ALEXANDER EMBIRICOS

LENNY'S PODCAST

BILINGUAL TRANSCRIPT

ORIGINAL BY

Lenny Rachitsky

@lennysan • x.com/lennysan

ANALYSIS BY

@Penny777 • x.com/penny777

Generated: 2026-01-16

Alexander Embiricos - 双语对照

Lenny's Podcast: Alexander Embiricos (OpenAI Codex) - Bilingual Transcript

(00:00:00) Lenny Rachitsky

English:

You lead work on Codex.

中文翻译:

你负责 Codex 的领导工作。

(00:00:01) Alexander Embiricos

English:

Codex is OpenAI's coding agent. We think of Codex as just the beginning of a software engineering teammate. It's a bit like this really smart intern that refuses to read Slack, doesn't check Datadog unless you ask it to.

中文翻译:

Codex 是 OpenAI 的编程智能体（coding agent）。我们认为 Codex 仅仅是软件工程“队友”的一个开端。它有点像那种极其聪明、但拒绝看 Slack 消息，除非你要求否则绝不主动检查 Datadog（监控工具）的实习生。

(00:00:12) Lenny Rachitsky

English:

I remember Karpathy tweeted the gnarliest bugs that he runs into that he just spends hours trying to figure out nothing else has solved, he gives it to Codex, lets it run for an hour and it solves it.

中文翻译:

我记得 Karpathy 发过推文说，他遇到过一些最棘手的 bug，花了好几个小时尝试解决，用尽其他办法都无济于事，最后他把问题丢给 Codex，让它运行一个小时，结果它真的解决了。

(00:00:21) Alexander Embiricos

English:

Starting to see glimpses of the future where we're actually starting to have Codex be on call for its own training. Codex writes a lot of the code that helps manage its training run, the key infrastructure. So we have a Codex code review that's catching a lot of mistakes. It's actually caught some pretty interesting configuration mistakes. One of the most mind-blowing examples of acceleration, the Sora Android app, like a fully new app, we built it in 18 days and then 10 days later, so 28 days total, we went to the public.

中文翻译:

我们已经开始窥见未来的一角：Codex 甚至开始为它自己的训练过程“值班”（on call）。Codex 编写了大量用于管理其训练运行的代码，也就是那些关键的基础设施。我们还让 Codex 进行代码审查，它捕捉到了很多错误，甚至发现了一些非常有趣的配置失误。关于效率提升，最令人震撼的例子之一是 Sora 的安卓应用——这是一个全新的 App，我们仅用 18 天就完成了开发，10 天后（总计 28 天）就正式向公众发布了。

(00:00:45) Lenny Rachitsky

English:

How do you think you win in this space?

中文翻译:

你认为在这个领域怎样才能胜出？

(00:00:47) Alexander Embiricos

English:

One of our major goals with Codex is to get to proactivity. If we're going to build a super system, has to be able to do things. One of the learnings over the past year is that for models to do stuff, they're much more effective when they can use a computer. It turns out the best way for models to use computers is simply to write code. And so we're kind of getting to this idea where if you want to build any agent, maybe you should be building a coding agent.

中文翻译:

Codex 的主要目标之一是实现“主动性”。如果我们想构建一个超级系统，它必须具备执行能力。过去一年的经验告诉我们，模型要执行任务，如果能直接操作电脑，效率会高得多。而事实证明，模型操作电脑的最佳方式就是编写代码。所以我们逐渐形成了一个观点：如果你想构建任何类型的智能体（agent），也许你首先应该构建一个编程智能体。

(00:01:04) Lenny Rachitsky

English:

When you think about progress on Codex, I imagine you have a bunch of evals and there's all these public benchmarks.

中文翻译:

当你思考 Codex 的进展时，我猜你们一定有一堆评估测试（evals）和各种公开的基准测试。

(00:01:10) Alexander Embiricos

English:

A few of us are constantly on Reddit. There's praise up there and there's a lot of complaints. What we can do is as a product team just try to always think about how are we building a tool so that it feels like we're maximally accelerating people rather than building a tool that makes it more unclear what you should do as the human?

中文翻译:

我们团队有几个人经常逛 Reddit。那里既有赞扬也有很多抱怨。作为产品团队，我们能做的就是始终思考：我们如何构建一个工具，让人们感觉到效率得到了最大化的提升，而不是构建一个让作为人类的你更不清楚该做什么的工具？

(00:01:24) Lenny Rachitsky

English:

Being at OpenAI, I can't not ask about how far you think we are from AGI.

中文翻译:

既然你在 OpenAI，我不能不问：你觉得我们离 AGI（通用人工智能）还有多远？

(00:01:28) Alexander Embiricos

English:

The current underappreciated limiting factor is literally human typing speed or human multitasking speed.

中文翻译:

目前一个被低估的限制因素，其实就是人类的打字速度或者人类处理多任务的速度。

(00:01:35) Lenny Rachitsky

English:

Today, my guest is Alexander Embiricos, product lead for Codex, OpenAI's incredibly popular and powerful coding agent. In the words of Nick Turley, head of ChatGPT and former podcast guest, "Alex is one of my all time favorite humans I've ever worked with, and bringing him and his company into OpenAI ended up being one of the best decisions we've ever made." Similarly, Kevin Weil, OpenAI's CPO, said, "Alex is simply the best."

中文翻译:

今天的嘉宾是 Alexander Embiricos，他是 OpenAI 备受欢迎且功能强大的编程智能体 Codex 的产品负责人。用 ChatGPT 负责人、也是本播客往期嘉宾 Nick Turley 的话来说：“Alex 是我合作过的最喜欢的人之一，把他和他的公司带进 OpenAI 是我们做过的最正确的决定之一。” 同样，OpenAI 的首席产品官 Kevin Weil 也评价道：“Alex 简直是最棒的。”

(00:01:59) Lenny Rachitsky

English:

In our conversation, we chat about what it's truly like to build product at OpenAI, how Codex allowed the Sora team to ship the Sora app, which became the number one app in the app store in under one month. Also, the 20x growth Codex is seeing right now and what they did to make it so good at coding, why his team is now focused on making it easier to review code, not just write code, his AGI timelines, his thoughts on when AI agents will actually be really useful, and so much more. A huge thank you to Ed Bayes, Nick Turley, and Dennis Yang for suggesting topics for this conversation. If you enjoy this podcast, don't forget to subscribe and follow it in your favorite podcasting app or YouTube. And if you become an annual subscriber of my newsletter, you get a year free of 19 incredible products... Head on over to lennysnewsletter.com and click Product Pass.

中文翻译:

在我们的对话中，我们聊到了在 OpenAI 做产品究竟是什么样的体验；Codex 是如何帮助 Sora 团队在不到一个月内发布 Sora App 并使其登顶应用商店榜首的。此外，我们还讨论了 Codex 目前正经历的 20 倍增长，以及他们为了让它如此擅长编程所做的努力；为什么他的团队现在专注于让代码审查（不仅仅是编写）变得更容易；他的 AGI 时间线；他对 AI 智能体何时能真正变得实用的看法，等等。非常感谢 Ed Bayes、Nick Turley 和 Dennis Yang 为本次对话提供的话题建议。如果你喜欢这个播客，别忘了在应用或 YouTube 上订阅。如果你成为我时事通讯的年度订阅者，你将免费获得 19 款出色产品的一年使用权……请访问 lennysnewsletter.com 并点击 Product Pass。

(00:02:55) Lenny Rachitsky

English:

With that, I bring you Alexander Embiricos, after a short word from our sponsors.

中文翻译:

在听完赞助商的简短介绍后，让我们正式欢迎 Alexander Embiricos。

(00:03:00) Lenny Rachitsky (Sponsor: WorkOS)

English:

Here's a puzzle for you. What do OpenAI, Cursor, Perplexity, Vercel, Plaid, and hundreds of other winning companies have in common? The answer is they're all powered by today's sponsor, WorkOS. If you're building software for enterprises, you've probably felt the pain of integrating single sign-on, SCIM, RBAC, audit logs, and other features required by big customers. WorkOS turns those deal blockers into drop-in APIs with a modern developer platform built specifically for B2B SaaS. Whether you're a seed stage startup trying to land your first enterprise customer or a unicorn expanding globally, WorkOS is the fastest path to becoming enterprise ready and unlocking growth. They're essentially Stripe for enterprise features. Visit workos.com to get started or just hit up their Slack support where they have real engineers in there who answer your questions super fast. WorkOS allows you to build like the best, with delightful APIs, comprehensive docs and a smooth developer experience. Go to workos.com to make your app enterprise ready today.

中文翻译:

给你出一个谜题。OpenAI、Cursor、Perplexity、Vercel、Plaid 以及数百家其他成功的公司有什么共同点？答案是：它们都由今天的赞助商 WorkOS 提供支持。如果你正在为企业构建软件，你可能感受过集成单点登录（SSO）、SCIM、RBAC（基于角色的访问控制）、审计日志以及大客户要求的其他功能的痛苦。WorkOS 将这些交易阻碍变成了即插即用的 API，并提供专门为 B2B SaaS 构建的现代开发平台。无论你是试图签下第一个企业客户的种子期初创公司，还是正在全球扩张的独角兽，WorkOS 都是实现企业级就绪并释放增长的最快路径。

它们本质上是企业级功能的“Stripe”。访问 workos.com 开始使用，或者直接联系他们的 Slack 支持，那里有真正的工程师会飞速回答你的问题。WorkOS 让你能像顶尖公司一样构建产品，拥有令人愉悦的 API、详尽的文档和流畅的开发体验。今天就去 workos.com 让你的应用具备企业级能力吧。

(00:04:01) Lenny Rachitsky (Sponsor: Fin)

English:

This episode is brought to you by Fin, the number one AI agent for customer service. If your customer support tickets are piling up, then you need Fin. Fin is the highest performing AI agent on the market with a 65% average resolution rate. Fin resolves even the most complex customer queries. No other AI agent performs better. In head to head bake offs with competitors, Fin wins every time. Yes, switching to a new tool can be scary, but Fin works on any help desk with no migration needed, which means you don't have to overhaul your current system or deal with delays in service for your customers. And Fin is trusted by over 6,000 customer service leaders and top companies like Anthropic, Shutterstock, Synthesia, Clay, Vanta, Lovable, Monday.com and more. And because Fin is powered by the Fin AI Engine, which is a continuously improving system that allows you to analyze, train, test, and deploy with ease, Fin can continuously improve your results too. So if you're ready to transform your customer service and scale your support, give Fin a try for only 99 cents per resolution. Plus Fin comes with a 90-day money back guarantee. Find out how Fin can work for your team at fin.ai/Lenny. That's fin.ai/lenny.

中文翻译:

本集节目由 Fin 为您带来，它是排名第一的客服 AI 智能体。如果你的客服工单堆积如山，那么你需要 Fin。Fin 是市场上性能最强的 AI 智能体，平均解决率达 65%。Fin 甚至能解决最复杂的客户咨询。没有其他 AI 智能体表现得更好。在与竞争对手的正面交锋中，Fin 每次都能胜出。是的，更换新工具可能令人畏惧，但 Fin 可以在任何服务台（help desk）上运行，无需迁移，这意味着你不需要彻底改造现有系统，也不会让客户遭遇服务延迟。Fin 深受 6,000 多名客服主管以及 Anthropic、Shutterstock、Synthesia、Clay、Vanta、Lovable、Monday.com 等顶尖公司的信赖。由于 Fin 由 Fin AI 引擎驱动（这是一个持续改进的系统，让你能轻松分析、训练、测试和部署），Fin 也能持续提升你的业务成果。如果你准备好转型客服并扩展支持规模，不妨试试 Fin，每次解决仅需 99 美分。此外，Fin 还提供 90 天退款保证。访问 fin.ai/Lenny 了解 Fin 如何为你的团队服务。

(00:05:14) Lenny Rachitsky

English:

Alexander, thank you so much for being here and welcome to the podcast.

中文翻译:

Alexander，非常感谢你能来，欢迎来到本播客。

(00:05:18) Alexander Embiricos

English:

Thank you so much. I've been following for ages and I'm excited to be here.

中文翻译:

非常感谢。我关注你很久了，很高兴能来到这里。

(00:05:21) Lenny Rachitsky

English:

I'm even more excited. I really appreciate that. I want to start with your time at OpenAI. So you joined OpenAI about a year ago. Before that, you had your own startup for about five years. Before that, you were a product manager at Dropbox. I imagine OpenAI is very different from every other place you've worked. Let me just ask you this, what is most different about how OpenAI operates and what's something that you've learned there that you think you're going to take with you wherever you go, assuming you ever leave?

中文翻译:

我更兴奋，非常感谢。我想从你在 OpenAI 的时光聊起。你大约一年前加入 OpenAI，在那之前你经营了五年的初创公司，再之前你是 Dropbox 的产品经理。我猜 OpenAI 与你工作过的其他地方都大不相同。我想问，OpenAI 的运作方式最特别的地方是什么？你在那里学到了什么，是你认为无论以后去哪（假设你会离开的话）都会带走的？

(00:05:49) Alexander Embiricos

English:

By far, I would say the speed and ambition of working at OpenAI are just dramatically more than what I can imagine. And I guess it's kind of an embarrassing thing to say because everyone who's a startup founder thinks like, "Oh yeah, my startup moves super fast and the talent bar is super high and we're super ambitious." But I have to say, working in OpenAI just made me reimagine what that even means.

中文翻译:

到目前为止，我会说在 OpenAI 工作的速度和雄心壮志远超我的想象。说这话可能有点尴尬，因为每个初创公司创始人都会觉得：“哦，我的公司跑得飞快，人才门槛极高，我们野心勃勃。”但我必须说，在 OpenAI 工作让我重新定义了这些词的含义。

(00:06:11) Lenny Rachitsky

English:

We hear this a lot about feels like every AI company is just like, "Oh my God, I can't believe how fast they're moving." Is there an example of just like, "Wow, that wouldn't have happened this quickly anywhere else"?

中文翻译:

我们经常听到这种说法，感觉每家 AI 公司都是“天哪，不敢相信他们动作这么快”。有没有什么例子让你觉得“哇，这在别处绝对不可能这么快发生”？

(00:06:20) Alexander Embiricos

English:

The most obvious thing that comes to mind is just the explosive growth of Codex itself. I think it's a while since we bumped our external number, but it's like the 10x-ing of Codex's scale was just super fast in a matter of months and it's well more since then. And once you've lived through that, or at least speaking

for myself, having lived through that now, I feel like anytime I'm going to spend my time on building tech product, there's that speed and scale that I now need to meet.

(00:06:52):

If I think of what I was doing in my startup, it moved way slower and there's always this balance with startups of how much do you commit to an idea that you have versus find out that it's not working and then pivot. But I think one thing I've realized at OpenAI is the amount of impact that we can have and, in fact, need to have to do a good job is so high that I have to be way more ruthless with how I spend my time now.

中文翻译:

我脑海中浮现的最明显的例子就是 Codex 本身的爆发式增长。虽然我们有一段时间没更新对外公布的数据了，但 Codex 规模实现 10 倍增长只用了短短几个月，而那之后增长得更多。一旦你经历过这种速度——至少对我而言，现在经历过之后——我觉得以后无论什么时候开发科技产品，我都必须达到这种速度和规模。

(00:06:52):

回想我在初创公司的时候，进展要慢得多。初创公司总是在“对一个想法投入多少”和“发现行不通后转型”之间寻找平衡。但在 OpenAI 我意识到，我们能产生的影响力（事实上也是为了做好工作必须产生的影响力）是如此巨大，以至于我现在必须对自己如何分配时间变得更加“冷酷无情”。

(00:07:15) Lenny Rachitsky

English:

Before we get to Codex, is there a way that they've structured the org or, I don't know, the way that OpenAI operates that allows the team to move this quickly? Because everyone wants to move super fast. I imagine there's a structural approach to allowing this to happen.

中文翻译:

在聊 Codex 之前，OpenAI 的组织架构或者运作方式有什么特别之处，能让团队跑得这么快吗？因为每个人都想快，我猜一定有某种结构性的方法在支撑。

(00:07:29) Alexander Embiricos

English:

I mean, so one thing is just the technology that we're building with has just transformed so many things from both how we build, but also what kinds of things we can enable for users. And we spend most of our time talking about the sort of improvements within the foundation models, but I believe that even if we had no more progress today with models, which is absolutely not the case, but even if we had no more progress, we are way behind on product. There's so much more product to build. So I think just the moment is ripe, if that makes sense.

(00:08:01):

But I think there's a lot of counterintuitive things that surprised me when I arrived as far as how things are structured. One example that comes to mind is when I was working on my startup and before that, when I was at Dropbox, it was very important, especially as a PM to always rally the ship and it was like make sure you're pointed in the right direction and then you can accelerate in that direction. But here, I think because we don't exactly know what capabilities will even come up soon and we don't know what's going to work technically, and then we also don't know what's going to land even if it works technically,

it's much more important for us to be very humble and learn a lot more empirically and just try things quickly. And the org is set up in that way to be incredibly bottoms up.

(00:08:45):

This is, again, one of those things that, as you were saying, everyone wants to move fast. I think everyone likes to say that they're bottoms up, or at least a lot of people do, but OpenAI is truly, truly bottoms up. And that's been a learning experience for me that now it'll be interesting if I ever work at... I don't think it'll even make sense to work at a non-AI company in the future. I don't even know what that means. But if I were to imagine it or go back in time, I think I would run things totally new.

中文翻译:

首先，我们所使用的技术本身就改变了许多事情，包括我们构建产品的方式，以及我们能为用户提供的功能。我们大部分时间都在讨论基础模型（foundation models）的改进，但我相信，即使模型从今天起不再进步（当然事实并非如此），我们在产品层面也依然远远落后。还有太多的产品值得去构建。所以我觉得现在是时机成熟了。

(00:08:01):

但在组织结构方面，我刚到这里时确实被很多反直觉的事情惊到了。比如在我做初创公司或在 Dropbox 时，作为 PM，最重要的事情通常是“稳住航向”，确保大家朝着正确的方向前进，然后在这个方向上加速。但在 OpenAI，因为我们并不确切知道很快会出现什么新能力，不知道技术上什么行得通，甚至不知道技术行得通后用户是否买账，所以对我们来说，保持谦逊、更多地通过经验学习并快速尝试变得重要得多。整个组织架构被设定为极其彻底的“自下而上”（bottoms up）。

(00:08:45):

这又是那种大家都在说的事情——每个人都想快，每个人都爱说自己是“自下而上”的，但 OpenAI 是真的、彻底的自下而上。这对我来说是一次深刻的学习经历。如果以后我去……其实我觉得未来去一家非 AI 公司工作已经没意义了，我甚至不知道那意味着什么。但如果让我回过头去重新运行一些事情，我想我会采取完全不同的方式。

(00:09:10) Lenny Rachitsky

English:

What I'm hearing is this ready, fire, aim is the approach more than ready, aim, fire. And there's something, and as you process that, because that may not come across well, but I actually have heard this a lot at AI companies is because you don't know, and Nick Turley shared I think the same sentiment, because you don't know how people will use it it doesn't make sense to spend a lot of time making it perfect. It's better to just get it out there in a primordial way, see how people use it, and then go big on that use case.

