经验？是的，我是说，你总是希望找到在某个领域有非常好的技能的人，而那正是你想组建的团队所需要的，不管是在公司内部还是创业。嗯，但你也希望找到那些和他们相处让你感到愉快的人，对吧？因为你会花大量时间和这些人一起攻克真正困难的问题，你希望这些人自我意识不强、有团队精神，并且可能拥有与你互补的技能。嗯，我一直觉得在小团队里工作在那里别人知道我不知道的东西，而我可能有一些别人没那么擅长的技能

组队标准：技能匹配团队所需之外，更要找相处愉快、低自我（low ego）、有团队精神、技能与你互补的人——因为你们将长时间共同面对困难问题。

low ego *phr.*
自我意识弱的，谦逊不居功的（招聘语境高频词）

complimentary skills *phr.*
互补技能（此处为 complementary 的常见误拼，二者发音相同）

54:22

You know is super fun because you're collectively building something or working on something that none of you could maybe do individually. ually, but in the process of working on that, you actually gain a lot of new knowledge and new skills for yourself and so do they. And you kind of want to view your engineering or research career as you have an amazing tool belt of techniques. And you always want to be adding new tools to that tool belt because you never know when you might come across a problem where you need these four specialized tools rather than these three. And adding more tools makes it more likely that the problems you encounter in the future will be solvable by you.

这真的特别有意思，因为你们是在共同构建某个东西，或者共同做一件你们任何一个人单独都做不成的事。但是在做这件事的过程中，你自己实际上会获得很多新知识和新技能，他们也一样。所以你应该这样看待自己的工程或研究生涯：你有一条装满各种技术的超棒工具腰带。而你要一直往这条工具腰带上添加新工具，因为你永远不知道什么时候会遇到一个问题，需要的是这四种专门的工具，而不是那三种。工具越多，你未来遇到的问题就越有可能被你解决。

「工具腰带」比喻：把职业生涯看作持续收集技术工具的过程。在互补团队中做单人做不成的事，本身就是最好的学习方式——工具越多，未来问题可解的概率越大。

tool belt *phr.*
工具腰带；比喻个人积累的技能与方法库

55:07

>> Now, one last thing. I'm pretty sure someone in this room or multiple people will eventually build something as consequential as you've done with map reduce, TPU, distillation, etc., etc. What problem do you hope they would be working on? Oh yeah. I mean I think there's a lot of interesting problems in the world and I'll just rattle off a few. This is not exhaustive because the world is a very big place and full of problems. You know I'm particularly excited about new approaches to hardware. You know we that thought experiment there was kind of you know a you know a indication of that or much more efficient inference hardware. You know, I think there are radically different kinds of algorithms for machine learning that might be much more data efficient than the approaches we're using today. If you think about our large scale models today, they probably see a thousand times as much data as a human does by the age of 18.

>> 最后一个问题。我很确定，这个房间里的某个人、或者好几个人，最终会做出跟你做的 MapReduce、TPU、蒸馏等等一样有影响力的东西。你希望他们去解决什么问题呢？哦，是啊。我是说，我觉得这世界上有很多有意思的问题，我就随口说几个。这并不是全部，因为世界非常大，充满了各种问题。你知道，我特别期待硬件方面的新思路。我们刚才那个思想实验其实就有点这个意思，或者说更高效的推理硬件。我觉得可能存在完全不同类型的机器学习算法，它们的数据效率可能远远高于我们今天用的这些方法。想想我们今天的大规模模型，它们看过的数据可能是一个人到 18 岁时所见数据量的一千倍。

关键数据：大模型见过的数据约是人类 18 岁前的一千倍，而人类在很多方面仍更强或相当。这指向 Jeff 期待的方向之一：数据效率高得多的全新学习算法。

rattle off /ˈrætl ɔːf/ *phr.*
一口气流利地说出（rattle off a few 随口列举几个）

exhaustive /ɪgˈzɔːstɪv/ *adj.*
详尽无遗的（not exhaustive 并非完整清单）

56:09

Yet, the human by the age of 18 is better in a lot of things and, you know, on par with those frontier models that have seen way more data. So could you come up with much more data efficient systems that can learn continuously learn from their own actions? continual learning is a really interesting thing. I think multi– aent interactions is an interesting thing. you know I think you know creating ways of having better discourse among people in the world could be interesting. Are there ways to have much more civil conversations and you know helping people meet other people are all over the world that they should know based on their interests. You know these are kind of interesting things. I think there's lots of cool things in the world and we should all go and strive to make even cooler things occur.

然而这个人到 18 岁时，在很多事情上都做得更好，而且跟那些看过多得多数据的前沿模型不相上下。那么，你能不能设计出数据效率高得多的系统，能够从自己的行动中持续学习？呃，持续学习是个非常有意思的方向。我觉得多智能体交互也是个有意思的方向。嗯，我觉得，创造一些方式让世界上的人们能有更好的对话交流，可能也很有意思。有没有办法让对话更加文明？还有帮助人们认识世界各地那些基于兴趣本该认识的人。你知道，这些都是挺有意思的事情。我觉得这世界上有很多很酷的东西，我们都应该去努力让更酷的东西出现。

56:59 主持人

>> That sounds wonderful. Thank you so much Jeff Dane. That's all we have today. >> Appreciate it. >> Thank you all.

>> 听起来太棒了。非常感谢你，Jeff Dean。今天就到这里。>> 谢谢，很荣幸。>> 谢谢大家。

Jeff 的开放问题清单：持续学习（从自身行动中不断学习）、多智能体交互、以及一个非技术方向——让互联网上的对话更文明、帮人基于兴趣结识世界各地的同好。

on par with *phr.*
与…相当，不相上下

continual learning *phr.*
持续学习（ML 术语：模型在部署后不断从新经验中学习）

discourse /ˈdɪskɔːrs/ *n.*
话语，公共讨论（better discourse 更良性的公共对话）

civil /ˈsɪvl/ *adj.*
文明礼貌的（civil conversations 文明的对话）

第二部分　核心句型 · 值得模仿的表达

1. depending on (exactly) your definition of …, it seems …

“Depending on exactly your definition of junior engineer it seems pretty spot-on”

先限定前提再下判断，是回应争议性问题的稳妥句式。适合在结论依赖于概念界定时使用，显得严谨而不武断。仿写：Depending on your definition of success, the project seems on track.

2. Imagine what you could do with something where …

“Imagine what you could do with something where the latency is, you know, 50x better”

用祈使句邀请听者想象一个量化改变后的世界，是展示技术愿景的经典开场。where 引导定语从句描述新条件。适合 pitch 和演讲中激发听众想象。

3. people don't quite realize how … it is to …

“People don't quite realize how possible it is to have agent-based systems that can run … for days or weeks”

指出普遍认知盲区的委婉句式：don't quite realize 比 don't know 语气缓和，how + 形容词 + it is to 结构突出被低估的程度。表达反共识观点时好用。

4. not (necessarily) being anchored on …, but how you would solve it from first principles

“Not necessarily being anchored on exactly how that problem is solved today, but how you would solve it from first principles”

not…but… 对比结构承载方法论：否定现状锚定，肯定第一性原理。anchored on 是形象的认知隐喻。适合论述创新思维方式的场景。

5. The clearer you are on …, the more …

“The clearer you are on what it is you want, the more the agent will have sort of guidelines”

「the 比较级, the 比较级」表示同步递进关系。注意嵌入的 what it is you want 是强调性名词从句（比 what you want 更突出）。可仿写：The earlier you test, the fewer surprises you'll get.

6. look for something where … 0% or 1% of the time, not 20%

“Look for something where the model succeeds 0% or 1% of the time not 20%”

用具体数字对比代替抽象形容词，把「选完全做不到的事」这一反直觉建议说得精确可执行。where 从句限定筛选条件，not + 数字构成简洁的排除。给建议时值得模仿的量化表达。

7. it's good to not take as a given things that most people seem to take as a given

“Sometimes it's good to not take as a given things that most people seem to take as a given”

take … as a given 意为「视为理所当然」；此句宾语后置（take as a given + things that…），并用同一短语前后呼应形成修辞张力。表达质疑常识的立场时非常地道。

8. ask yourself: if … and the best possible outcome happens, will …?