中文翻译:

我听起来这更像是“准备、开火、瞄准”，而不是传统的“准备、瞄准、开火”。当你消化这一点时（因为这听起来可能不太对劲），我确实在很多 AI 公司听到过类似的说法。因为你不知道用户会怎么用它（Nick Turley 也表达过类似的观点），所以花大量时间把它做得完美是没有意义的。更好的做法是以一种原始的状态把它推出去，观察人们怎么用，然后在那个用例上大举投入。

(00:09:39) Alexander Embiricos

English:

Yeah. Okay, to use this analogy a little bit, I feel like there is an aim component, but the aim component is much fuzzier. It's kind of like, roughly what do we think can happen? Someone I've learned a ton from working here is a research lead, and he likes to say that at OpenAI, we can have really good conversations about something that's a year plus from now, and there's a lot of ambiguity in what will happen, but that's a right sort of timeline. And then we can have really good conversations about what's happening in low months or weeks. But there's this awkward middle ground, which was as you start approaching a year, but you're not at a year where it's very difficult to reason about, right?

(00:10:18):

And so as far as aiming, I think we want to know, "Okay, what are some of the futures that we're trying to build towards?" And a lot of the problems we're dealing with in AI, such as alignment are problems you need to be thinking out really far out into the future. So we're kind of aiming fuzzily there, but when it comes down to the more tactically like, "Oh yeah, what product will we build and therefore how will people use that product?" That's the place where we're much more like, "Let's find out empirically."

中文翻译:

是的。沿用这个比喻，我觉得确实有“瞄准”的成分，但这个目标要模糊得多。大概就是：我们认为大致会发生什么？我在这里从一位研究主管身上学到了很多，他喜欢说，在 OpenAI，我们可以很好地讨论一年多以后的事情，虽然有很多不确定性，但那是合适的时间跨度。我们也可以很好地讨论未来几周或几个月内发生的事。但有一个尴尬的中间地带，就是当你接近一年但还没到一年的时候，那非常难以推断。

(00:10:18):

所以在“瞄准”方面，我们想知道：“我们要努力构建的未来图景是什么？”AI 领域的很多问题，比如对齐（alignment），都需要你思考得非常长远。所以我们在那里进行模糊的瞄准。但当涉及到具体的战术问题，比如“我们要构建什么产品，人们会怎么用它”，在这些地方我们更倾向于“通过实践去发现”。

(00:10:41) Lenny Rachitsky

English:

That's a good way of putting it. Something else that when people hear this, people sometimes hear companies like yours saying, "Okay, we're going to be bottoms up. We're going to try a bunch of stuff. We're not going to have exactly a plan of where it's going in the next few months." The key is you all hire the best people in the world. And so that feels like a really key ingredient in order to be this successful at bottoms up work.

中文翻译:

这个说法很好。还有一点，当人们听到像你们这样的公司说“我们要自下而上，我们要尝试一堆东西，我们对未来几个月没有确切计划”时，关键在于你们雇佣的是世界上最优秀的人才。这似乎是这种“自下而上”模式能取得成功的关键要素。

(00:11:02) Alexander Embiricos

English:

It just super resonates with me. I was just, again, surprised or even shocked when I arrived at the level of individual drive and autonomy that everyone here has. So I think the way that OpenAI runs, you can't read this or listen to a podcast and be like, "I'm just going to deploy this to my company." Maybe this is a harsh thing to say, but I think very few companies have the talent caliber to be able to do that. So it might need to be adjusted if you were going to implement this.

中文翻译：

这让我非常有共鸣。我刚到这里时，再次被每个人的个人驱动力和自主性所震惊。所以我觉得 OpenAI 的运作方式，你不能只是读读文章或听听播客就说“我也要在我的公司推行这套”。这话可能有点重，但我认为极少有公司拥有能支撑这种模式的人才水准。所以如果你想借鉴，可能需要进行调整。

(00:11:34) Lenny Rachitsky

English:

Okay. So let's talk Codex. You lead work on Codex. How's Codex going? What numbers can you share? Is there anything you can share there? Also, just not everyone knows exactly what Codex is, explain what Codex is.

中文翻译：

好，那我们聊聊 Codex。你负责 Codex 的工作。Codex 进展如何？有什么数据可以分享吗？另外，并不是所有人都确切知道 Codex 是什么，请解释一下。

(00:11:45) Alexander Embiricos

English:

Totally, yeah. So I had the very lucky job of living in the future and leading products on Codex. And Codex is OpenAI's coding agent. So super concretely, that means it's an IDE extension, a VS code extension that you can install or a terminal tool that you can install. And when you do so, you can then basically pair with Codex to answer questions about code, write code, run tests, execute code, and do a bunch of the work in that thick middle section of the software development lifecycle, which is all about writing code that you're going to get into production.

(00:12:21):

More broadly, we think of Codex as what it currently is just the beginning of a software engineering teammate. So when we use a big word like teammate, some of the things we're imagining are that it's not only able to write code, but actually it participates early on in the ideation and planning phases of writing software and then further downstream in terms of validation, deploying and maintaining code.

(00:12:46):

To make that a little more fun, one thing I like to imagine is if you think of what Codex is today, it's a bit like this really smart intern that refuses to read Slack and doesn't check Datadog or Century unless you ask it to. And so no matter how smart it is, how much are you going to trust it to write code without you also working with it? So that's how people use it mostly today is they pair with it. But we want to get to the point where it can work just like a new intern that you hire, you don't only ask them to write code, but you ask them to participate across the cycle. So you know that even if they don't get something right the first try, they're eventually going to be able to iterate their rate there.

中文翻译：

没问题。我很幸运能“生活在未来”并领导 Codex 的产品工作。Codex 是 OpenAI 的编程智能体。具体来说，它是一个你可以安装的 IDE 扩展（比如 VS Code 插件），或者是一个终端工具。安装后，你就可以与 Codex “结对”，让它回答代码问题、写代码、运行测试、执行代码，并完成软件开发生命周期中那个繁重的中间环节——也就是编写要投入生产环境的代码。

(00:12:21):

更广泛地说，我们认为现在的 Codex 只是软件工程“队友”的一个雏形。当我们使用“队友”这个大词时，我们想象的是它不仅能写代码，还能在软件开发的早期构思和计划阶段参与进来，并在下游的验证、部署和维护代码中发挥作用。

(00:12:46):

为了让这个比喻更有趣一点：如果你看今天的 Codex，它有点像那个极其聪明但不看 Slack、除非你叫它否则不看 Datadog 或 Sentry（错误监控工具）的实习生。所以无论它多聪明，如果不跟它一起工作，你能在多大程度上信任它写的代码？这就是为什么今天人们主要是和它“结对”使用。但我们希望达到这样一个阶段：它就像你雇佣的新实习生一样，你不仅要求他们写代码，还要求他们参与整个流程。你知道即使他们第一次没做对，最终也能通过迭代达到目标。

(00:13:21) Lenny Rachitsky

English:

I thought the point about not reading Slack and Datadog was it's just not distracted, it's just constantly focused and is always in flow. But I get what you're saying there is it doesn't have all the context on everything that's going on.

中文翻译:

我刚才还以为你说它不看 Slack 和 Datadog 是为了不分心，始终保持专注和心流状态呢。但我明白你的意思了，它是缺乏对正在发生的事情的全局背景信息（context）。

(00:13:31) Alexander Embiricos

English:

Yeah. And that's not only true when it's performing a task, but again, if you think of the best team and teammates, you don't tell them what to do. Maybe when you first hire them, you have a couple meetings and you're like, "Hey," you learn, "Okay, these prompts work for this teammate, these prompts don't. This is how to communicate with this person." Then eventually you give them some starter tasks, you delegate a few tasks. But then eventually you just say like, "Hey, great. Okay, you're working with this set of people in this area of the code base. Feel free to work with other people on other parts of the code base too, even. And yeah, you tell me what you think makes sense to be done." And so we think of this as proactivity and one of our major goals with Codex is to get to proactivity.

(00:14:09):

I think this is critically important to achieve the mission of OpenAI, which is to deliver the benefits of AGI to all humanity. I like to joke today that AI products, and it's a half joke, they're actually really hard to use because you have to be very thoughtful about when it could help you. And if you're not prompting a model to help you, it's probably not helping you at that time. And if you think of how many times the average user is prompting AI today, it's probably tens of times. But if you think of how many times people could actually get benefit from a really intelligent entity, it's thousands of times per day. And so a large part of our goal with Codex is to figure out what is the shape of an actual teammate agent that is helpful by default.

中文翻译:

没错。这不仅体现在执行任务时。如果你想想最好的团队和队友，你是不需要告诉他们该做什么的。也许刚雇佣时，你会开几次会，了解“OK，这些提示词对这个队友有效，那些无效。这是和这个人沟通的方式。”然后你给他们一些入门任务，委派一些工作。但最终你会说：“嘿，太棒了。你现在和这组人在这个代码库区域工

作。甚至可以随时和其他人在其他部分合作。你来告诉我你认为该做什么。”我们称之为“主动性”（proactivity），这是 Codex 的主要目标之一。

(00:14:09):

我认为这对于实现 OpenAI 的使命——将 AGI 的益处带给全人类——至关重要。我常开玩笑说（半开玩笑），现在的 AI 产品其实挺难用的，因为你必须非常仔细地思考它什么时候能帮你。如果你不提示模型去帮你，它那时可能就没在帮。普通用户今天可能每天提示 AI 几十次，但如果有一个真正智能的实体，人们每天其实可以从中获益数千次。所以 Codex 的很大一部分目标是弄清楚：一个“默认就能提供帮助”的队友智能体究竟应该是什么样子的。

(00:14:54) Lenny Rachitsky

English:

When you think about Cursor and even Cloud Code, it's like a IDE that helps you code and auto completes code and maybe does some agentic work. What I'm hearing here is the vision is different, which is it's a teammate. It's like a remote teammate, a building code for you that you talk to and ask to do things. And that also does IDE, auto complete and things like that. Is that a kind of a differentiator in the way you think about Codex?

中文翻译:

当人们想到 Cursor 甚至 Cloud Code 时，会觉得它们是帮助你编码、自动补全代码、可能还做点智能体工作的 IDE。但我听你说的愿景不同：它是一个队友。就像一个远程队友，在为你编写代码，你可以和它交谈、要求它做事。它同时也具备 IDE 补全等功能。这是你思考 Codex 的一种差异化方式吗？

(00:15:18) Alexander Embiricos

English:

It's basically this idea that if you're a developer and you're trying to get something done, we want you to just feel like you have superpowers and you're able to move much, much faster. But we don't think that in order for you to reap those benefits, you need to be sitting there constantly thinking about, "How can I invoke AI at this point to do this thing?" We want you to be able to plug it in to the way that you work and have it just start to do stuff without you having to think about it.

中文翻译:

基本上就是这个想法：如果你是一名开发者，想要完成某件事，我们希望你感觉自己拥有了超能力，能跑得飞快。但我们不认为，为了获得这些好处，你必须坐在那儿不停地想：“我现在该怎么调用 AI 来做这件事？”我们希望你能把它接入你的工作流程，让它在你不需要思考的情况下就开始自动做事。

(00:15:44) Lenny Rachitsky

English:

Okay. I have a lot of questions along those lines, but just how's it going? Is there any stats, any numbers you can share about how Codex is doing?

中文翻译:

明白。关于这些我有很多问题，但先问问近况：Codex 表现如何？有什么统计数据或数字可以分享吗？

(00:15:49) Alexander Embiricos

English:

Yeah, Codex has been growing absolutely explosively since the launch of GPT-5 back in August. There's definitely some interesting product insights to talk about as to how we unlock that growth, if you're interested. But again, the last stat we shared there was we were well over 10x since August. In fact, it's been 20x since then. Also, the Codex models are serving many trillions of tokens a week now, and it's basically our most served coding model. One of the really cool things that we've seen is that the way that we decided to set up the Codex team was to build a really tightly integrated product and research team that are iterating on the model and the harness together. And it turns out that lets you just do a lot more and try many more experiments as to how these things will work together.

(00:16:35):

And so we were just training these models for use in our first party harness that we were very opinionated about. And then what we've started to see more recently actually is that other major API coding customers are now starting to adopt these models as well. And so we've reached a point where actually the Codex model is the most served coding model in the API as well.

中文翻译:

是的，自 8 月份 GPT-5 发布以来，Codex 经历了绝对爆发式的增长。如果你感兴趣的话，关于我们如何开启这种增长，确实有一些有趣的产品洞察可以聊。我们上次分享的数据是自 8 月以来增长了 10 倍以上，实际上现在已经达到了 20 倍。此外，Codex 模型现在每周处理数万亿个 token，它基本上是我们被调用次数最多的编程模型。我们发现一件很酷的事：我们决定组建 Codex 团队的方式是建立一个高度整合的产品和研究团队，共同迭代模型和“外壳”（harness，指运行环境）。事实证明，这让你能做更多事情，尝试更多关于两者如何协同工作的实验。

(00:16:35):

最初我们只是为了在自家的、带有强烈设计偏好的环境中使用而训练这些模型。但最近我们发现，其他主要的 API 编程客户也开始采用这些模型。所以我们现在已经达到了这样一个阶段：Codex 模型也是 API 中被调用最多的编程模型。

(00:16:55) Lenny Rachitsky

English:

You hinted at this, what unlocked this growth, I'm extremely interested in hearing that. It felt like before, I don't know, maybe this was before you joined the team, it just felt like Cloud Code was killing it. Just everyone was sitting on top of Cloud Code. It was by far the best way to code. And then all of a sudden Codex comes around. I remember Karpathy tweeted that he just has never seen a model like this. I think the tweet was the gnarliest bugs that he runs into that he just spends hours trying to figure out nothing else has solved, he gives it to Codex, lets it run for an hour and it solves it. What'd you guys do?

中文翻译:

你刚才暗示了这一点，我非常想听听是什么开启了这种增长。感觉以前（也许是在你加入团队之前），Cloud Code 表现得非常出色，每个人都在用它，它是公认最好的编程方式。然后突然间 Codex 出现了。我记得 Karpathy 发推说他从未见过这样的模型。推文大意是：他遇到的最棘手的 bug，花几小时都解决不了，丢给 Codex 跑一小时就解决了。你们到底做了什么？

(00:17:30) Alexander Embiricos

English:

We have this strong sort of mission here at OpenAI basically to build AGI. And so we think a lot about how can we shape the product so that it can scale. Earlier I was mentioning like, "Hey, if you're an engineer, you should be getting help from AI thousands of times per day," and so we thought a lot about the primitives for that when we launched our first version of Codex, which was Codex Cloud. And that was basically a product that had its own computer, lived in the cloud, you could delegate to it. And the coolest part about that is you could run many, many tasks in parallel. But some of the challenges that we saw are that it's a little bit harder to set that up, both in terms of environment configuration, like giving the model the tools it needs to validate its changes and to learn how to prompt in that way.

(00:18:20):

My analogy for this is, going back to this teammate analogy, it's like if you hired a teammate, but you're never allowed to get on a call with them and you can only go back and forth asynchronously over time. That works for some teammates and eventually that's actually how you want to spend most of your time. So that's still the future, but it's hard to initially adopt. And so we still have that vision of like, that's what we're trying to get you to, a teammate that you delegate to and then is proactive, and we're seeing that growing. But the key unlock is actually first you need to land with users in a way that's much more intuitive and trivial to get value from.

(00:18:54):

So the way that most people discover, the vast majority of users discover Codex today is either they download an IDE extension or they run it in their CLI and the agent works there with you on your computer interactively. And it works within a sandbox, which is actually a really cool piece of tech to help that be safe and secure, but it has access to all those dependencies. So if the agent needs to do something, it needs to run a command, it can do so within the sandbox. We don't have to set up any environment. And if it's a command that doesn't work in the sandbox, it can just ask you. And so you can get into this really strong feedback loop using the model. And then over time, our team's job is to help turn that feedback loop into you as a byproduct of using the product, configuring it so that you can then be delegating to it down the line.

(00:19:38):

And again, analogy, keep coming back to it, but if you hire a teammate and you ask them to do work, but you just give them a fresh computer from the store, it's going to be hard for them to do their job. But if as you work with them side by side, you could be like, "Oh, you don't have a password for this service we use, here's the password for this service. Yeah, don't worry, feel free to run this command," then it's much easier for them to then go off and do work for hours without you.

中文翻译:

在 OpenAI，我们有一个坚定的使命，就是构建 AGI。所以我们经常思考如何塑造产品使其能够规模化。我之前提到过，“如果你是工程师，你应该每天从 AI 那里获得数千次帮助”，所以当我们发布第一个版本的 Codex（即 Codex Cloud）时，我们考虑了很多实现这一点的基础原语。那基本上是一个拥有自己电脑、运行在云端的产品，你可以向它委派任务。最酷的地方在于你可以并行运行非常多的任务。但我们发现了一些挑战：设置起来有点难，包括环境配置，比如给模型提供验证更改所需的工具，以及学习如何以那种方式进行提示。

(00:18:20):

我的比喻是（回到队友的比喻）：这就像你雇了一个队友，但你永远不能和他们通话，只能通过异步消息来回沟通。这对某些队友有效，而且最终你也确实希望大部分时间这样工作。所以这仍然是未来，但初期很难上手。我们依然持有那个愿景：让你拥有一个可以委派任务且具有主动性的队友，我们也看到这部分在增长。但关键的突破在于：你首先需要以一种更直观、更容易获得价值的方式落地到用户手中。

(00:18:54):

所以今天绝大多数用户发现 Codex 的方式是：要么下载 IDE 扩展，要么在 CLI（命令行）中运行，智能体就在你的电脑上与你交互。它在一个沙箱（sandbox）中运行，这是一项非常酷的技术，能确保安全，同时它能访问所有依赖项。如果智能体需要执行命令，它可以在沙箱内完成，我们不需要设置任何环境。如果某个命令在沙箱中行不通，它会直接问你。这样你就进入了一个非常强大的反馈循环。随着时间的推移，我们团队的工作就是帮助你通过使用产品，顺便完成配置，从而让你以后能把任务委派给它。

(00:19:38):

再用那个比喻：如果你雇了一个队友，却只给他们一台刚从商店买来的新电脑，他们很难开展工作。但如果你和他们并肩工作，告诉他们：“噢，你没有这个服务的密码，给你。没关系，尽管运行这个命令。”那么之后他们就能离开你独立工作好几个小时。

(00:20:01) Lenny Rachitsky

English:

So what I'm hearing is the initial version of Codex was almost too far in the future. It's like a remote in the cloud agent that's coding for you asynchronously. And what you did is, "Okay, let's actually come back a little bit, let's integrate into the way engineers already integrate into IDs and locally and help them on ramp to this new world,"

中文翻译:

所以我听下来，Codex 的最初版本几乎太超前了。它像是一个云端的远程智能体，异步地为你写代码。而你们做的是：“好吧，让我们往回退一点，集成到工程师现有的 IDE 和本地工作流中，帮助他们平滑过渡到这个新世界。”

(00:20:21) Alexander Embiricos

English:

Totally. And it was quite interesting because we dogfood product a ton at OpenAI. So dogfood as in we use our own product. And so Codex has been accelerating OpenAI over the course of the entire year, and the cloud product was a massive accelerant to the company as well. It just turns out that this was one of those places where the signal we got from dogfooding is a little bit different from the signal you get from the general market because at OpenAI, we train reasoning models all day and so we're very used to this kind of prompting and think upfront, run things massively in parallel and it would take some time and then come back to it later asynchronously. And so now when we build, we still get a ton of signal from dogfooding internally, but we're also very cognizant of the different ways that different audiences use the product.