“Ask yourself if I work on this problem and the best possible outcome happens you know will the world be a lot better in some way”

自我拷问式的决策框架句：用最好情形（best possible outcome）做思想实验来检验选题上限。适合写决策类文章或演讲结尾的行动号召。

第三部分　生词总表 · 复习用

词汇	音标	词性	释义	出现
abstraction	/æbˈstrækʃən/	n.	抽象（层）（计算机科学核心概念：隐藏底层细节的接口）	41:16
amortize	/ˈæmərtaɪz/	v.	摊销，摊薄（把一次性开销分摊到多次操作上）	12:51
analogy	/əˈnælədʒi/	n.	类比（a core analogy 一个核心类比）	15:02
anchored on	/ˈæŋkərd/	phr.	被…锚定、束缚（认知科学中的锚定效应）	9:39
at the forefront	—	phr.	处于最前沿（models at the forefront 前沿模型）	26:49
at the heart of	—	phr.	处于…的核心	7:47
back of the envelope calculation	—	phr.	信封背面式估算，粗略推算（与 napkin math 同义）	6:39
bandwidth	/ˈbændwɪdθ/	n.	带宽（单位时间可传输的数据量）	11:09
batching	/ˈbætʃɪŋ/	n.	批处理（把多个样本打包一起计算以提高硬件效率）	12:51
bottlenecks	/ˈbɑːtlneks/	n.	瓶颈	9:39
brought many many chips to bear on	—	phr.	bring…to bear on：把（资源、力量）投入运用于	8:48
building block	—	phr.	构建模块，基础组件	29:04
cache footprint	—	phr.	缓存占用（程序运行时占用高速缓存的量）	20:48
cache miss	—	phr.	缓存未命中（数据不在高速缓存中、须访问更慢存储的情形）	10:25
caution	/ˈkɔːʃən/	v.	告诫，提醒注意（I would caution that… 我要提醒的是…）	26:49
checkpointing	/ˈtʃekpɔɪntɪŋ/	n.	检查点机制（定期保存状态以便故障后恢复）	40:09
civil	/ˈsɪvl/	adj.	文明礼貌的（civil conversations 文明的对话）	56:09
cobbling together	/ˈkɑːblɪŋ/	phr.	cobble together：草草拼凑，攒出来	35:02
complimentary skills	—	phr.	互补技能（此处为 complementary 的常见误拼，二者发音相同）	53:10
compound	/kəmˈpaʊnd/	v.	（误差、利息）复利式累积放大（errors that compound 不断累积的误差）	22:01

compounding	/kəmˈpaʊndɪŋ/	v.	复利式积累，不断叠加增强	25:37
computationally intensive	—	phr.	计算密集的，算力开销极大的	44:12
consequential	/ˌkɑːnsəˈkwenʃəl/	adj.	影响重大的，举足轻重的	8:48
consolidated	/kənˈsɑːlɪdeɪtɪd/	v.	整合，归并（get consolidated into 被整合为）	16:07
continual learning	—	phr.	持续学习（ML 术语：模型在部署后不断从新经验中学习）	56:09
crisp	/krɪsp/	adj.	清晰利落的（crisp design docs 简明清晰的设计文档）	31:07
crux	/krʌks/	n.	症结，最关键之处（the crux of… …的核心）	33:51
decompose	/ˌdiːkəmˈpoʊz/	v.	分解，拆解（decompose a problem into… 把问题拆成…）	17:47
decomposition	/ˌdiːˌkɑːmpəˈzɪʃən/	n.	分解，拆解（problem decomposition 问题拆解）	1:44
degrade	/dɪˈgreɪd/	v.	退化，性能下降	22:41
delightful	/dɪˈlaɪtfl/	adj.	令人愉悦的（硅谷产品话语：delightful product 体验极佳的产品）	25:37
dictate	/ˈdɪkteɪt/	v.	口述（dictate emails 用语音口述邮件）	6:39
discard	/dɪˈskɑːrd/	v.	丢弃，舍弃	23:43
discourse	/ˈdɪskɔːrs/	n.	话语，公共讨论（better discourse 更良性的公共对话）	56:09
disk seek	—	phr.	磁盘寻道（磁头移动到目标数据位置的操作，毫秒级耗时）	10:25
distill	/dɪˈstɪl/	v.	提炼，萃取（此处双关：从教训中「蒸馏」出结论）	49:31
distillation	/ˌdɪstɪˈleɪʃən/	n.	蒸馏（ML 术语：用大模型的输出训练小模型的技术）	47:15
durable	/ˈdʊrəbl/	adj.	持久的，经得起时间考验的（a durable thing 能长期站得住的事）	26:49
endeavor	/ɪnˈdevər/	n.	（费力的）事业，大工程（a six-month endeavor 一项半年的大工程）	45:22
epochs	/ˈepəks/	n.	轮次（ML 术语：完整过一遍训练数据集为一个 epoch）	13:51
error margins	—	phr.	误差余量（为容错预留的设计空间）	37:25

esoteric	/ˌesəˈterɪk/	adj.	玄奥的，只有内行才懂的	35:02
exhaustive	/ɪgˈzɔːstɪv/	adj.	详尽无遗的（not exhaustive 并非完整清单）	55:07
fabrication	/ˌfæbrɪˈkeɪʃən/	n.	（芯片）制造（chip fabrication，简称 fab）	36:10
fall off	—	n./phr.	衰减，下降幅度（the fall off in network bandwidth 带宽的衰减）	11:09
fault	/fɔːlt/	v.	指责，挑错（I don't fault them 我不怪他们）	48:30
fiddling with	—	phr.	摆弄，反复调试（fiddle with the model 调模型）	6:39
first principles	—	phr.	第一性原理（抛开现有方案，从基本事实出发推导）	9:39
fleet	/fliːt/	n.	机群，（服务器）集群（原义为舰队）	6:00
frontier lab	—	phr.	前沿实验室（指 OpenAI、DeepMind、Anthropic 等顶级 AI 研究机构）	50:09
general purpose	/ˌdʒenrəl ˈpɜːrpəs/	adj.	通用的（与 specialized 专用的相对）	3:21
go off the rail	—	phr.	（原为 go off the rails）脱轨，失控跑偏	22:01
halved	/hævd/	v.	使减半（halve the error rate 把错误率降低一半）	6:39
handing it off	—	phr.	hand off：移交，交接（把任务交给别人）	32:50
harnesses	/ˈhɑːrnɪsɪz/	n.	（智能体的）运行脚手架、驾驭框架（原义为马具，AI 行业指包裹模型的执行环境）	18:50
impediment	/ɪmˈpedɪmənt/	n.	障碍，阻碍（no real impediment to… 没有实质障碍）	46:04
infer	/ɪnˈfɜːr/	v.	推断，推测（infer what it is you meant 推断你的本意）	31:45
inference	/ˈɪnfərəns/	n.	推理（ML 术语：模型部署后的运行阶段，与训练相对）	3:21
inklings	/ˈɪŋklɪŋz/	n.	端倪，模糊的迹象（see inklings of 看到…的苗头）	4:27
interconnect	/ˌɪntərkəˈnekt/	n.	互连（芯片间的高速连接通道）	8:48
intermixed	/ˌɪntərˈmɪkst/	v.	混杂在一起（be intermixed with 与…交织混杂）	40:09
internalized	/ɪnˈtɜːrnəlaɪzd/	v.	内化，真正消化理解（把认识变成自己思维的一部分）	4:27