中文翻译:

完全正确。这很有趣，因为在 OpenAI 我们大量进行“吃自家狗粮”（dogfooding），也就是使用我们自己的产品。Codex 在过去一年里一直在加速 OpenAI 的内部开发，云端产品对公司来说也是一个巨大的加速器。只是事实证明，内部测试得到的信号与大众市场的信号有些不同。因为在 OpenAI，我们整天都在训练推理模型，所以我们非常习惯这种提示方式：预先思考好，大规模并行运行，等一段时间，然后再异步回来查看结果。所以现在我们构建产品时，虽然仍从内部测试中获得大量信号，但也非常清醒地意识到不同受众使用产品的不同方式。

(00:21:12) Lenny Rachitsky

English:

That's really funny. It's like live in the future, but maybe not too far in the future. And I could see how everyone at OpenAI is living very far in the future, and sometimes that won't work for everyone.

中文翻译:

这很有趣。就像是“生活在未来，但别太远”。我能想象 OpenAI 的每个人都生活在遥远的未来，而这有时并不适合所有人。

(00:21:23) Alexander Embiricos

English:

Yeah.

中文翻译:

是的。

(00:21:23) Lenny Rachitsky

English:

What about just intelligence training data? I don't know, is there something else that helped Codex accelerate its ability to actually code? Is it better, cleaner data? Is it more just models advancing? Is there anything else that really helped accelerate?

中文翻译:

那智能训练数据方面呢？有没有其他因素帮助 Codex 提升了实际编程能力？是更好、更干净的数据？还是仅仅因为模型本身的进步？还有什么真正起到了加速作用？

(00:21:38) Alexander Embiricos

English:

Yeah, so there's a few components here. I guess you were mentioning models and the models have improved a ton. In fact, just last Wednesday, we shipped GPT-5.1-Codex-Max, a very accurately named model, that is awesome. It is awesome both because it is for any given task that you were using GPT-5.1-Codex for, it's roughly 30% faster at accomplishing that task. But also it unlocks a ton of intelligence. So if you use it at our higher reasoning levels, it's just even smarter. And that tweet you were saying Karpathy made about, "Hey, give this your gnarliest bugs," obviously there's a ton going on in the market right now, but Codex-Max is definitely carrying that mantle of us tackling the hardest bugs. So that is super cool.

(00:22:28):

But I will say it's like some of how we're thinking about this is evolving a little bit from being like, "Yeah, we're just going to think about the model and let's just train the best model," to really thinking about what is an agent actually overall? And I'm not going to try to define agent exactly, but at least the stack that we think of it as having is it's like you have this model, really smart reasoning model that knows how to do a specific kind of task really well, so we can talk about how we make that possible. But then actually

we need to serve that model through an API into a harness, and both of those things also have a really big role here.

(00:23:02):

So for instance, one of the things that we're really proud of is you can have GPT-5.1-Codex-Max work for really long periods of time. That's not normal, but you can set it up to do that or that might happen. But now routinely we'll hear about people saying, "Yeah, it ran overnight or it ran for 24 hours." And so for a model to work continuously for that amount of time, it's going to exceed its context window. And so we have a solution for that, which we call compaction.

(00:23:28):

But compaction is actually a feature that uses all three layers of that stack. So you need to have a model that has a concept of compaction and knows like, "Okay, as I start to approach this context window, I might be asked to prepare to be run in a new context window." And then at the API layer, you need an API that understands this concept and has an endpoint that you can hit to do this change. And at the harness layer, you need a harness that can prepare the payload for this to be done. So shipping this compaction feature that now just made this behavior possible to anyone using Codex actually meant working across all three things. And I think that's increasingly going to be true.

(00:24:02):

Another maybe underappreciated version of this is if you think about all the different coding products out there, they all have very different tool harnesses with very different opinions on how the model should work. So if you want to train a model to be good at all the different ways it could work, maybe you have a strong opinion that it should work using semantic search. Maybe you have a strong opinion that it should call bespoke tools or maybe you have, in our case, a strong opinion that it should just use the shell and work in the terminal, you can move much faster if you're just optimizing for one of those worlds. So the way that we built Codex is that it just uses the shell, but in order to make that safer and secure, we have a sandbox that the model is used to operating in.

(00:24:45):

So I think one of the biggest accelerants, to go all the way back to answer to your question, is just we're building all three things in parallel and tuning each one and constantly experimenting with how those things work with a tightly integrated product and research team.

中文翻译:

是的，这里有几个组成部分。你提到了模型，模型确实进步巨大。事实上，就在上周三，我们发布了 GPT-5.1-Codex-Max，一个命名非常准确的模型，它非常棒。它之所以棒，是因为对于你之前用 GPT-5.1-Codex 处理的任何任务，它的完成速度大约快了 30%。同时它还释放了巨大的智能。如果你在更高的推理级别使用它，它会变得更聪明。你提到的 Karpathy 那条关于“把最棘手的 bug 交给它”的推文，虽然现在市场上竞争激烈，但 Codex-Max 绝对继承了这一衣钵，专门对付最难的 bug。这非常酷。

(00:22:28):

但我得说，我们的思考方式正在发生演变，不再只是“我们要考虑模型，我们要训练最好的模型”，而是真正思考“一个智能体（agent）整体上到底是什么？”我不想给智能体下精确定义，但至少我们认为它拥有的技术栈是这样的：你有一个模型，一个非常聪明的推理模型，它知道如何出色地完成特定任务。但随后我们需要通过 API 将该模型提供给一个“外壳”（harness），这两者在这里也扮演着重要角色。

(00:23:02):

例如，我们非常自豪的一点是，你可以让 GPT-5.1-Codex-Max 连续工作很长时间。这并不寻常，但你可以这样设置。现在我们经常听到人们说：“是的，它跑了一整夜”或者“它跑了 24 小时”。对于一个模型来说，连续

工作这么长时间会超出它的上下文窗口（context window）。为此我们有一个解决方案，称之为“压缩”（compaction）。

(00:23:28):

但“压缩”实际上是一个涉及技术栈所有三层的功能。你需要一个具有“压缩”概念的模型，它知道：“OK，当我接近上下文窗口限制时，我可能会被要求准备好在新的窗口中运行。”在 API 层，你需要一个理解这个概念并拥有相应端点的 API。在“外壳”层，你需要一个能准备好执行此操作所需数据的环境。所以，发布这个让所有 Codex 用户都能实现长时运行的“压缩”功能，实际上意味着要跨越这三层进行开发。我认为这种情况会越来越多。

(00:24:02):

另一个可能被低估的方面是：如果你想想市面上所有的编程产品，它们都有非常不同的工具环境，对模型应该如何工作有非常不同的见解。如果你想训练一个模型在所有工作方式下都表现出色——也许你坚持认为它应该使用语义搜索，或者你坚持认为它应该调用定制工具，或者像我们一样，坚持认为它应该直接使用 Shell 并在终端工作——如果你只针对其中一种模式进行优化，你的进展会快得多。我们构建 Codex 的方式是让它直接使用 Shell，但为了确保安全，我们提供了一个模型习惯操作的沙箱。

(00:24:45):

所以，回到你的问题，最大的加速器之一就是：我们通过一个高度整合的产品和研究团队，并行构建这三样东西，调整每一项，并不断实验它们如何协同工作。

(00:24:59) Lenny Rachitsky

English:

Do you think you win in this space? Do you think it'll always be this kind of race with other models constantly leapfrogging each other? Do you think there's a world where someone just runs away with it and no one else can ever catch up? Is there a path to just, "We win"?

中文翻译:

你认为你们能在这个领域胜出吗？你觉得这会一直是一场其他模型不断互相超越的竞赛吗？你认为会出现某个人遥遥领先，其他人再也追不上的局面吗？是否存在一条“我们赢了”的路径？

(00:25:15) Alexander Embiricos

English:

Again, comes back to this idea of building a teammate, and not just a teammate that participates in team planning and prioritization, not just a teammate that really tests its code and helps you maintain and deploy it. But even a teammate... If you think, again, an engineering teammate, they can also schedule a calendar invite or move standup or do whatever, right? And so in my mind, if we just imagine that every day or every week some crazy new capability is just going to be deployed by a research lab, it's just impossible for us as humans to keep up and use all this technology. So I think we need to get to this world where you kind of just have an AI teammate or super assistant that you just talk to and it just knows how to be helpful on its own. So you don't have to be reading the latest tips for how to use it, you've plugged it in and it just provides help.

(00:26:10):

So that's kind of the shape of what I think we're building. And I think that will be a very sticky winning product if we can do so. So the shape that in my head, at least I have, is that we build... Maybe a fun topic

is like, "Is Chat the right interface for AI?" I actually think Chat is a very good interface when you don't know what you're supposed to use it for. In the same way that if I think of I'm on MS Teams or in Slack with a teammate, Chat is pretty good. I can ask for whatever I want. It's kind of the common denominator for everything. So you can chat with a super assistant about whatever topic you want, whether it be coding or not. And then if you are a functional expert in a specific domain such as coding, there's a GUI that you can pull up to go really deep and look at the code and work with the code.

(00:26:54):

So I think what we need to build as OpenAI is basically this idea of you have Chat, ChatGPT and not as a tool that's ubiquitously available to everyone, you start using it even outside of work to just help you. You become very comfortable with the idea of being accelerated with AI. So then you get to work and you just can naturally just, "Yeah, I'm just going to ask it for this and I don't need to know about all the connectors or all the different features. I'm just going to ask it for help and it'll surface to me the best way that it can help at this point in time and maybe even chime in when I didn't ask it for help." So in my mind, if we can get to that, I think that's how we really build the winning product.

中文翻译:

这又要回到构建“队友”的想法上了。不仅是一个参与团队规划和优先级排序的队友，不仅是一个会测试代码并帮你维护部署的队友。甚至……如果你想想一个工程队友，他们还能安排日历邀请、调整站会时间等等，对吧？在我看来，如果我们只是想象每天或每周研究实验室都会发布某种疯狂的新能力，我们人类根本不可能跟上并利用好所有这些技术。所以我们需要进入这样一个世界：你拥有一个 AI 队友或超级助手，你只需和它交谈，它自己就知道如何提供帮助。你不需要阅读最新的使用技巧，只要接入它，它就会提供帮助。

(00:26:10):

这就是我认为我们正在构建的东西的雏形。如果我们能做到这一点，那将是一个非常有粘性的获胜产品。我脑海中的构思是……也许一个有趣的话题是：“聊天（Chat）是 AI 的正确界面吗？”我其实认为，当你不知道该用它做什么时，聊天是一个非常好的界面。就像我在 MS Teams 或 Slack 上和队友交流一样，聊天很好用，我可以提任何要求，它是所有事情的公约数。所以你可以就任何话题（无论是否关于编程）与超级助手聊天。而如果你是某个特定领域（如编程）的专家，你可以调出一个图形界面（GUI）来深入查看和操作代码。

(00:26:54):

所以我觉得 OpenAI 需要构建的基本上是这样一种理念：你拥有 ChatGPT，它不仅是一个随处可见的工具，你甚至在工作之外也用它来提供帮助。你变得非常习惯于被 AI 加速。当你开始工作时，你会很自然地觉得：“是的，我就问它这个，我不需要了解所有的连接器或各种功能。我只是寻求帮助，它会向我展示此时此刻提供帮助的最佳方式，甚至在我没开口时主动介入。”在我看来，如果我们能做到这一点，那就是我们打造获胜产品的方式。

(00:27:32) Lenny Rachitsky

English:

This is so interesting because with my chat with Nick Turley, the head of ChatGPT, I think he shared that the original name for ChatGPT was Super Assistant or something like that. And it's interesting that there's that approach to the super assistant and then there's this Codex approach. It's almost like the B2C version and the B2B version. And what I'm hearing is the idea here is, okay, you start with coding and building and then it's doing all this other stuff for you, scheduling meetings, I don't know, probably posting in Slack, I don't know, shipping designs. I don't know, is the idea that this is the business version of ChatGPT in a sense, or is there something else there?

中文翻译:

这太有意思了，因为在我和 ChatGPT 负责人 Nick Turley 聊天时，他提到 ChatGPT 最初的名字可能是“超级助手”（Super Assistant）之类的。有趣的是，既有那种超级助手的路径，又有 Codex 这种路径。这有点像 B2C 版本和 B2B 版本。我听到的意思是：OK，你从编程和构建开始，然后它为你做所有其他事情——安排会议、在 Slack 发帖、交付设计。这在某种意义上是 ChatGPT 的商业版吗？还是有别的含义？

(00:28:08) Alexander Embiricos

English:

Yeah. So we're getting to the one-year time horizon conversation. A lot of this might happen sooner, but in terms of fuzziness, I think we're at the one year. So I'll give you a contention and a plausible way we get there, but as for how it happens, who knows? So basically, if we're going to build a super assistant, it has to be able to do things. So we're going to have a model and it's going to be able to do stuff affecting your world. And one of the learnings I think we've seen over the past year or so is that for models to do stuff, they're much more effective when they can use a computer.

(00:28:41):

Right, okay, so now we're like, okay, we need the super assistant that can use a computer, or many computers. And now the question is, okay, well, how should it use the computer? And there's lots of ways to use a computer. You could try to hack the OS and use accessibility APIs, maybe a bit easier as you could point and click. That's a little slow and unpredictable sometimes. And another way, it turns out the best way for models to use computers is simply to write code. So we're kind of getting to this idea where, well, if you want to build any agent, maybe you should be building a coding agent and maybe to the user, a non-technical user, they won't even know they're using a coding agent, the same way that no one thinks about are they using the internet or not, which is they're more just like, "Is WiFi on?"

(00:29:23):

So I think that what we're doing with Codex is we're building a software engineering teammate, and as part of that, we're kind of building an agent that can use a computer by writing code. And so we're already seeing some pull for this. It's quite early, but we're starting to see people who are using Codex for coding adjacent product purposes. And so as that develops, I think we'll just naturally see that, oh, it turns out we should just always have the agent write code if there is a coding way to solve a problem instead of... Even if you're doing a financial analysis, maybe write some code for that.

(00:29:55):

So basically like you were like, "Hey, is this the two ends of this product for the super assistant of ChatGPT?" In my mind, just coding is a core competency of any agent including ChatGPT. And so really what we think we're building is that competency. So here's the really cool thing about agents writing code is that you can import code. Code is composable, interoperable. Because one very reductive view we could have for an agent is it's just going to be given a computer and it's just going to point and click and go around. But that is the future. And then how we get there is difficult to chart a path because a lot of the questions around building agents aren't like, "Can the agent do it?" But it's more about, "Well, how can we help the agent understand the context that it's working in?" And the team that's using it probably has a way that they like to do things. They have guidelines. They probably want certain deterministic guarantees about what the agent can or cannot do. Or they want to know that the agent understands this detail.

(00:30:57):

An example would be if we're looking at a crash reporting tool, hitting a connector for it, every sub-team probably has a different meta prompt for how they want the crashes to be analyzed. And so we start to

get to this thing where, yeah, we have this agent sitting in front of a computer, but we need to make that configurable for the team or for the user and let them... Stuff that the agent does often, we probably just want to build in as a competency that this agent has that it can do.

(00:31:24):

So I think we end up with this generalizable thing, that you were saying, of an agent that can just write its own scripts for whatever it wants to do. But I think that the really key part here is can we make it so that everything that the agent has to do often or that it does well, we can just remember and store so that the agent doesn't have to write a script for that again? Or maybe if I just joined a team and you are already on the same team as me, I can just use all those scripts that the agents had written already.

中文翻译:

是的。我们现在聊到了“一年期”的话题。很多事情可能会发生得更快，但就模糊性而言，我觉得我们是在看一年后的样子。我给你一个论点和一种可能的实现路径，但至于具体怎么发生，谁知道呢？基本上，如果我们想构建一个超级助手，它必须能够“做事”。所以我们要有一个模型，它能做一些影响你现实世界的事情。过去一年我们学到的一点是：模型要做事，如果能操作电脑，效率会高得多。

(00:28:41):

好，现在我们需要一个能操作电脑（或多台电脑）的超级助手。问题是，它该如何操作电脑？有很多方法。你可以尝试破解操作系统并使用辅助功能 API；或者简单点，让它模拟点击。但这有时很慢且不可预测。另一种方式，事实证明模型操作电脑的最佳方式就是编写代码。所以我们得出了这样一个想法：如果你想构建任何智能体，也许你应该构建一个编程智能体。对于非技术用户来说，他们甚至不知道自己在用编程智能体，就像没人会想自己是不是在用互联网，他们只会想“WiFi 连上了吗？”

(00:29:23):

所以我觉得我们在 Codex 上做的是构建一个软件工程队友，作为其中的一部分，我们实际上是在构建一个能通过写代码来操作电脑的智能体。我们已经看到了一些这方面的需求。虽然还很早，但我们开始看到有人将 Codex 用于编程相关的产品用途。随着这种趋势的发展，我想我们会自然而然地发现：如果有一个编程的方法能解决问题，我们就应该让智能体去写代码，而不是……哪怕你是在做财务分析，也许也该为此写点代码。

(00:29:55):

所以，回到你说的“这是否是 ChatGPT 超级助手的两端”，在我看来，编程是任何智能体（包括 ChatGPT）的核心能力。我们认为我们正在构建的就是这种能力。智能体写代码最酷的一点是：代码是可以导入的，是可组合的，是可互操作的。对智能体的一种非常简化的看法是，给它一台电脑，让它点点划划。但那是未来。如何到达那里很难规划路径，因为关于构建智能体，很多问题不在于“智能体能不能做”，而在于“我们如何帮助智能体理解它工作的上下文？”使用它的团队可能有自己的做事方式、准则，可能希望对智能体能做什么或不能做什么有确定的保证，或者希望智能体理解某些细节。

(00:30:57):

举个例子，如果我们看一个崩溃报告工具，每个子团队可能都有不同的元提示词（meta prompt）来规定他们希望如何分析崩溃。所以我们开始进入这样一个阶段：是的，我们有一个坐在电脑前的智能体，但我们需要让它对团队或用户是可配置的，让他们……对于智能体经常做的事情，我们可能只想把它内置为这个智能体具备的一项能力。

(00:31:24):

所以我觉得我们最终会得到你所说的那种通用的东西：一个能为任何想做的事编写脚本的智能体。但关键在于，我们能否让智能体经常做或做得好的事情被记住和存储，这样它就不必再次编写脚本？或者如果我刚加入一个团队，而你已经在团队里了，我可以重用智能体之前为你写的所有脚本。

(00:31:53) Lenny Rachitsky

English:

Yeah, it's like if this is our teammate, they can share things that it's learned from working with other people at the company. It just makes sense as a metaphor.