iterate	/ˈɪtəreɪt/	v.	迭代，反复改进（iterate on that 在此基础上不断迭代）	20:48
latency	/ˈleɪtənsi/	n.	延迟，时延（系统响应所需时间）	3:21
lays out	—	phr.	lay out：布局，排布（芯片设计中的布局规划）	42:03
lossy	/ˈlɔːsi/	adj.	有损的（lossy compression 有损压缩，与 lossless 无损相对）	15:02
low ego	—	phr.	自我意识弱的，谦逊不居功的（招聘语境高频词）	53:10
low precision	—	phr.	低精度（ML 硬件术语：用更少比特表示数值以换取速度和能效）	7:47
microbenchmark	/ˌmaɪkroʊˈbentʃmɑːrk/	n.	微基准测试（测量单个小操作性能的测试）	19:46
napkin math	—	phr.	餐巾纸计算，粗略的量级估算	6:00
neural approximation	—	phr.	神经网络近似（用神经网络拟合昂贵计算过程的替代模型）	44:12
neuromorphic	/ˌnʊroʊˈmɔːrfɪk/	adj.	神经形态的（neuromorphic computing 模仿大脑结构的计算）	39:28
niche	/nɪtʃ/	adj./n.	小众的（细分市场）；a niche model 小众专用模型	29:04
nudge	/nʌdʒ/	n.	轻推，小提示（with a nudge from some people 在人的一点提示下）	47:15
obscured	/əbˈskjʊrd/	v.	被遮蔽，被淹没（get obscured by 被…掩盖）	41:16
off the distribution	—	phr.	偏离（训练数据的）分布（ML 术语，即 out-of-distribution）	22:41
on par with	—	phr.	与…相当，不相上下	56:09
openloop	/ˈoʊpən luːp/	adj.	（open-loop）开环的（控制论术语：无反馈校正的系统）	22:01
orchestration	/ˌɔːrkɪˈstreɪʃən/	n.	编排，协调调度（原指管弦乐配器，引申为对多组件系统的统筹）	17:47
order of magnitude	—	phr.	数量级（十倍为一个数量级）	9:39
plausible	/ˈplɔːzəbl/	adj.	貌似可行的，说得通的（plausible ways 可能可行的路径）	23:43
populate	/ˈpɑːpjuleɪt/	v.	填充（populate a data structure 向数据结构填入数据）	19:46

proprietary	/prəˈpraɪəteri/	adj.	专有的，私有的（proprietary system 公司内部专有系统）	24:30
protein folding	—	phr.	蛋白质折叠（由氨基酸序列预测三维结构的生物学难题）	30:04
question assumptions	—	phr.	质疑假设（question 作动词：对…提出质疑）	38:30
racks	/ræks/	n.	机架（数据中心中安放服务器的柜架）	37:25
rattle off	/ˈrætl ɔːf/	phr.	一口气流利地说出（rattle off a few 随口列举几个）	55:07
recipe	/ˈresəpi/	n.	配方；ML 行业指训练方案的完整配置组合	46:04
recursively	/rɪˈkɜːrsɪvli/	adv.	递归地（recursively self-improve 递归自我改进）	46:04
redundant	/rɪˈdʌndənt/	adj.	冗余的（redundant paths 冗余路径；工程中冗余是容错手段）	38:30
retrieval	/rɪˈtriːvəl/	n.	检索（RAG 中的 R，即从外部知识库取回相关信息）	16:07
rewarding	/rɪˈwɔːrdɪŋ/	adj.	有成就感的，值得付出的	52:36
rhyme with	/raɪm/	phr.	与…押韵；引申为与…惊人地相似（源自「历史不重复但押韵」）	39:28
robust	/roʊˈbʌst/	adj.	稳健的，健壮的（能承受故障和异常）	41:16
scarce	/skers/	adj.	稀缺的（the scarce skill 稀缺技能）	33:51
scrape together	—	phr.	东拼西凑，勉强凑齐（scrape together enough compute 凑够算力）	45:22
screen	/skriːn/	v.	筛选（药物/材料研发术语：screen candidates 筛选候选物）	45:22
shaky	/ˈʃeɪki/	adj.	不稳的，摇摇欲坠的（gets shaky 开始不稳）	22:01
shipped	/ʃɪpt/	v.	上线，发布（科技行业用法：ship in production 推上生产环境）	2:40
specs	/speks/	n.	规格说明（specification 的缩写，软件行业常用）	31:07
spot-on	/ˌspɑːt ˈɑːn/	adj.	完全准确的，一语中的的（英式口语色彩，作表语）	0:50
squint at	/skwɪnt/	phr.	眯起眼看；引申为换个角度、模糊细节地重新审视问题	9:39
steering	/ˈstɪrɪŋ/	v.	掌舵，引导方向	33:51