中文翻译:

是的，就像如果这是我们的队友，它可以分享从公司其他人那里学到的东西。这个比喻非常合理。

(00:32:01) Alexander Embiricos

English:

Right. Yeah.

中文翻译:

没错。

(00:32:02) Lenny Rachitsky

English:

It feels like you're in the Karpathy camp of, "Agents today are not that great and mostly slop and maybe in the future they'll be awesome." Does that resonate?

中文翻译:

感觉你属于 Karpathy 的那个阵营，即“今天的智能体还不怎么样，大多是些粗制滥造的东西（slop），也许未来会很棒”。这能引起你的共鸣吗？

(00:32:11) Alexander Embiricos

English:

So I think coding agents are pretty great. I think we're seeing a ton of value there.

中文翻译:

我觉得编程智能体已经非常棒了。我们已经看到了巨大的价值。

(00:32:11) Lenny Rachitsky

English:

Yeah, that feels right. That feels right, yeah.

中文翻译:

是的，这感觉没错。

(00:32:17) Alexander Embiricos

English:

And then I think agents outside of coding, it's still very early. And this is just my opinion, but I think they're going to get a whole lot better once they can use coding too in a composable way. It's kind of the fun part of when you're building for software engineers, at my startup, we were building for software engineers too for a lot of that journey, and they're just such a fun audience to build for because they also like building for themselves and are often even more creative than we are in thinking about how to use the technology. So by building for software engineers, you get to just observe a ton of emergent behaviors and things that you should do and build into the product.

中文翻译:

至于编程之外的智能体，现在还处于非常早期。这只是我的个人观点，但我认为一旦它们也能以可组合的方式利用编程能力，它们会变得好得多。为软件工程师构建产品最有趣的地方在于（在我之前的初创公司，大部分时间也是在为工程师服务），他们是一个非常有意思的受众，因为他们也喜欢为自己构建工具，而且在思考如何使用技术方面往往比我们更有创意。所以通过为工程师构建产品，你可以观察到大量的涌现行为，以及你应该做并内置到产品中的功能。

(00:32:54) Lenny Rachitsky

English:

I love how you say that because a lot of people building for engineers get really annoyed because the engineers they're just always complaining about stuff. They're like, "Ah, that sucks. Why'd you build it this way?" I love that you enjoy it, but I think it's probably because you're building such an amazing tool for engineers that can actually solve problems and just code for them.

中文翻译:

我很喜欢你这么说，因为很多为工程师构建产品的人会觉得很烦，因为工程师总是在抱怨。他们会说：“啊，这太烂了，你为什么要这么设计？”我很高兴你乐在其中，但我猜这可能是因为你正在为他们构建一个如此惊人的工具，能真正解决问题并直接为他们写代码。

(00:33:12) Lenny Rachitsky

English:

Kind of along those lines, there's always this talk of what will happen with jobs, engineers, coding, do you have to learn coding? All these things. Clearly the way you're describing it is it's a teammate, it's going to work with you, make you more superhuman, it's not going to replace you. What's the way you just think about the impact on the field of engineering, having this super intelligent engineering teammate?

中文翻译:

顺着这个话题，总有人在讨论工作、工程师、编程的未来会怎样，是否还需要学习编程等等。显然，你描述它的方式是一个队友，它会与你合作，让你变得更像“超人”，而不是取代你。你如何看待拥有这样一个超级智能的工程队友对工程领域产生的影响?

(00:33:33) Alexander Embiricos

English:

I think there's two sides to it, but the one we were just talking about is this idea that maybe every agent should actually use code and be a coding agent. And in my mind, that's just a small part of this broader idea that, hey, as we make code even more ubiquitous... I mean, you could probably claim it's ubiquitous

today, even pre AI, right? But as we make code even more ubiquitous, it's actually just going to be used for many more purposes. And so there's just going to be a ton more need for humans with this competency.

(00:34:01):

So that's my view. I think this is quite a complex topic. So it's something we talk about a lot and we have to see how it pans out. But I think what we can do basically as a product team building in the space is just try to always think about how are we building a tool so that it feels like we're maximally accelerating people rather than building a tool that makes it more unclear what you should do as the human?

(00:34:27):

I think, to give an example right now, nowadays when you work with a coding agent, it writes a ton of code, but it turns out writing code is actually one of the most fun parts of software engineering for many software engineers. So then you end up reviewing AI code. And that's often a less fun part of the job for many software engineers. So I actually think we see that this plays out all the time in a ton of micro decisions. So we as a product team, we're always thinking about, "Okay, how do we make this more fun? How do we make you feel more empowered? Where is this not working?" And I would argue that reviewing agent written code is a place that today is less fun.

(00:35:04):

So then I think, "Okay, what can we do about that?" Well, we can ship a code review feature that helps you build confidence in the AI written code. Okay, cool. Another thing we could do is we can make it so that the agent's better able to validate its work. And it gets all the way down into micro decisions. If you're going to have an agent capability to validate work, and let's say you have... I'm thinking of Codex Web right now, you have a pain that sort of reflects the work the agent did, what do you see first? Do you see the diff or do you see the image preview of the code it wrote? And I think if you're thinking about this from perspective, "How do I empower the human? How do I make them feel as accelerated as possible?" You obviously see the image first. You shouldn't be reviewing the code unless first you've seen the image, unless maybe it's been reviewed by an AI and now it's time for you to take a look.

中文翻译:

我认为这有两个方面。一方面是我们刚才聊到的，也许每个智能体都应该使用代码并成为编程智能体。在我看来，这只是一个更宏大想法的一小部分：随着我们让代码变得更加无处不在（虽然在 AI 出现之前它已经无处不在了），它实际上会被用于更多的用途。因此，对具备这种能力的人类的需求只会大大增加。

(00:34:01):

这是我的观点。这是一个相当复杂的话题，我们经常讨论，必须观察它如何演变。但作为在这个领域构建产品的团队，我们基本上能做的就是始终思考：我们如何构建一个工具，让人们感到效率得到了最大化的提升，而不是构建一个让作为人类的你更不清楚该做什么的工具？

(00:34:27):

举个现在的例子：如今当你和编程智能体合作时，它会写大量代码。但事实证明，对许多软件工程师来说，写代码其实是软件工程中最有趣的部分之一。结果现在你变成了在审查 AI 写的代码，而这通常是工作中没那么有趣的部分。我认为这体现在无数的微观决策中。作为产品团队，我们总是在想：“如何让这变得更有趣？如何让你感到更有掌控感？哪里做得不好？”我认为审查智能体编写的代码在今天就是一个没那么有趣的环节。

(00:35:04):

那么我想，“OK，我们能为此做些什么？”我们可以发布一个代码审查功能，帮助你建立对 AI 编写代码的信心。太棒了。另一件事是，我们可以让智能体更好地验证自己的工作。这甚至深入到微观决策：如果你有一个能验证工作的智能体能力，比如在 Codex Web 中，你有一个展示智能体工作成果的面板，你首先看到的是什

么？是代码差异（diff），还是它编写的代码生成的图像预览？如果你从“如何赋能人类、如何让他们感到最快加速”的角度出发，你显然应该先看到图像。除非你先看到了图像，或者它已经被 AI 审查过了，否则你不应该直接去审查代码。

(00:35:49) Lenny Rachitsky

English:

When I had Michael Truell, the CEO of Cursor on the podcast, he had this kind of vision of us moving to something beyond code. And I've seen this rise of something called spec-driven development where you just write the spec and then the AI writes code for you. So you start working at this higher abstraction level. Is that something you see where we're going, just like engineers not having to actually write code or look at code and there's going to be this higher level of abstraction that we focus on?

中文翻译:

当我邀请 Cursor 的 CEO Michael Truell 上播客时，他提出了一种愿景，即我们将走向超越代码的阶段。我也看到一种被称为“规范驱动开发”（spec-driven development）的兴起，即你只需编写规范（spec），然后 AI 为你编写代码。这样你就开始在更高的抽象层级上工作。你认为这是我们的发展方向吗？即工程师不再需要实际编写或查看代码，我们将专注于更高层级的抽象？

(00:36:16) Alexander Embiricos

English:

Yeah. I mean, I think there's constantly these levels of abstraction and they're actually already played out today. Today, coding agents, mostly it's prompt to patch. We're starting to see people doing spec-driven development or planned and driven development. That's actually one of the ways when people ask, "Hey, how do you run Codex on a really long task?" Well, it's like often collaborate with it first to write a plan.md, like a markdown file that's your plan. And once you're happy with that, then you ask it to go off and do work. And if that plan has verifiable steps, it'll work for much longer. So we're totally seeing that.

(00:36:50):

I think spec-driven development is an interesting idea. It's not clear to me that it'll work out that way because a lot of people don't like writing specs either, but it seems plausible that some people will work that way. A bit of a joke idea though is if you think of the way that many teams work today, they often don't necessarily have specs, but the team is just really self-driven and so stuff just gets done. And so almost that it's like, I'm coming up with this on the spot, so it's not a good name, but chatter-driven development where it's just like stuff is happening on social media and in your team communications tools. And then as a result, code gets written and deployed.

(00:37:29):

So yeah, I think I'm a little bit more oriented in that way of I don't even necessarily want to have to write a spec. Sometimes I want to, only if I like writing specs. Other times I might just want to say like, "Hey, here's the customer service channel and tell me what's interesting to know, but if it's a small bug, just fix it." I don't want to have to write a spec for that, right?

(00:37:51):

I have this sort of hypothetical future that I like to share sometimes with people as a provocation, which is in a world where we have truly amazing agents, what does it look like to be a solopreneur? And one terrible idea for how it could look is that actually there's a mobile app and every idea that the agent has

to do is just vertical video on your phone and then you can swipe left if you think it's a bad idea and you can swipe right if it's a good idea. And you can press and hold and speak to your phone if you want to give feedback on the idea before you swipe. And in this world, basically what your job is is just to plug in this app into every single signal system or system of record, and then you just sit back and swipe. I don't know.

中文翻译:

是的。我认为抽象层级一直在不断演进，而且今天已经在发生了。现在的编程智能体大多是“提示词转补丁”（prompt to patch）。我们开始看到人们进行规范驱动开发或计划驱动开发。事实上，当人们问“如何让 Codex 执行超长任务”时，一种方法就是先和它协作写一个 `plan.md`（Markdown 格式的计划文件）。一旦你对计划满意，再让它去执行。如果计划中有可验证的步骤，它能持续工作更久。所以我们完全看到了这种趋势。

(00:36:50):

我觉得规范驱动开发是个有趣的想法，但我不确定它是否会成为主流，因为很多人也不喜欢写规范。不过，确实可能有人会这样工作。我有一个开玩笑的想法：如果你想想现在很多团队的工作方式，他们不一定有规范，但团队非常有自觉性，事情自然就办成了。这几乎像是——我现编一个词，可能不太好听——“闲聊驱动开发”（chatter-driven development），即社交媒体上或团队沟通工具里正在讨论某些事，结果代码就被写出来并部署了。

(00:37:29):

所以，我更倾向于这种方式：我甚至不一定想写规范。除非我喜欢写，否则我可能只想说：“嘿，这是客服频道，告诉我有什么值得关注的，如果是小 bug 就直接修了。”我不想为此写个规范，对吧？

(00:37:51):

我有一个假设的未来场景，有时会分享给别人作为一种启发：在一个拥有真正惊人智能体的世界里，一个“超级个体户”（solopreneur）是什么样的？一个有点糟糕但有趣的想法是：有一个手机 App，智能体产生的每一个想法都以竖屏短视频的形式呈现在你手机上，觉得主意烂就左滑，觉得好就右滑。如果你想在滑动前给点反馈，就长按并对着手机说话。在这个世界里，你的工作基本上就是把这个 App 接入每一个信号系统或记录系统，然后你只需坐下来不停地滑动。我不知道这算不算好。

(00:38:39) Lenny Rachitsky

English:

I love this. So this is like Tinder meets TikTok meets Codex.

中文翻译:

我太喜欢这个了。这简直是 Tinder 遇见 TikTok 再遇见 Codex。

(00:38:42) Alexander Embiricos

English:

It's pretty terrible.

中文翻译:

这听起来挺可怕的。

(00:38:43) Lenny Rachitsky

English:

No, this is great. So the idea here is this agent is watching and listening to you, paying attention to the market, your users, and it's like, "Cool, here's something I should do." It's like a proactive engineer just like, "Here, we should build this feature, fix this thing."

中文翻译:

不，这太棒了。所以这里的核心是，这个智能体在观察你、倾听你，关注市场和你的用户，然后它会说：“酷，这是我该做的事。”它就像一个主动的工程师，会说：“嘿，我们应该开发这个功能，或者修复这个东西。”

(00:38:56) Alexander Embiricos

English:

Exactly. Exactly.

中文翻译:

没错，正是如此。

(00:38:58) Lenny Rachitsky

English:

I think it's a really good idea.

中文翻译:

我觉得这真的是个好主意。

(00:39:00) Alexander Embiricos

English:

Communicating with you in the lowest effort way for your consumers.

中文翻译:

以对你（消费者）来说成本最低的方式进行沟通。

(00:39:02) Lenny Rachitsky

English:

Yeah, yeah, the modern way we communicate, swipe left to right and vertical feed. And then the Sora video, okay, so I see how this all connects now. I see.

中文翻译:

是啊，现代人的沟通方式：左右滑动和竖屏流。再加上 Sora 视频，OK，我现在明白这一切是怎么联系起来的了。

(00:39:11) Alexander Embiricos

English:

Yeah. To be clear, we're not building that, but it's a fun idea. I mean, in this example though, one of the things that it's doing is it's consuming external signals, right? I think the other really interesting thing is if we think about what is the most successful AI product to date, I would argue, it's funny actually not to confuse things at all, but the first time we used the brand Codex at OpenAI was actually the model powering GitHub Copilot. This is way back in the day, years ago. And so we decided to reuse that brand recently because it's just so good, Codex, code execution.

(00:39:46):

But I think actually auto completion and IDEs is one of the most successful AI products today. And part of what's so magical about it is that when it can surface ideas for helping you really rapidly, when it's right, you're accelerated. When it's wrong, it's not that annoying. It can be annoying, but it's not that annoying. So you can create this mixed initiative system that's contextually responding to what you're attempting to do.So in my mind, this is a really interesting thing for us as OpenAI as we're building.

(00:40:22):

So for instance, when I think about launching a browser, which we did with Atlas, in my mind, one of the really interesting things we can then do is we can then contextually surface ways that we can help you as you're going about your day. And so we break out of this, we're just looking at code or we're just in your terminal into this idea that, "Hey, a real teammate is dealing with a lot more than just code. They're dealing with a lot of things that are web content. So how can we help you with that?"

中文翻译:

是的。澄清一下，我们并没在做那个 App，但这确实是个有趣的想法。在这个例子中，它做的一件事是消耗外部信号，对吧？另一件有趣的事是，如果我们思考至今最成功的 AI 产品是什么，我会认为——有趣的是，为了不引起混淆，OpenAI 第一次使用 Codex 这个品牌其实是作为驱动 GitHub Copilot 的模型，那是好几年前的事了。我们最近决定重新启用这个品牌，因为它太棒了：Codex，代码执行（Code Execution）。

(00:39:46):

但我认为 IDE 中的自动补全确实是当今最成功的 AI 产品之一。它的神奇之处在于，当它能飞快地为你提供帮助建议时，如果它是对的，你就被加速了；如果它是错的，也没那么烦人。虽然有时会烦，但程度有限。所以你可以创建一个“混合主动性”（mixed initiative）系统，根据你试图做的事情进行上下文响应。在我看来，这对我们 OpenAI 正在构建的东西来说非常有趣。

(00:40:22):

例如，当我想到发布浏览器（就像我们用 Atlas 所做的那样）时，我认为其中一件非常有趣的事是：我们可以在你处理日常事务时，根据上下文呈现帮助你的方式。这样我们就跳出了“只看代码”或“只在终端”的局限，进入了这样一个理念：“嘿，一个真正的队友处理的不只是代码。他们处理很多网页内容。那么我们该如何在那方面帮你呢？”

(00:40:51) Lenny Rachitsky

English:

Man, there's so much there. I love this. Okay, so auto complete on web with the browser. That's so interesting. Just like, "Here's all the things that we can help you with as you're browsing and going about your day."

(00:41:01):

I want to talk about Atlas. I'll come back to that. Codex, code execution, did not know that. That's really clever. I get it now. Okay, and then this chatter, what is a chatter-driven development? No, this is a really good idea, but it reminds me, I had Dhanji on the podcast, CTO of Block, and they have this product called Goose, which is their own internal agent thing. And he talked about an engineer at Block just has Goose watch him with his screen and listens to every meeting and proactively does work that he should probably want to do. So ships to PR, sends an email, drafts a Slack message. So he's doing exactly what you're describing in kind of a very early way.

中文翻译:

天哪，信息量太大了。我喜欢这个。OK，浏览器上的网页自动补全，这太有意思了。就像是“当你浏览网页处理日常事务时，这些是我们能帮你的所有事情”。

(00:41:01):

我想聊聊 Atlas，稍后回来。Codex 代表代码执行（Code Execution），我以前还真不知道，这很聪明。我明白了。还有那个“闲聊驱动开发”，这主意真不错。它让我想起我曾邀请 Block 的 CTO Dhanji 上播客，他们有一个叫 Goose 的内部智能体产品。他说 Block 的一名工程师就让 Goose 观察他的屏幕并旁听每一场会议，然后主动去做他可能想做的工作：比如提交 PR、发邮件、起草 Slack 消息。他正在以一种非常早期的方式实践你所描述的东西。

(00:41:45) Alexander Embiricos

English:

Yeah, that's super interesting. And I bet you, so if we went and asked them what the bottleneck to that productivity is, did they share what it is?

中文翻译:

是的，那非常有趣。我敢打赌，如果我们去问他们这种生产力的瓶颈是什么，他们有提到吗?

(00:41:54) Lenny Rachitsky

English:

Probably looking at it and just making sure this is the right thing to do, yeah.

中文翻译:

大概是查看它做的事，并确保那是正确的事，没错。

(00:41:58) Alexander Embiricos

English:

Yeah. So we see this now. We have a Slack integration for Codex. People love if there's something that you need to do quickly, people will just @ mention Codex, "Why do you think this bug is happening?" It doesn't have to be an engineer. Even maybe data scientists often here are using Codex a ton to just answer questions like, "Why do you think this metric moved? What happened?" So questions, you get the answer right back in Slack. It's amazing, super useful. But as for when it's writing code, then you have to go back and look at the code.

(00:42:25):

So the real, I think, bottleneck right now is validating that the code worked and writing code review. So in my mind, if we wanted to get to something like the friend you were talking about's world, I think we really need to figure out how to get people to configure their coding agents to be much more autonomous on those later stages of the work.

中文翻译:

是的。我们现在也看到了这一点。我们有 Codex 的 Slack 集成。人们非常喜欢，如果有需要快速处理的事，大家就会 @Codex 问：“你觉得这个 bug 为什么会发生？”不一定是工程师，甚至数据科学家也经常用 Codex 来回答诸如“你觉得这个指标为什么变动了？发生了什么？”之类的问题。你在 Slack 里提问，直接就能得到答案。这很神奇，也非常有用。但当涉及到写代码时，你就必须回头去查看代码。

(00:42:25):

所以我觉得现在的真正瓶颈是验证代码是否有效以及进行代码审查。在我看来，如果我们想达到你那位朋友描述的境界，我们真的需要弄清楚如何让人们配置他们的编程智能体，使其在工作的后期阶段更具自主性。

(00:42:46) Lenny Rachitsky

English:

It makes sense. Like you said, writing code, I used to be an engineer, I was an engineer for 10 years, really fun to write code, really fun to just get in the flow, build architect, test. Not so fun to look at everyone else's code and just have to go through and be on the hook if it's doing something dumb that's going to take down production. And now that building has become easier, what I've always heard from companies that are really at the cutting edge of this is the bottleneck is now figuring out what to build. And then it's at the end of like, "Okay, we have all this, all 100 PRs to review. Who's going to go through all that?"

中文翻译:

有道理。就像你说的，写代码——我以前也是工程师，做了 10 年——写代码很有趣，进入心流、构建架构、测试都很有趣。但看别人的代码就没那么有趣了，而且如果代码出了什么蠢问题导致生产环境崩溃，你还得负责。现在构建变得更容易了，我从那些处于前沿的公司听到的说法是：现在的瓶颈在于弄清楚“该构建什么”。然后到了最后阶段：“OK，我们有 100 个 PR 要审查，谁来处理这些？”

(00:43:14) Alexander Embiricos

English:

Right.

中文翻译:

没错。

(00:43:15) Lenny Rachitsky

English:

Yeah.

中文翻译:

是的。

(00:43:17) Lenny Rachitsky (Sponsor: Jira Product Discovery)

English:

This episode is brought to you by Jira Product Discovery. The hardest part of building products isn't actually building products. It's everything else. It's proving that the work matters, managing stakeholders, trying to plan ahead. Most teams spend more time reacting than learning, chasing updates, justifying roadmaps, and constantly unblocking work to keep things moving. Jira Product Discovery puts you back in control. With Jira Product Discovery, you can capture insights and prioritize high impact ideas. It's flexible so it adapts to the way your team works and helps you build a roadmap that drives alignment, not questions. And because it's built on Jira, you can track ideas from strategy to delivery all in one place. Less chasing, more time to think, learn, and build the right thing. Get Jira Product Discovery for free at atlassian.com/lenny. That's atlassian.com/lenny.

中文翻译:

本集节目由 Jira Product Discovery 为您带来。构建产品最难的部分其实不是“构建”本身，而是其他所有事情：证明工作的价值、管理利益相关者、尝试提前规划。大多数团队花在应对突发状况上的时间比学习的时间还多——追逐更新、解释路线图、不断排除障碍以维持进度。Jira Product Discovery 让你重获掌控。通过它，你可以捕捉洞察并优先处理高影响力的想法。它非常灵活，能适应你团队的工作方式，并帮你构建一个能带来共识而非质疑的路线图。因为它构建在 Jira 之上，你可以从战略到交付一站式跟踪想法。减少追逐，留出更多时间思考、学习和构建正确的东西。在 atlassian.com/lenny 免费获取 Jira Product Discovery。

(00:44:08) Lenny Rachitsky

English:

What has the impact of Codex been on the way you operate as a product person as a PM? It's clear how engineering is impacted, code is written for you. What has it done to the way you operate and the way PMs operate at OpenAI?

中文翻译:

Codex 对你作为产品经理（PM）的运作方式产生了什么影响？工程端受到的影响很明显，代码有人代劳了。那它对你以及 OpenAI 的 PM 们的运作方式有什么改变？

(00:44:24) Alexander Embiricos

English:

Yeah, I mean, I think mostly I just feel much more empowered. I've always been sort of more technical leaning PM, and especially when I'm working on products for engineers, I feel like it's necessary to dogfood the product. But even beyond that, I just feel like I can do much, much more as a PM. And Scott Belsky talks about this idea of compressing the talent stack. I'm not sure if I've phrased that right. But it's basically this idea that maybe the boundaries between these roles are a little bit less needed than before because people can just do much more. And every time someone can do more, you can skip one communication boundary and make the team that much more efficient.

(00:45:03):

So I think we see it in a bunch of functions now, but I guess since you asked about products specifically, now answering questions much, much easier. You can just ask Codex for thoughts on that. A lot of PM

type work, understanding what's changing. Again, just ask Codex for help with that. Prototyping is often faster than writing specs. This is something that a lot of people have talked about.

(00:45:29):

I think something that, I don't think it's super surprising, but something that's slightly surprising is we see... We're mostly building Codex to write code that's going to be deployed to production, but actually we see a lot of throwaway code written with Codex now. It's kind of going back to this idea of ubiquitous code. So you'll see someone wants to do an analysis. If I want to understand something, it's like, okay, just give Codex a bunch of data, but then ask it to build an interactive data viewer for this data. That's just too annoying to do in the past, but now it's just totally worth the time of just getting an agent to go do something.

(00:46:02):

Similarly, I've seen some pretty cool prototypes on our design team about if you want to... Well, a designer basically wanted to build an animation, and this is the Coin Animation Codex, and it was like normally it'd be too annoying to program this animation. So they just vibe coded a animation editor and then they use the animation editor to build the animation, which they then check into their repo.

(00:46:24):

Actually, our designers, there's a ton of acceleration there. And speaking of compressing the talent stack, I think our designers are very PME. So they do a ton of product work and they actually have an entire vibe coded side prototype of the Codex app. And so a lot of how we talk about things is we'll have a really quick jam because there's 10,000 things going on, and then the designer will go think about how this should work. But instead of talking about it again, they'll just vibe code a prototype of that in their standalone prototype. We'll play with it. If we like it, they'll vibe engineer that prototype into an actual PR to land. And then depending on their comfort with the code base, like Codex utilizing Rust is a little harder, maybe they'll land it themselves or they'll get close and then an engineer can help them land the PR.

(00:47:11):

We recently shipped the Sora Android app, and that was one of the most mind-blowing examples of acceleration, actually, because usage of Codex internally at OpenAI is obviously really, really high, but it's been growing over the course of the year, both in terms of now it's basically all technical staff use it, but even the intensity and know how of how to make the most of coding agents has gone up by a ton. And so the Sora Android app, a fully new app, we built it in 18 days. It went from zero to launch to employees, and then 10 days later, so 28 days total, we went to just GA, to the public, and that was done just with the help of Codex. So pretty insane velocity.

(00:47:55):

I would say it was a little bit... I don't want to say easy mode, but there is one thing that Codex is really good at if you're a company that's building software on multiple platforms, so you've already figured out some of the underlying APIs or systems, asking Codex to port things over is really effective because it has something you can go look at. And so the engineers on that team were basically having Codex go look at the iOS app, produce plans of work that needed to be done, and then go implement those. And it was looking at iOS and Android at the same time. And so basically it was two weeks to launch to employees, four weeks total. Insanely fast.

中文翻译:

是的，我觉得最主要的是我感到自己被极大地赋能了。我一直是一个偏技术型的 PM，尤其是在为工程师开发产品时，我觉得亲自测试（dogfooding）是必须的。但除此之外，我感觉作为 PM 我能做的事情变多了。Scott

Belsky 谈到过“压缩人才栈”（compressing the talent stack）的想法，我不确定我表述得对不对，但基本上就是说，角色之间的界限可能不再像以前那么必要了，因为每个人能做的事都变多了。每当有人能多做一点，你就可以省去一个沟通环节，让团队效率更高。

(00:45:03):

我们在很多职能部门都看到了这一点。既然你问到产品，现在回答问题变得简单多了，你可以直接问 Codex 对某个问题的看法。很多 PM 类型的工作，比如理解发生了什么变化，也可以直接找 Codex 帮忙。原型设计往往比写规范更快，这是很多人都讨论过的。

(00:45:29):

还有一点虽然不算太意外，但挺有意思：我们构建 Codex 主要是为了编写生产环境的代码，但实际上我们现在看到大量用 Codex 编写的“一次性代码”。这又回到了“代码无处不在”的想法。比如有人想做分析，想理解某些东西，他会给 Codex 一堆数据，然后让它写一个交互式的数据查看器。以前这太麻烦了，但现在让智能体去做完全值得。

(00:46:02):

同样，我在设计团队也看到了一些很酷的原型。比如一位设计师想做一个动画（这是 Coin 动画 Codex），通常编写这种动画程序太烦人了。于是他们直接“凭感觉编程”（vibe coded）了一个动画编辑器，然后用这个编辑器制作动画，最后提交到仓库。

(00:46:24):

事实上，我们的设计师也得到了巨大的加速。说到压缩人才栈，我觉得我们的设计师非常有 PM 范儿。他们做大量的产品工作，甚至有一个完全靠“凭感觉编程”搞出来的 Codex App 侧边原型。我们讨论事情时通常会快速头脑风暴，因为有成千上万件事在同时进行，然后设计师会去思考如何实现。但他们不再只是口头讨论，而是直接在独立原型里把想法“写”出来。我们试玩一下，如果喜欢，他们就会把那个原型“凭感觉工程化”成一个真正的 PR。根据他们对代码库的熟悉程度（比如 Codex 使用 Rust 语言比较难），他们可能自己搞定，或者做到差不多，然后让工程师帮忙合并 PR。

(00:47:11):

我们最近发布了 Sora 的安卓应用，这实际上是最令人震撼的加速案例之一。OpenAI 内部对 Codex 的使用率极高，而且一年来一直在增长，现在基本上所有技术人员都在用，而且使用强度和技巧也提升了很多。Sora 安卓版是一个全新的 App，我们只用了 18 天就完成了从零到向员工发布的整个过程。10 天后（总计 28 天），我们就正式向公众发布（GA）了。这完全是在 Codex 的帮助下完成的。这种速度简直疯狂。

(00:47:55):

我不想说这是“简单模式”，但如果你是一家在多个平台上构建软件的公司，Codex 有一点非常擅长：如果你已经搞定了底层的 API 或系统，让 Codex 进行迁移是非常有效的，因为它有现成的东西可以参考。那个团队的工程师基本上是让 Codex 去看 iOS 版 App，生成需要完成的工作计划，然后去执行。它同时看着 iOS 和安卓的代码。所以基本上是两周向员工发布，总共四周全量发布。快得惊人。

(00:48:31) Lenny Rachitsky

English:

What makes that even more insane is it became the number one app in the app store. This just boggles the mind. Okay, so 28 days?

中文翻译:

更疯狂的是，它还成了应用商店的第一名。这简直不可思议。所以，28 天？

(00:48:39) Alexander Embiricos

English:

Yeah, so imagine number one app in the app store with a handful of engineers. I think it was two or three possibly in a handful of weeks.

中文翻译:

是的，想象一下，几个工程师，我记得可能就两三个，在几周内就做出了应用商店排名第一的 App。

(00:48:51) Lenny Rachitsky

English:

Yeah, this is absurd. Wow.

中文翻译:

这太荒谬了。哇。

(00:48:56) Alexander Embiricos

English:

Yeah, so that's a really fun example of acceleration. And then Atlas was the other one that I think Ben did a podcast, the engine lead on Atlas, sharing a little bit about how we built there. Atlas is actually... I mean, it's a browser, and building a browser is really hard. So we had to build a lot of difficult systems in order to do that. And basically we got to the point where that team has a ton of power users of Codex right now, and it got to the point they where basically... We were talking to them about it, because a lot of those engineers are people I used to work with before at my startup. And so they'd say, "Before this would've taken us two to three weeks for two to three engineers, and now it's like one engineer, one week." So massive acceleration there as well.

(00:49:49):

And what's quite cool is that we shipped Atlas on Mac first, but now we're working on the Windows version. So the team now is ramping up on Windows and they're helping us make Codex better on Windows too, which is admittedly earlier, just the model we shipped last week is the first model that natively understands PowerShell. So PowerShell being the native Shell language on Windows. So yeah, it's been really awesome to see the whole company getting accelerated by Codex from... And most obviously, also research and improving how quickly we train models and how well we do it. And then even design, as we talked about, and marketing. Actually, we're at this point now where my product marketer is often also making string changes just directly from Slack or updating docs directly from Slack.

中文翻译:

是的，那是加速的一个非常有趣的例子。Atlas 是另一个例子，我想 Atlas 的引擎主管 Ben 之前做过播客分享我们是如何构建它的。Atlas 实际上……它是一个浏览器，而构建浏览器非常难。为了实现它，我们必须构建许多复杂的系统。目前那个团队里有很多 Codex 的重度用户，情况基本上是……我们和他们聊过，因为其中很多工程师是我以前在初创公司共事过的。他们说：“以前这需要两三个工程师花两三周，现在一个工程师一周就能搞定。”那里也有巨大的加速。

(00:49:49):

很酷的一点是，我们先发布了 Mac 版 Atlas，现在正在开发 Windows 版。所以团队现在正在 Windows 上发力，他们也在帮我们改进 Codex 在 Windows 上的表现。诚然 Windows 端还比较早期，但我们上周发布的模型是第一个原生理解 PowerShell 的模型（PowerShell 是 Windows 的原生 Shell 语言）。所以，看到整个公司都被 Codex 加速真的很棒——最明显的当然是研究部门，提升了我们训练模型的速度和质量；还有我们聊过的设计部门，甚至营销部门。事实上，我们现在的产品营销人员经常直接在 Slack 里修改字符串或更新文档。

(00:50:37) Lenny Rachitsky

English:

These are amazing examples. You guys are living at the bleeding edge of what is possible, and this is how other companies are going to work. Just shipping, again, what became the number one app in the app store and just beloved all over the... It just took over, I don't know, the world for at least a week. Built, you said, in 28 days and I don't know, 10 days, 18 days just to get the core of it working.

中文翻译:

这些例子太棒了。你们生活在可能性的最前沿，这也是其他公司未来的运作方式。再次强调，发布了一个成为应用商店第一、风靡全球至少一周的 App，只用了 28 天，而核心功能只用了 18 天。

(00:51:00) Alexander Embiricos

English:

Yeah, so it was like 18 days we had a thing that employees were playing with, and then 10 days later we were out.

中文翻译:

是的，18 天时我们就有了一个员工可以试玩的东西，10 天后我们就发布了。

(00:51:05) Lenny Rachitsky

English:

And you said just a couple engineers.

中文翻译:

而且你说只有几个工程师。

(00:51:07) Alexander Embiricos

English:

Yeah.

中文翻译:

是的。

(00:51:07) Lenny Rachitsky

English:

Two or three. Okay. And then Atlas you said took a week to build?

中文翻译:

两三个。好。然后你说 Atlas 只用了一周就建成了?

(00:51:12) Alexander Embiricos

English:

No, no, no. So Atlas, not the whole week, but Atlas was a really meaty project. And so I was talking to one of the engineers on Atlas about just what they use Codex for. And it's basically like, "We use Codex for absolutely everything." And I was like, "Okay, well, how would you measure the acceleration?" And so basically the answer I got back was, "Previously would've taken two to three weeks for two to three engineers, and now it's like one engineer, one week."

中文翻译:

不不不。Atlas 整个项目不止一周，它是一个非常庞大的项目。我当时是和 Atlas 的一名工程师聊他们用 Codex 做什么，他说：“我们几乎所有事情都用 Codex。”我问：“那你怎么衡量这种加速？”他给我的回答是：“以前需要两三个工程师花两三周的工作，现在一个工程师一周就能完成。”

(00:51:36) Lenny Rachitsky

English:

Do you think this eventually moves to non-engineers doing this sort of thing? Does it have to be an engineer building this thing? Could Sora have been built by, I don't know, a PM or designer?

中文翻译:

你认为这最终会演变成非工程师也能做这类事情吗？必须由工程师来构建吗？Sora 能由 PM 或设计师构建吗?

(00:51:45) Alexander Embiricos

English:

I think we will very much get to the point, well, basically where the boundaries are a little bit blurred. I think you're going to want someone who understands the details of what they're building, but what details those are will evolve. Kind of like how now if you're writing Swift, you don't have to speak assembly. There's a handful of people in the world, and it's really important that they exist and speak assembly, maybe more than a handful, but that's a specialized function that most companies don't need to have.

(00:52:14):

So I think we're just going to naturally see an increase in layers of abstraction. And then the cool thing is now we're entering the language layer of abstraction, like natural language, and then natural language itself is really flexible. You could have engineers talking about a plan and then you could have engineers talking about a spec, and then you could have engineers talking about just a product or an idea. So I think we can also start moving up those layers of abstraction as well.

(00:52:39):

But I do think this is going to be gradual. I don't think it's going to go off to all of a sudden nobody ever writes anything, any code and it's just specs. I think it's going to be much more like, "Okay, we've set up our coding agent to be really good at previewing the build or at running tests," maybe that's the first part that most people have set up. And it's like, "Okay, now we've set it up so they can execute the build and it can see the results of its own changes, but we haven't yet built a good integration harness so that it can," in the case of Atlas... By the way, I don't know if they've done any of this or not. I think they've done a lot of this. But maybe the next stage is enable it to load a few sample pages to see how well those work. So then, okay, now we're going to set it up to do that.

(00:53:18):

And I think for some time at least, we're going to have humans curating which of these connectors or systems or components that the agent needs to be good at talking to. And then in the future, there will be an even greater unlock where Codex tells you how to set it up or maybe sets itself up in a repo.