stirred together into a soup	—	phr.	搅成一锅汤（比喻信息混杂交融、无法分辨来源）	16:42
strive	/straɪv/	v.	力求，奋力追求（strive to do 努力做到）	51:24
surface	/ˈsɜːrfɪs/	n.	（产品的）交互界面、承载面（行业用法：product surface）	25:37
take as a given	—	phr.	视为理所当然，当作默认前提	36:10
tool belt	—	phr.	工具腰带；比喻个人积累的技能与方法库	54:22
tooling	/ˈtuːlɪŋ/	n.	工具链，成套工具（internal tooling 内部工具）	24:30
trade-off	/ˈtreɪd ɔːf/	n.	权衡取舍（两者不可兼得时的抉择）	51:24
vigorous	/ˈvɪgərəs/	adj.	精力充沛的，生机勃勃的	50:09
weigh	/weɪ/	v.	权衡，掂量（weigh that 权衡这一点）	26:49

第四部分　理解自测 · 8 题

1. Jeff 在 2025 年 AI Ascent 上做了什么预测？一年后他如何自评？
2. TPU 的诞生源于一笔怎样的「餐巾纸计算」？
3. 为什么机器学习训练必须做 batching？Jeff 认为这是模型问题吗？
4. Jeff 建议创业者选择通用模型表现如何的问题？为什么 20% 成功率反而是坏信号？
5. 关于「一天出错 20 次的晶体管」的思想实验想说明什么？
6. 量子化学的例子中，神经近似模型带来了多大加速？这改变了什么？
7. 蒸馏论文被 NeurIPS 拒稿的理由是什么？Jeff 从中提炼的教训是？
8. 当智能体包揽写代码之后，Jeff 认为什么成为稀缺技能？如何培养？

参考答案

1. 他预测 AI 达到初级工程师水平（段 0）；一年后他认为基本准确，且低估了复杂任务能力的增速和智能体向编程之外领域的扩散（段 1）。
2. 2013 年估算：若每个 Google 用户每天用 3 分钟语音识别，服务器集群就得翻倍。为此不如造专用芯片——即 TPU，能效比当时 CPU/GPU 高 30~80 倍（段 7–9）。
3. 不是模型问题而是系统能耗问题：数据搬运的能耗约是计算本身的一千倍，打成批次才能把搬运开销摊薄为「千分之批大小」（段 14–16）。
4. 选模型成功率 0%~1% 的问题。20% 说明该能力已在模型中涌现，随着数据和规模扩大很快会变好，创业空间会被通用模型吞没（段 33）。
5. 示范如何质疑人人视为理所当然的假设：分布式系统早已用不可靠部件构建可靠系统（三副本、纠删码），晶体管层面或许也可以，即使结论未必成立，定期质疑假设本身有价值（段 41–43）。
6. 把一整晚的密度泛函理论模拟加速约 30 万倍且精度接近，使筛选 1000 万个候选分子从半年工程变成午饭工夫——快速评估器彻底改变科研循环的节奏（段 49–50）。
7. 审稿意见称其「不太可能有显著影响」。教训：审稿人可能缺乏工业视角；被拒不必气馁，放上 arXiv 让实践检验价值——Gemini Flash 正是靠蒸馏做出来的（段 52–54）。
8. 对「让智能体做什么」的选题品味。培养方式：积累多样问题经验；写下未来 12 个月的重要事项预测并定期回看校准；做疯狂的思想实验（段 39–41）。