中文翻译:

我认为我们确实会达到那个阶段，基本上界限会变得模糊。你仍然需要有人理解他们正在构建的东西的细节，但这些“细节”的定义会演变。就像现在你写 Swift 语言，不需要懂汇编语言一样。世界上只有少数人懂汇编，他们存在并掌握这项技能很重要，但这是大多数公司不需要的专门职能。

(00:52:14):

所以我认为我们会自然而然地看到抽象层级的增加。最酷的是我们现在进入了语言抽象层，即自然语言。自然语言本身非常灵活：你可以让工程师讨论计划，讨论规范，或者仅仅讨论产品或想法。所以我们也可以开始在这些抽象层级上向上移动。

(00:52:39):

但我认为这将是渐进的。我不认为会突然变成没人写代码、全是规范。它更像是：“OK，我们已经把编程智能体设置得非常擅长预览构建或运行测试了”，这可能是大多数人首先设置的部分。然后是：“OK，现在我们设置好了，让它能执行构建并查看自己更改的结果，但我们还没建立好一个好的集成环境让它能……”以 Atlas 为例（顺便说下，我不知道他们是否已经做了这些，我想他们已经做了很多），也许下一阶段是让它能加载几个示例页面看看效果。好，那我们就设置它去做这件事。

(00:53:18):

我认为至少在一段时间内，仍然需要人类来筛选智能体需要擅长沟通的连接器、系统或组件。而在未来，会有更大的突破：Codex 会告诉你如何设置它，或者它甚至能在仓库里自动完成设置。

(00:53:34) Lenny Rachitsky

English:

What a wild time to be alive. Wow. I'm curious just the second order effects of this sort of thing, just how quickly it is to build stuff. What does that do? Does that mean distribution becomes much, much more important? Does it mean ideas are just worth a lot more? It's interesting to think about how quick how that changes.

中文翻译:

活在这样一个时代真是疯狂。哇。我很好奇这种事情的二阶效应，即构建东西变得如此之快。这会带来什么？这意味着分发（distribution）变得极其重要吗？还是意味着想法变得更有价值了？思考这种变化如何发生很有趣。

(00:53:51) Alexander Embiricos

English:

I'm curious what you think. I still don't think ideas are worth as much as maybe a lot of people think. I still think execution is really hard. You can build something fast, but you still need to execute well on it, still needs to make sense and be a coherent thing overall, yeah, and distribution is massive.

中文翻译:

我也好奇你怎么想。我仍然不认为想法像很多人想象的那样值钱。我仍然认为执行非常难。你可以快速构建出东西，但你仍然需要出色地执行，它仍然需要有意义且整体连贯。是的，分发确实非常关键。

(00:54:08) Lenny Rachitsky

English:

Yeah. Just feels like everything else is now more important. Everything that isn't the building piece, which is coming up with an idea, getting to market, profit, all that kind of stuff.

中文翻译:

是的。感觉现在除了“构建”之外的一切都变得更重要了。比如构思想法、推向市场、盈利等等。

(00:54:18) Alexander Embiricos

English:

Yeah. I think we might've been in this weird temporary phase where, for a while, it was so hard to build product that you mostly just had to be really good at building product and it maybe didn't matter if you had an intimate understanding of a specific customer. But now I think we're getting to this point where actually if I could only choose one thing to understand, it would be really meaningful understanding of the problems that a certain customer has. If I could only go in with one core competency.

(00:54:52):

So I think that's ultimately still what's going to matter most. If you're starting a new company today and you have a really good understanding and network of customers that are currently underserved by AI tools, I think you're set. Whereas if you're good at building websites, but you don't have any specific customer to build for, I think you're in for a much harder time.

中文翻译:

是的。我认为我们可能曾处于一个奇怪的过渡阶段：有一段时间，构建产品太难了，以至于你只要擅长构建产品就行，是否深入了解特定客户可能没那么重要。但现在我认为我们正处于这样一个时点：如果我只能选择理解一件事，那一定是深入理解某个特定客户所面临的问题。如果我只能带一项核心竞争力入场的话。

(00:54:52):

所以我认为最终这仍然是最重要的。如果你今天创办一家新公司，并且对目前 AI 工具服务不到位的客户群体有深刻的理解和人脉，我认为你就成功了一半。而如果你只是擅长建网站，却没有特定的客户目标，我认为你的日子会难过得多。

(00:55:14) Lenny Rachitsky

English:

Bullish on vertical AI startups is what I'm hearing. Yeah, I completely agree. There's the general thing that can solve a lot of problems and then there's like, "We're going to solve presentations incredibly well and we're going to understand the presentation problem better than anyone and we're going to plug into your workflows and all these other things that matter for a very specific problem." Okay, incredible.

(00:55:36):

When you think about progress on Codex, I imagine you have a bunch of evals and there's all these public benchmarks. What's something you look at to tell you, "Okay, we're making really good progress," I imagine it's not going to be the one thing, but what do you focus on? What's something you're trying to push? What's a KPI or two?

中文翻译:

听起来你很看好垂直领域的 AI 初创公司。是的，我完全同意。既有能解决很多问题的通用工具，也有像“我们要把演示文稿做得极好，我们要比任何人都更了解演示文稿的问题，我们要接入你的工作流”这种针对特定问题的方案。太棒了。

(00:55:36):

当你思考 Codex 的进展时，我猜你们有一堆评估测试和公开基准。你通过看什么来告诉自己“OK，我们取得了很好的进展”？我猜不会是单一指标，但你关注什么？你在努力推动什么？有一两个 KPI 吗？

(00:55:51) Alexander Embiricos

English:

One of the things that I'm constantly reminding myself of is that a tool like Codex naturally is a tool that you would become a power user of. So we can accidentally spend a lot of our time thinking about features that are very deep in the user adoption journey. And so we can kind of end up oversolving for that. And so I think it's just critically important to go look at your D7 retention. Just go try the product, sign up from scratch again. I have a few too many ChatGPT Pro accounts that I've, in order to maximally correctly dogfood, signed up for on my Gmail and they charge me 200 bucks a month. I need to expense those. But I think just the feeling of being end user and the early retention stats are still super important for us because as much as this category is taking off, I think we're still in the very early days of people using them.

(00:56:44):

Another thing that we do that I think we might be the most user feedback/social media pilled team out there in this space is like a few of us are constantly on Reddit and Twitter, and there's praise up there and there's a lot of complaints, but we take the complaints very seriously and look at them. And I think that, again, because you can use a coding agent for so many different things, it often is kind of broken in many sort of ways for specific behaviors. So we actually monitor a lot just what the vibes are on social media pretty often, especially I think for Twitter/X, it's a little bit more hypey and then Reddit is a little more negative but real actually. So I've started increasingly paying attention to how people are talking about using Codex on Reddit actually.

中文翻译:

我经常提醒自己的一点是：像 Codex 这样的工具，用户自然会变成重度用户（power user）。所以我们可能会不小心花太多时间去思考那些处于用户采用路径后期的深层功能，从而导致过度设计。因此，我认为查看 D7（第七日）留存率至关重要。亲自去尝试产品，重新注册一个账号。为了最准确地进行内部测试，我注册了太多 ChatGPT Pro 账号，每个月要扣我 200 美元，我得去报销。但我认为，作为最终用户的使用感受和早期留存数据对我们来说仍然超级重要，因为尽管这个品类正在爆发，但我认为人们使用它们还处于非常早期的阶段。

(00:56:44):

我们做的另一件事是——我觉得我们可能是这个领域里最沉迷于用户反馈和社交媒体的团队——我们有几个人经常泡在 Reddit 和 Twitter 上。那里有赞扬也有很多抱怨，但我们非常严肃地对待并查看这些抱怨。因为编程智能体可以用于太多不同的事情，它在特定行为上往往会以各种方式“坏掉”。所以我们经常监测社交媒体上的“氛围”（vibes）。尤其是 Twitter/X，那里比较浮躁（hypey），而 Reddit 则稍微负面一点，但其实更真实。所以我现在越来越关注 Reddit 上人们是如何讨论使用 Codex 的。

(00:57:39) Lenny Rachitsky

English:

This is important for people to know. Which of the subreddits do you check most? Is there like an r/Codex or?

中文翻译:

这对大家很重要。你最常看哪个子版块（subreddit）？有类似 r/Codex 之类的吗?

(00:57:44) Alexander Embiricos

English:

I mean, the algorithm's pretty good at surfacing stuff, but r/Codex is there.

中文翻译:

算法在推送内容方面做得很好，不过 r/Codex 确实存在。

(00:57:48) Lenny Rachitsky

English:

Okay. I'll take. Very interesting. And then if people tag you on Twitter, you still see that, but maybe not as powerful as seeing it on Reddit.

中文翻译:

好，我记下了。很有趣。如果人们在 Twitter 上 @ 你，你仍然能看到，但可能不如在 Reddit 上看到的那么有冲击力。

(00:57:56) Alexander Embiricos

English:

Well, yeah. Well, the thing with Twitter is it's a little bit more one-to-one, even if it's in public. Whereas with Reddit, those are really good upvoting mechanics and maybe most people are still not bots, unclear. So you get good signal on what matters and what other people think.

中文翻译:

是的。Twitter 更多是一对一的，即使是在公开场合。而 Reddit 有非常好的点赞投票机制，而且可能大多数人还不是机器人（虽然不确定）。所以你能获得关于“什么最重要”以及“其他人怎么想”的良好信号。

(00:58:09) Lenny Rachitsky

English:

So interestingly, Atlas, I want to talk about that briefly. You guys launched Atlas. I tweeted actually that I tried Atlas and then I don't love the AI only search experience. I was just like, "I just want Google sometimes," or whatever. Just waiting for AI to give me an answer, I'm like, "I don't want to... " And there was no way to switch. I just tweeted, "Hey, I'm switching back. It's not great." And I feel like I made some PMs at OpenAI sad. And I saw someone tweet, "Okay, we have Atlas now," which I imagine was always part of the plan. It's probably an example of just, "We got to ship stuff, see how people use it and then we figure it out." So I guess one is that, I don't know, is there anything there? And two, I'm just curious, why are you guys building a web browser?

中文翻译:

有趣的是 Atlas，我想简单聊聊。你们发布了 Atlas。我其实发过推文说我试用了 Atlas，但我不太喜欢那种“纯 AI”的搜索体验。我当时觉得“有时我只想用 Google”。等着 AI 给我答案时，我觉得“我不想等……”，而且当时没法切换。我发推说：“嘿，我要换回去了，这不太好用。”我觉得我可能让 OpenAI 的一些 PM 伤心了。然后我看到有人发推说：“OK，我们现在有 Atlas 了（指改进后的版本）”，我猜这一直在计划中。这可能就是“先发布，看大家怎么用，然后再解决”的一个例子。所以，第一，关于这个有什么想说的吗？第二，我很好奇，你们为什么要开发网页浏览器?

(00:58:48) Alexander Embiricos

English:

So I worked on Atlas for a bit. I don't work on it now. But a bit of the narrative here for me just to tell my story a bit was I was working on this screen sharing, pair programming startup, and then we joined OpenAI. And so the idea was really to build a contextual desktop assistant. And the reason I believe that's so important is because I think that it's really annoying to have to give all your context to an assistant and then to figure out how it can help you. So if it could just understand what you are trying to do, then it could maximally accelerate you. So I still think of Codex actually as a contextual assistant from a little bit of a different angle, starting with coding tasks.

(00:59:30):

But some of the thinking, at least for me personally, I can't speak for the whole project, was that a lot of work is done in the web. And if we could build a browser, then we could be contextual for you, but in a much more first class way. We weren't hacking other desktop software which have very varied support for what content they're rendering to the accessibility tree. We wouldn't be relying on screenshots, which are a little bit slower and unreliable. Instead, we could be In the rendering engine and extract whatever we needed to help you. And also I like to think of video games, I don't know if you've played, I don't know, say Halo, you walk up to an object, I mean, this is true for many games, you press... Man, it's been a long time, this is embarrassing. Press X and it just does the right thing. And I was one of those guys who always read the instruction manual for every video game that I bought.

(01:00:24):

And I remember the first time I read about a contextual action and I just thought it was this really cool idea. And the thing about a contextual action is we need to know what you are attempting to do. We have a little bit of context and then we can help. And I think this is critically important because imagine this world that we reach where we have agents that are helping you thousands of times per day.

(01:00:49):

Imagine if the only way we could tell you that we helped you was if we could push notify you. So you get a thousand push notifications a day of an AI saying like, "Hey, I did this thing. Do you like it? " It'd be super annoying, right? Whereas imagine, going back to software engineering, I was looking at a dashboard and I noticed some key metric had gone down. And at that point in time, an AI could maybe go take a look and then surface the fact that it has an opinion on why this metric went down and maybe a fix right there right when I'm looking at the dashboard. That would much more keep me in flow and enable the agent to take action on many more things.

(01:01:26):

So in my mind, part of why I'm excited for us to have a browser is that I think we have then much more context around what we should help with. Users have much more control over what they want us to look at. It's like, "Hey, if you want us to take action on something, you can open it in your AI browser. If you don't, then then you can open it in your other browser." So really clear control and boundaries. And then we have the ability to build UX that's mixed initiatives so that we can surface contextual actions to you at the time that they're helpful as opposed to just randomly notifying you.

中文翻译:

我曾在 Atlas 团队工作过一段时间，现在不在了。对我来说，故事是这样的：我之前在做一家屏幕共享、结对编程的初创公司，后来加入了 OpenAI。当时的想法是构建一个“上下文桌面助手”。我认为这非常重要，因为把所有的背景信息（context）都喂给助手，然后再让它想办法帮你，这太烦人了。如果它能直接理解你正在尝试做什么，它就能最大程度地加速你的工作。所以我仍然认为 Codex 实际上是一个上下文助手，只是切入角度不同，它是从编程任务开始的。

(00:59:30):

但至少对我个人而言（我不能代表整个项目），部分想法是：大量的工作是在网页上完成的。如果我们能构建一个浏览器，我们就能以一种更“一等公民”的方式为你提供上下文支持。我们不需要去破解其他桌面软件（它们对渲染到辅助功能树的内容支持各异），也不需要依赖截图（截图较慢且不可靠）。相反，我们可以直接进入渲染引擎，提取任何我们需要的东西来帮助你。我也喜欢联想电子游戏，比如《光环》（Halo），你走向一个物体，按下一个键（天哪，太久没玩了，真尴尬），按下 X 键，它就会执行正确的动作。我以前是那种每买一个游戏都会读说明书的人。

(01:00:24):

我记得第一次读到“上下文动作”（contextual action）时，我觉得这个主意太酷了。上下文动作的关键在于：我们需要知道你正试图做什么，我们有一点背景信息，然后我们就能提供帮助。我认为这至关重要，因为想象一下，当我们达到智能体每天帮你数千次的世界时。

(01:00:49):

想象一下，如果告诉你的唯一方式是发推送通知。你每天收到一千条推送，AI 说：“嘿，我做了这件事，你喜欢吗？”那会烦死人的，对吧？而想象一下（回到软件工程），我正在看一个仪表盘，发现某个关键指标下降了。就在那一刻，AI 也许可以去查看一下，然后就在我盯着仪表盘时，提示它对指标下降原因的看法，甚至直接给出一个修复方案。这会让我更好地保持在心流状态，并让智能体能处理更多事情。

(01:01:26):

所以在我看来，我对我们拥有浏览器感到兴奋的部分原因是：我们拥有了更多关于“该帮什么忙”的上下文。用户对他们想让我们看什么也有了更多的控制权。就像是：“嘿，如果你想让我们对某事采取行动，就在 AI 浏览器里打开它；如果不想，就用别的浏览器。”控制权和边界非常清晰。然后我们就有能力构建“混合主动性”的 UX（用户体验），在对你有帮助的时候呈现上下文动作，而不是随机地通知你。

(01:01:58) Lenny Rachitsky

English:

Hearing the vision for Codex being the super assistant, it's not just there to code for you. It's trying to do a lot for you as a teammate, as this kind of super teammate, and that makes you awesome at work. So I get this. Speaking of that, are there other non-engineering common use cases for Codex? Just ways that non-engineers... We talked about designers prototyping and building stuff, are there any fun or unexpected ways people are using Codex that aren't engineers?

中文翻译:

听了你对 Codex 作为超级助手的愿景，它不仅仅是为你写代码。它试图作为一个队友、一个超级队友为你做很多事，让你在工作中表现卓越。我明白了。说到这，Codex 有哪些非工程类的常见用例吗？非工程师的使用方式……我们聊过设计师做原型和构建东西，还有什么有趣或意想不到的非工程师使用方式吗？

(01:02:24) Alexander Embiricos

English:

I mean, there's a load of unexpected ways, but I think most of where we're seeing real traction with people using things are still for now very, I would say, coding adjacent or tech-oriented, places where there's a mature ecosystem or maybe you're doing data analysis or something like that. I personally am expecting that we're going to see a lot more of that over time. But for now, we're keeping the team very focused on just coding for now because there's so much more work to do.

中文翻译:

意想不到的方式有很多，但我认为目前看到真正有吸引力的用法仍然非常……我会说是“编程相邻”或“技术导向”的。比如在生态系统成熟的领域，或者你在做数据分析之类的事情。我个人预计随着时间的推移，我们会看到更多这类用法。但目前，我们让团队非常专注于编程本身，因为还有太多的工作要做。

(01:02:54) Lenny Rachitsky

English:

For people that are thinking about trying out Codex, does it work for all kinds of code bases? What code does it support? If you're like, I don't know, SAP, can you add Codex and start building things? What's the sweet spot? Where does it start to not be amazing yet?

中文翻译:

对于那些想尝试 Codex 的人，它适用于各种代码库吗？它支持什么代码？如果你在用，比如 SAP，你能加入 Codex 并开始构建吗？它的“甜点位”（最擅长的领域）在哪？哪里是它目前表现还没那么惊艳的地方？

(01:03:11) Alexander Embiricos

English:

I'm really glad you asked this question actually because the best way to try Codex is to give it your hardest tasks, which is a little different than some of the other coding agents. Some tools you might think, "Okay, let me start easy or just vibe code something random and decide if I like the tool." Whereas we're really building Codex to be the professional tool that you can give your hardest problems to. And that writes high quality code in your enormous code base that is in fact not perfect right now. So yeah, I think if you're going to try Codex, you want to try it on a real task that you have and not necessarily dumb that task down to something that's trivial, but actually a good one would be you have a hard bug and you

don't know what's causing that bug and you ask Codex to help figure that out or to implement that the fix.

中文翻译:

我很高兴你问这个问题，因为尝试 Codex 的最佳方式是给它最难的任务，这与其他一些编程智能体有点不同。有些工具你可能会想：“好吧，让我先从简单的开始，或者随便‘凭感觉’写点东西，看看喜不喜欢这个工具。”而我们构建 Codex 的目标是让它成为一个专业工具，你可以把最难的问题交给它。它能在你庞大且目前并不完美的代码库中编写高质量代码。所以，如果你要尝试 Codex，你应该在真实的、具有挑战性的任务上尝试，而不是把它降级为琐碎的小事。一个很好的例子是：你有一个很难缠的 bug，不知道原因，你让 Codex 帮你想清楚或者实现修复。

(01:04:00) Lenny Rachitsky

English:

I love that answer. Just give it to your hardest problem.

中文翻译:

我喜欢这个回答。直接把最难的问题丢给它。

(01:04:03) Alexander Embiricos

English:

I will say if you're like, "Hey, okay, well, the hardest problem I have is that I need to build a new unicorn business," obviously that's not going to work. Not yet. So I think it's like give it the hardest problem, but something that is still one question or one task to start. That's if you're testing and then over time you can learn how to use it for bigger things.

中文翻译:

但我得说，如果你说：“嘿，我最难的问题是我需要建立一个新的独角兽企业”，那显然行不通，至少现在还不行。所以，给它最难的问题，但最好是从一个具体的问题或任务开始。这是针对测试阶段的，随着时间的推移，你可以学会如何用它处理更大的事情。

(01:04:25) Lenny Rachitsky

English:

Yeah. What languages does it support?

中文翻译:

明白。它支持哪些语言？

(01:04:27) Alexander Embiricos

English:

Basically, the way we've trained Codex is there's a distribution of languages that we support and it's fairly aligned with the frequency of these languages in the world. So unless you're writing some very esoteric language or some private language, it should do fine in your language.

中文翻译:

基本上，我们训练 Codex 的方式是支持一系列语言分布，这与这些语言在世界上的使用频率相当一致。所以除非你在写一些非常冷门的语言或某种私有语言，否则它在你的语言上应该表现良好。

(01:04:41) Lenny Rachitsky

English:

If someone was just getting started, is there a tip you could share to help them be successful? If you could just whisper a little tip into someone just setting up Codex for the first time to help them have a really good time, what's something you would whisper?

中文翻译:

如果有人刚开始使用，你能分享一个帮助他们成功的秘诀吗？如果你能对第一次安装 Codex 的人悄悄说一个小建议，让他们有很好的体验，你会说什么？

(01:04:54) Alexander Embiricos

English:

I might say try a few things in parallel. So you could try giving it a hard task, maybe ask it to understand the code base, formulate a plan with it around an idea that you have and kind of build your way up from there. And the meta idea here is, again, it's like you're building trust with a new teammate. And so you wouldn't go to a new teammate and just give them like, "Hey, do this thing. Here's zero context." You would start by first making sure they understand the code base and then you would maybe align on an approach and then you would have them go off and do bit by bit. And I think if you use Codex in that way, you'll just naturally start to understand the different ways of prompting it because it's a super powerful agent and model, but it is a little bit different to prompt Codex than other models.

中文翻译:

我会说，尝试并行做几件事。你可以试着给它一个艰巨的任务，比如让它理解代码库，围绕你的想法和它一起制定一个计划，然后从那里开始逐步深入。这里的核心理念还是：你正在与一个新队友建立信任。你不会直接找个新队友说：“嘿，做这件事，背景信息为零。”你会先确保他们理解代码库，然后就方法达成一致，再让他们一点点去做。如果你以这种方式使用 Codex，你就会自然而然地开始理解提示它的不同方式，因为它是一个超级强大的智能体和模型，但提示 Codex 的方式与其他模型确实略有不同。

(01:05:38) Lenny Rachitsky

English:

Just a couple more questions. One, we touched on this a little bit, as AI does more and more coding, there's always this question of, "Should I learn to code and why should I spend time doing this sort of thing?" For people that are trying to figure out what to do with their career, especially if they're into software engineering computer science, do you think there's specific elements of computer science that are more and more important to lean into, maybe things they don't need to worry about? What do you think people should be leaning into skill-wise as this becomes more and more of a thing in our workplace?

中文翻译:

最后还有几个问题。第一，我们稍微提到过，随着 AI 编写的代码越来越多，总会有这样一个问题：“我还需要学习编程吗？为什么还要花时间做这种事？”对于那些正在规划职业生涯的人，尤其是对软件工程和计算机科学感兴趣的人，你认为计算机科学中哪些特定元素正变得越来越重要，哪些可能不再需要担心？随着 AI 在职场中变得越来越普遍，你认为人们在技能方面应该向什么方向倾斜？

(01:06:11) Alexander Embiricos

English:

I think there's a couple angles you could go at this from. Well, the easiest one to think of at least is just be a doer of things. I think that with coding agents getting better and better over time, it's just what you can do as even someone in college or a new grad is just so much more than what that was before. And so I think you just want to be taking advantage of that. And definitely when I'm looking at hiring folks who are earlier career, it's definitely something that I think about is how productive are they using the latest tools? They should be super productive. And if you think of it in that way, they actually have less of a handicap than before versus a more senior career person because the divide is actually getting smaller because they've got these amazing coding agents now. So that's one thing, which is, I guess the advice is just learn about whatever you want, but just make sure you spend time doing things, not just fulfilling homework assignments, I guess.

(01:07:11):

I think the other side of it though is that it's still deeply worth understanding what makes a good overall software system. So I still think that skills, like really strong systems engineering skills, or even really effective communication and collaboration with your team, skills like that I think are important or are going to continue to matter for quite some time. I don't think it's going to be all of a sudden the AI coding agents are just able to build perfect systems without your help. I think it's going to look much more gradual where it's like, okay, we have these AI coding agents, they're able to validate their work. It's still important.

(01:07:51):

For example, I'm thinking of an engineer who was working on Atlas, since we were talking about it, he set up Codex so that it can verify its own work, which is a little bit non-trivial because of the nature of the Atlas project. So the way that he did that was he actually prompted Codex like, "Hey, why can't you verify your work? Fix it," and did that on a loop. And so you still, at various phases, are going to want a human in the loop to help configure the coding agent to be effective. So I think you still want to be able to reason about that. So maybe it's less important that you can type really fast and you understand exactly how to write... Not that anyone writes a 4H loop or something, or you don't need to know how to implement a specific algorithm. But I think you need to be able to reason about the different systems and what makes a software engineering team effective. So I think that's the other really important thing.

(01:08:40):

Then maybe the last angle that you could take is, I think if you're on the frontier of knowledge for a given thing, I still think that's deeply interesting to go down, partially because that knowledge is still going to be... Agents aren't going to be as good at that, but also partially because I think that by trying to advance the frontier of a specific thing, you'll actually end up being forced to take advantage of coding agents and using them to accelerate your own workflow as you go.

中文翻译:

我认为可以从几个角度来看。首先，最简单的一点就是做一个“行动者”（doer）。随着编程智能体变得越来越好，即使是大学生或应届生，能做的事情也比以前多得多。所以你应该充分利用这一点。当我招聘职场新人

时，我肯定会考虑：他们使用最新工具的效率有多高？他们应该是非常高效的。从这个角度看，与资深人士相比，新人的劣势其实变小了，因为有了这些神奇的编程智能体，差距正在缩小。所以建议是：学你想学的任何东西，但确保花时间去“做”事情，而不仅仅是完成作业。

(01:07:11):

另一方面，深入理解“什么是好的整体软件系统”仍然非常有价值。我仍然认为，像强大的系统工程能力，甚至与团队进行有效的沟通和协作，这些技能在相当长一段时间内都将至关重要。我不认为 AI 编程智能体会突然间就能在没有人类帮助的情况下构建出完美的系统。这会是一个渐进的过程：我们有了 AI 编程智能体，它们能验证自己的工作，但这仍然很重要。

(01:07:51):

举个例子，我想起 Atlas 项目的一名工程师，他设置了 Codex 让它能验证自己的工作。由于 Atlas 项目的性质，这并不容易。他的做法是提示 Codex：“嘿，你为什么不能验证自己的工作？修复它”，然后循环执行。所以，在各个阶段，你仍然需要“人在回路”（human in the loop）来帮助配置编程智能体使其高效。你仍然需要具备推理能力。也许打字快不快、是否精通某种循环写法不再那么重要，你也不需要知道如何实现某种特定算法。但你需要能够对不同系统进行推理，并理解是什么让一个软件工程团队变得高效。这是另一个重点。

(01:08:40):

最后一个角度是：如果你处于某个领域的知识前沿，我认为深入钻研仍然非常有意义。部分原因是智能体在这些前沿知识上还没那么强，另一部分原因是，在尝试推进特定领域前沿的过程中，你实际上会被迫利用编程智能体来加速你自己的工作流。

(01:09:09) Lenny Rachitsky

English:

What's an example that when you talk about being at the frontier of something?

中文翻译:

你能举个关于“处于某事的前沿”的例子吗？

(01:09:12) Alexander Embiricos

English:

Codex writes a lot of the code that helps manage its training runs, the key infrastructure. We move pretty fast and so we have a Codex code review is catching a lot of mistakes. It's actually cause some pretty interesting configuration mistakes. And we're starting to see glimpses of the future where we're actually starting to have Codex even be on call for its own training, which is pretty interesting. So there's lots there.

中文翻译:

Codex 编写了大量帮助管理其训练运行的代码，也就是关键的基础设施。我们动作很快，所以我们用 Codex 进行代码审查，它发现了很多错误，甚至是一些非常有趣的配置失误。我们开始看到未来的雏形：Codex 甚至开始为它自己的训练过程“值班”，这非常有趣。这方面有很多可以聊的。

(01:09:38) Lenny Rachitsky

English:

Wait, what does that mean to be on call for its own training? So it's running, it's training and it's like, "Oh, something broke, someone needs..." And does it alert people or it's like, "Here, I'm going to fix the problem and restart"?

中文翻译:

等等，“为它自己的训练值班”是什么意思？是说它在运行、在训练，然后它发现“噢，出故障了，需要有人……”，它是提醒人类，还是说“我来修复问题并重启”？

(01:09:47) Alexander Embiricos

English:

This is an early idea that we're figuring out. But the basic idea is that during a training run, there's a bunch of graphs that today humans are looking at and it's really important to look at those. We call this babysitting.

中文翻译:

这是一个我们正在探索的早期想法。基本思路是：在训练运行期间，有很多图表目前是需要人类去盯着看的，而且盯着看非常重要。我们管这叫“看孩子”（babysitting）。

(01:09:59) Lenny Rachitsky

English:

Because it's very expensive to train, I imagine, and very important to move fast and-

中文翻译:

因为训练非常昂贵，我猜，而且快速推进非常重要……

(01:10:03) Alexander Embiricos

English:

Exactly. And there's a lot of systems underlying the training run. And so a system could go down or there could be an error somewhere that gets introduced. And so we might need to fix it or pause things or, I don't know, there's lots of actions we might need to take. And so basically having Codex run on a loop to evaluate how those charts are moving over time is this idea that we have to how to enable us to train way more efficiently.

中文翻译:

没错。训练运行背后有很多底层系统。某个系统可能会宕机，或者某个地方引入了错误。我们可能需要修复它、暂停它，或者采取各种行动。所以，让 Codex 循环运行来评估这些图表随时间的变化趋势，就是为了让我们能更高效地进行训练。

(01:10:26) Lenny Rachitsky

English:

I love that. And this is very much along the lines of this is the future of agents. Codex isn't just for building code, it's a lot more than that.

中文翻译:

我喜欢这个。这非常符合“智能体的未来”这一思路。Codex 不仅仅是为了写代码，它的用途远不止于此。

(01:10:34) Alexander Embiricos

English:

Yeah.

中文翻译:

是的。

(01:10:36) Lenny Rachitsky

English:

Okay, last question. Being at OpenAI, I can't not ask about your AGI timeline and how far you think we are from AGI. I know this isn't what you work on, but there's a lot of opinions, a lot of, I don't know, timelines. How far do you think we are from a humanly human version of AI, whatever that means to you?

中文翻译:

好，最后一个问题。既然在 OpenAI，我不能不问你的 AGI 时间线，以及你认为我们离 AGI 还有多远。我知道这不是你的研究领域，但有很多观点和时间线。你认为我们离一个“像人一样的人工智能”（无论你怎么定义它）还有多远？

(01:10:56) Alexander Embiricos

English:

For me, I think that it's a little bit about when do we see the acceleration curves go like this? Or I don't know which way I'm mirrored here. When do we see the hockey stick? And I think that the current limiting factor, I mean, there's many, but I think a current underappreciated limiting factor is literally human typing speed or human multitasking speed on writing prompts. And like you were talking about, it's like you can have an agent watch all the work you're doing, but if you don't have the agent also validating its work, then you're still bottlenecked on can you go review all that code?

(01:11:29):

So my view is that we need to unblock those productivity loops from humans having to prompt and humans having to manually validate all the work. So if we can rebuild systems to let the agent be default useful, we'll start unlocking hockey sticks. Unfortunately, I don't think that's going to be binary. I think it's going to be very dependent on what you're building. So I would imagine that next year, if you're a startup and you're building new pieces, like some new app or something, it'll be possible for you to set it up on a stack where agents are much more self-sufficient than not. But now let's say, I don't know, you mentioned SAP, let's say you work in SAP, they have many complex systems and they're not going to be able to just get the agent to be self-sufficient overnight in those systems. So they're going to have to slowly maybe replace systems or update systems to allow the agent to handle more of the work end to end.

(01:12:22):

So basically my long answer to your question, maybe boring answer is that I think starting next year, we're going to see early adopters starting to hockey stick their productivity. And then over the years that follow, we're going to see larger and larger companies like hockey stick that productivity. And then somewhere in that fuzzy middle is when that hockey sticking will be flowing back into the AI labs and that's when we'll basically be at the AGI tier.

中文翻译:

对我来说，这取决于我们何时看到加速曲线呈现出……我不知道镜头里我是怎么镜像的……何时看到“曲棍球棒”式的增长。我认为目前的限制因素有很多，但一个被低估的因素确实是人类的打字速度，或者人类编写提示词时的多任务处理速度。就像你说的，你可以让智能体观察你做的所有工作，但如果智能体不能同时验证自己的工作，你仍然会卡在“你能不能去审查所有那些代码”这个瓶颈上。

(01:11:29):

所以我的观点是，我们需要把人类必须提示和人类必须手动验证工作的环节从生产力循环中解脱出来。如果我们能重建系统，让智能体默认就是有用的，我们就会开始解锁那种爆发式增长。不幸的是，我不认为这是非黑即白的。我认为这很大程度上取决于你在构建什么。我设想明年，如果你是一家初创公司，正在构建新东西（比如新 App），你可能可以在一个智能体高度自给自足的技术栈上运行。但如果你在 SAP 这样拥有许多复杂系统的公司工作，智能体不可能一夜之间在这些系统中实现自给自足。他们必须慢慢替换或更新系统，让智能体能处理更多端到端的工作。

(01:12:22):

所以，对于你的问题，我的长篇回答（也许有点无聊）是：我认为从明年开始，我们会看到早期采用者的生产力开始爆发式增长。在接下来的几年里，我们会看到越来越大的公司实现这种增长。而在那个模糊的中间地带，当这种生产力爆发回流到 AI 实验室时，我们就基本上达到了 AGI 的水平。

(01:12:48) Lenny Rachitsky

English:

I love this answer. It's very practical and it's something that comes up a lot on this podcast. Just like the time to reveal all the things AI is doing is really annoying and a big bottleneck. I love that you're working on this because it's one thing to just make coding much more efficient and do that for people. It's another to take care of that final step of, "Okay, is this actually great?" And that's so interesting that your sense is that's the limiting factor. It comes back to your earlier point of even if AI did not advance anymore, we have so much more potential to unlock as we learn to use it more effectively. So that is a really unique answer. I haven't heard that perspective on what is the big unlock. Human typing speed to review basically what AI is doing for us. So good.

中文翻译:

我喜欢这个回答。它非常务实，也是本播客经常出现的话题。比如查看 AI 所做的一切事情所花的时间非常烦人，是一个巨大的瓶颈。我很高兴你在致力于解决这个问题，因为让编程更高效是一回事，而处理最后一步“OK，这真的没问题吗？”则是另一回事。你认为这是限制因素，这很有趣。这回到了你之前的观点：即使 AI 不再进步，随着我们学会更有效地使用它，我们仍有巨大的潜力可以挖掘。这是一个非常独特的回答，我还没听过关于“重大突破口”的这种视角：人类审查 AI 工作成果时的打字速度。太棒了。

(01:13:31) Alexander Embiricos

English:

Okay, Alexander, we covered a lot of ground. Is there anything that we haven't covered? Is there anything you wanted to share, maybe double down on before we get to our very exciting lightning round?

中文翻译:

好，Alexander，我们聊了很多。还有什么没聊到的吗？在进入精彩的闪电轮环节之前，你有什么想分享或强调的吗？

(01:13:43) Alexander Embiricos

English:

I think one thing is that the Codex team is growing. And as I was just saying, we're still somewhat limited by human thinking speed and human typing speed. We're working on it. So if you're an engineer or a salesperson, or I'm hiring a product person, please hit us up. I'm not sure the best way to give contact info, but I guess you can go to our jobs page, or do they have contact for you actually? Do listeners have contact for you?

中文翻译:

我想说的一点是 Codex 团队正在壮大。正如我刚才所说，我们仍然在某种程度上受限于人类的思考速度和打字速度。我们正在努力解决。所以如果你是工程师、销售人员，或者我正在招聘的产品人员，请联系我们。我不确定提供联系方式的最佳方式，我想你可以去我们的招聘页面，或者听众有你的联系方式吗？

(01:14:10) Lenny Rachitsky

English:

Where they send me like, "Hey, I want to apply to Codex"? I do have a contact form at lennyrachitsky.com. I'm afraid of all the amazing people that are going to ping me. But there we go, we could try that. Let's see how that goes.

中文翻译:

让他们发给我说“嘿，我想申请 Codex”？我在 lennyrachitsky.com 确实有一个联系表单。我有点担心会有太多优秀的人来找我，但我们可以试试，看看效果如何。

(01:14:20) Alexander Embiricos

English:

Okay. Or maybe an easier version, we can edit all that out or up to you. Yeah, or I would just say you can drop us a DM. For example, I'm Embirico on Twitter, and hit me up if you're interested in joining the team.

中文翻译:

好。或者更简单点，你可以直接给我们发私信。比如我的 Twitter 账号是 @Embirico，如果你有兴趣加入团队，请联系我。

(01:14:33) Lenny Rachitsky

English:

What a dream job for so many people. What's a sign they... I don't know, what's a way to filter people a little bit so they're not flooding your inbox?

中文翻译:

对很多人来说这真是梦寐以求的工作。有什么标准可以……我不知道，有什么方法可以筛选一下，免得你的收件箱被淹没？

(01:14:42) Alexander Embiricos

English:

So specifically if you want to join the Codex team, then you need to be a technical person who uses these tools. And I think I would just ask yourself the question, "Hey, let's say I were to join OpenAI and work on Codex over the next six months and crush it, what does the life of a software engineer look like then?" And I think if you have an opinion on that, you should apply. And if you don't have an opinion on that and have to think about it first, depending on how long you have to think about it, I guess that would be the filter. I think there's a lot of people thinking about this space. So we're very interested in folks who have already been thinking about what the future should look like with agents. And we don't have to agree on where we're going, but I think we want people who are very passionate about the topic, I guess.

中文翻译:

具体来说，如果你想加入 Codex 团队，你必须是一个使用这些工具的技术人员。我想你可以问自己一个问题：“嘿，假设我加入 OpenAI 并在接下来的六个月里负责 Codex 且大获成功，那么到那时软件工程师的生活会是什么样子的？”如果你对此有自己的见解，你就应该申请。如果你没有见解，还得先想想（取决于你想多久），那这就是筛选标准。我认为有很多人在思考这个领域，所以我们对那些已经在思考“智能体时代的未来应该是怎样”的人非常感兴趣。我们不一定要在发展方向上达成一致，但我们需要对这个话题充满热情的人。

(01:15:28) Lenny Rachitsky

English:

It's very rare to be working on a product that has this much impact and is at such a bleeding edge of where it's possible. What a cool role for the right person. So it's awesome that you have an opening and this audience is a really good fit potentially for that role. So I hope we find someone, that would be incredible. With that, we've reached our very exciting lightning round. I've got five questions for you, Alexander. Are you ready?

中文翻译:

能在一个影响力如此巨大且处于可能性最前沿的产品上工作，是非常罕见的。对于合适的人来说，这是一个非常酷的角色。很高兴你们有职位空缺，而本播客的听众中可能有非常合适的人选。希望我们能找到那个人，那太棒了。接下来，我们进入精彩的闪电轮环节。我有五个问题要问你，Alexander。准备好了吗？

(01:15:54) Alexander Embiricos

English:

I don't know what these are, but I'm excited. Let's do it.

中文翻译:

我不知道是什么问题，但我很兴奋。开始吧。

(01:15:57) Lenny Rachitsky

English:

They're the same questions I ask everyone except for the last one. So probably not a surprise. I should probably make them more often a surprise. Okay, first question, what are a couple of books that you recommend most to other people, two or three books that come to mind?

中文翻译:

除了最后一个，其他都是我问每个人的老问题，所以可能没啥惊喜。我以后应该多准备点惊喜。好，第一个问题：你最推荐给别人的书有哪些？脑海中浮现的两三本。

(01:16:12) Alexander Embiricos

English:

I have been reading a lot of science fiction recently, and I'm sure this has been recommended before, but The Culture, I think it's Ian Banks is the name of the author. Part of why I love it is because it's basically relatively recent writing about a future with AI, but it's an optimistic future with AI. And I think a lot of sci-fi is fairly dystopian. But the joke, at least on The Culture subreddit is that, let me see if I can get this right, it is a space communist utopia, or I think it's a gay space communist utopia. And I just think it's really fun to think about, to use The Culture as a way to think about what kind of world can we usher in and what decisions can we make today to help usher in that world.

中文翻译:

我最近读了很多科幻小说。我确信这套书以前肯定有人推荐过，就是《文明》（The Culture）系列，作者是伊恩·班克斯（Iain M. Banks）。我喜欢它的部分原因是，它是相对较近期的关于 AI 未来的描写，而且是一个乐观的未来。我认为很多科幻小说都相当反乌托邦。但在《文明》的 Reddit 版块上有一个笑话（看我能不能说对）：它是一个“太空共产主义乌托邦”，或者说是“同性恋太空共产主义乌托邦”。我觉得思考《文明》系列很有趣，可以借此思考我们可以开启什么样的世界，以及我们今天可以做出哪些决定来帮助开启那个世界。

(01:17:02) Lenny Rachitsky

English:

Wow. I don't think anyone's recommended that. I know you're reading, you've mentioned before I start recording, Lord of the Rings right now. If you want another AI-ish sci-fi book, have you read Fire Upon the Deep?

中文翻译:

哇，我想还没人推荐过这个。我知道你正在读《指环王》，录音前你提过。如果你想要另一本关于 AI 的科幻书，你读过《深渊上的火》（Fire Upon the Deep）吗？

(01:17:15) Alexander Embiricos

English:

No, I haven't.

中文翻译:

没读过。

(01:17:15) Lenny Rachitsky

English:

Okay, it's incredibly good. It's like a sci-fi space opera sort of epic tale with super intelligence.

中文翻译:

好，那本书非常棒。它像是一部关于超级智能的科幻太空歌剧史诗。

(01:17:25) Alexander Embiricos

English:

Cool.

中文翻译:

酷。

(01:17:25) Lenny Rachitsky

English:

Yeah. Mostly not optimistic, but somewhat optimistic.

(01:17:30):

Okay, next question. Is there a favorite recent movie or TV show that you've really enjoyed?

中文翻译:

是的。大部分不乐观，但也有点乐观成分。

(01:17:30):

好，下一个问题。最近有什么非常喜欢的电影或电视剧吗?

(01:17:36) Alexander Embiricos

English:

Yeah, there's an anime called Jujutsu Kaisen, which I really like. Again, it's got a slightly dark topic of demons. But what I love about it is that the hero is really nice. And I think there's this new wave of anime and cartoons where the protagonists are really friendly and people who care about the world rather than being sort of, if you look at some older anime that started the genre, there's Evangelion or Akita and those characters, the protagonists are deeply flawed, quite unhappy. They didn't start the genre, but it was a trend for a while to poke fun at the idea that in these cartoons the protagonist was very young, but being given a ridiculous amount of responsibility to save the world. So there was kind of a wave of content that was critiquing this by making the character basically go through serious mental issues in the middle of the show. And I'm not saying this is better, but at least it's quite fun to have these really positive protagonists are just trying to help everyone around them.

中文翻译:

有一部叫《咒术回战》（Jujutsu Kaisen）的动漫，我很喜欢。虽然它的题材关于恶魔，有点阴暗，但我喜欢它是因为主角人真的很好。我认为现在有一股新浪潮，动漫里的主角变得非常友好，是关心世界的人。而如果你看一些早期的经典动漫，比如《新世纪福音战士》（Evangelion）或《阿基拉》（Akira），那些主角都有深刻的缺陷，非常不快乐。那曾是一种趋势，讽刺那些让年幼的主角承担拯救世界重任的设定，所以有一波作品通过让角色在剧中心理崩溃来批判这一点。我不是说现在的更好，但看到这些积极向上、只想帮助身边人的主角，至少感觉挺有趣的。

(01:18:43) Lenny Rachitsky

English:

I love how much we're learning about your personality hearing these recommendations. Nice protagonists, optimistic futures. I like the [inaudible 01:18:53].

中文翻译:

我喜欢通过这些推荐来了解你的个性。善良的主角，乐观的未来。我喜欢这种风格。

(01:18:53) Alexander Embiricos

English:

I think if you don't believe it, you can't will it into existence. So you need a balance.

中文翻译:

我认为如果你不相信它，你就无法让它成为现实。所以需要一种平衡。

(01:18:57) Lenny Rachitsky

English:

This is your training data.

(01:18:59):

Is there a product you recently discovered you really love? Could be an app, could be some clothing, could be some kitchen gadget, tech gadget, a hat.

中文翻译:

这就是你的“训练数据”。

(01:18:59):

最近有没有发现什么特别喜欢的产品？可以是 App、衣服、厨房用具、科技产品或者一顶帽子。

(01:19:09) Alexander Embiricos

English:

Yeah, so I have been quite into combustion engines and cars. Actually, the reason I came to America initially was because I wanted to work on US aircraft, but now I work in software. And so for the longest time, I've basically only had quite old sports cars, old just because they were more affordable. And then recently we got a Tesla instead. And I have to say that I find the Tesla software quite inspiring. In

particular, it has the self-driving feature. And I've mentioned a few times today, I think it's really interesting to think about how to build mixed initiative software that makes you feel maximally empowered as a human, maximally in control, but yet you're getting a lot of help. And I think they did a really good job with enabling the car to drive itself, but all these different ways that you can adjust what it's doing without turning off the self-driving. So you can accelerate, it'll listen to that. You can turn a knob to change its speed. You can steer slightly. I think it's actually a masterclass in building an agent that still leaves the human in control.

中文翻译:

是的，我最近对内燃机和汽车很感兴趣。其实我最初来美国是因为想从事美国飞机的研究，但现在我在做软件。很长一段时间里，我只开得起比较旧的跑车。最近我们换了一辆特斯拉。我必须说，特斯拉的软件非常具有启发性。特别是它的自动驾驶功能。我今天提到过好几次，思考如何构建“混合主动性”软件非常有趣——它让你作为人类感到被最大程度地赋能，拥有最大程度的控制权，同时又得到了大量的帮助。我认为他们在让汽车自动驾驶方面做得很好，你可以通过各种方式调整它的行为而无需关闭自动驾驶：你可以加速，它会听从；你可以转动旋钮改变速度；你可以轻微转向。我认为这在构建“让人类保持控制的智能体”方面堪称大师级作品。

(01:20:21) Lenny Rachitsky

English:

This reminds me Nick Turley's whole mantra is, "Are we maximally accelerated?"

中文翻译:

这让我想起 Nick Turley 的口头禅：“我们被最大程度地加速了吗？”

(01:20:26) Alexander Embiricos

English:

Yeah.

中文翻译:

是的。

(01:20:26) Lenny Rachitsky

English:

Feels like it's completely infiltrated everything at OpenAI, which makes sense, that tracks.

(01:20:32):

Two more questions. Do you have a life motto that you often think about and come back to in work or in life that's been helpful?

中文翻译:

感觉这种思想已经完全渗透进 OpenAI 的方方面面了，这很合理。

(01:20:32):

最后两个问题。你有没有什么人生格言，是你在工作或生活中经常想起并觉得很有帮助的？

(01:20:39) Alexander Embiricos

English:

I don't know if I have a life motto, but maybe I can tell you about the number one company value from my startup.

中文翻译:

我不知道我有没有人生格言，但我可以告诉你我那家初创公司的头号价值观。

(01:20:45) Lenny Rachitsky

English:

Love it.

中文翻译:

太好了。

(01:20:46) Alexander Embiricos

English:

Which is still something that sticks with me, which is to be kind and candid.

中文翻译:

它至今仍影响着我，那就是：友善且坦诚（Kind and Candid）。

(01:20:51) Lenny Rachitsky

English:

That tracks. Kind and candid. Wow, that's a great combo.

中文翻译:

这很符合你的风格。友善且坦诚，哇，真是个完美的组合。

(01:20:54) Alexander Embiricos

English:

Yeah. And we had to put them together because we, as founders, realized that we often would be nice and it wasn't actually the right thing to do. We would delay the difficult conversations and we were not candid. And so every time we would remind ourselves of this motto and then we would become more candid. And then six months later, we would realize that we were in fact not candid six months ago and we needed to be even more candid. So then the question is like, "Okay, how should we be candid?" It's like, "Okay, well, let's think of being candid as an act of kindness," but also think of that both in terms of doing it and willing ourselves to do it, but also in terms of how we frame it as people.

中文翻译:

是的。我们必须把它们放在一起，因为作为创始人，我们意识到我们经常为了表现得“友善”而没去做正确的事。我们会推迟艰难的对话，不够坦诚。所以每次我们都会用这个格言提醒自己，然后变得更坦诚。但六个月后，我们会发现其实六个月前的自己还是不够坦诚，我们需要更进一步。所以问题就变成了：“OK，我们该如何做到坦诚？”答案是：“好吧，让我们把坦诚看作是一种友善的行为”，不仅要强迫自己去做，还要思考作为一个人该如何去表达它。

(01:21:32) Lenny Rachitsky

English:

That is a beautiful way of summarizing how to lead well. What's the book about challenge directly but care deeply? Radical Candor.

中文翻译:

这是对如何领导的一个非常优美的总结。那本关于“直接挑战但深切关怀”的书叫什么来着？《彻底坦诚》（Radical Candor）。

(01:21:43) Alexander Embiricos

English:

Yeah, yeah.

中文翻译:

对，没错。

(01:21:44) Lenny Rachitsky

English:

So it's like another way of thinking about Radical Candor.

(01:21:46):

Okay, last question. I was looking up your last name just like, "Hey, what's the story here?" So your last name is Embiricos, and I was talking at ChatGPT and it told me the most famous individuals with the surname are the influential Greek poet and psychoanalyst Andreas Embiricos and his relative, the wealthy shipping magnate and art collector, George Embiricos. So the question is, which of these two do you most identify with, the Greek poet and psychoanalyst or the wealthy shipping magnate and art collector?

中文翻译:

所以这是对“彻底坦诚”的另一种思考方式。

(01:21:46):

好，最后一个问题。我查了一下你的姓氏，想看看有什么背景。你的姓是 Embiricos，我问了 ChatGPT，它告诉我这个姓氏最著名的人物是极具影响力的希腊诗人兼精神分析学家 Andreas Embiricos，以及他的亲戚——富有的航运大亨兼艺术收藏家 George Embiricos。所以问题是：你觉得自己更像哪一个？希腊诗人兼精神分析学家，还是富有的航运大亨兼艺术收藏家？

(01:22:19) Alexander Embiricos

English:

I think it's going to have to be the poet because he loved the island that our family's from.

中文翻译:

我想应该是那位诗人，因为他深爱着我们家族起源的那个岛屿。

(01:22:27) Lenny Rachitsky

English:

Wait, you know those people? Okay, this is not news to you. Okay.

中文翻译:

等等，你认识这些人？好吧，这对你来说不是新闻。

(01:22:30) Alexander Embiricos

English:

Well, I mean, it's an enormous family. But it's like Greek, so these big families, everyone's your uncle.

中文翻译:

我是说，这是一个庞大的家族。希腊人嘛，这种大家族里每个人都是你的叔叔伯伯。

(01:22:30) Lenny Rachitsky

English:

Love this. Okay.

中文翻译:

太棒了。

(01:22:36) Alexander Embiricos

English:

You know what I mean? My mother's Malaysian and also everyone is my uncle or aunt in Malaysia too, if that makes sense.

中文翻译:

你懂我意思吧？我母亲是马来西亚人，在马来西亚那边，每个人也都是我的叔叔阿姨。

(01:22:42) Lenny Rachitsky

English:

Yeah.

中文翻译:

明白。

(01:22:43) Alexander Embiricos

English:

But yeah, he loved this island that the family initiated from. I believe, I don't actually know where that's shipping magnate lived, I think it was New York or something. But anyway, we all came from this island called Andros, which is a really beautiful place. And it's like there's more livestock there than humans. Not too many tourists go there. But I think part of what I think is really cool is he published a lot and a lot of his writing is about the beauty of that island, which I think is super cool.

中文翻译:

是的，他深爱着家族起源的那个岛。我其实不知道那位航运大亨住在哪里，我想可能是纽约之类的。总之，我们都来自一个叫安德罗斯（Andros）的岛，那是个非常美丽的地方，那里的牲畜比人还多，游客也不多。我觉得很酷的一点是，他发表了很多作品，其中很多都在描写那个岛的美丽，这真的很棒。

(01:23:12) Lenny Rachitsky

English:

Wow, that was an amazing answer.

(01:23:14):

Two more questions, where can folks find you if they want to follow you online and maybe reach out? And then how can listeners be useful to you?

中文翻译:

哇，这个回答太棒了。

(01:23:14):

最后两个问题：如果大家想在网上关注你或联系你，可以在哪里找到你？另外，听众可以如何帮到你？

(01:23:20) Alexander Embiricos

English:

I'm one of those people who has social media only for the purposes of having work. My phone turns black and white at 9:00 PM at night. But yeah, so Twitter or X, @Embirico. And yeah, if you post in r/Codex, I'll probably see it. So you can go there.

(01:23:40):

How can listeners be useful? I would say please try Codex, please share feedback, let us know what to improve. We pay a ton of attention to feedback. I think, honestly, the growth has been amazing, but it's still very early times, so we still pay a lot of attention and hope to do so forever. And also, I would say if you're interested in working on the future of coding agents and then agents generally, then please apply to our job site and/or message me in those social media places.

中文翻译:

我是那种只有为了工作才用社交媒体的人。我的手机晚上 9 点就会变成黑白模式。不过，我的 Twitter/X 账号是 @Embirico。另外，如果你在 r/Codex 发帖，我大概率能看到。

(01:23:40):

听众可以如何帮到我？我想说，请尝试使用 Codex，请分享反馈，告诉我们哪里需要改进。我们非常重视反馈。老实说，虽然增长很惊人，但现在还处于非常早期，所以我们仍然非常关注反馈，并希望永远如此。另外，如果你有兴趣研究编程智能体以及通用智能体的未来，请申请我们的职位，或者在社交媒体上给我发消息。

(01:24:10) Lenny Rachitsky

English:

Alexander, this was awesome. I always love meeting people working on AI because it always feels like this very, I don't know, sterile, scary, mysterious thing. And then you meet the people building these tools and they're always just so awesome, and you especially, just so nice. And like the examples you shared, optimism and kindness, this is what we want to be. These are the kinds of people we want to be building these tools that are going to drive the future. So I'm really thankful that you did this. I'm grateful to have met you, and thank you so much for being here.

中文翻译:

Alexander，这太棒了。我一直很喜欢结识从事 AI 工作的人，因为 AI 听起来总像是一个冰冷、可怕、神秘的东西。但当你见到构建这些工具的人时，他们总是那么出色，尤其是你，人真的很好。就像你分享的例子：乐观和友善，这就是我们想要成为的样子。我们希望由这样的人来构建驱动未来的工具。非常感谢你参加这次访谈，很高兴认识你，谢谢你能来。

(01:24:45) Alexander Embiricos

English:

Yeah, thanks so much for having me. This was fun.

中文翻译:

是的，非常感谢邀请我。这很有趣。

(01:24:48) Lenny Rachitsky

English:

Thank you so much for listening. If you found this valuable, you can subscribe to the show on Apple Podcasts, Spotify, or your favorite podcast app. Also, please consider giving us a rating or leaving a review as that really helps other listeners find the podcast. You can find all past episodes or learn more about the show at lennyspodcast.com. See you in the next episode.

中文翻译:

非常感谢您的收听。如果您觉得本期节目有价值，可以在 Apple Podcasts、Spotify 或您喜欢的播客应用中订阅。此外，请考虑给我们评分或留下评论，这能极大地帮助其他听众发现这个播客。您可以在 lennyspodcast.com 找到所有往期节目或了解更多信息。下期节目再见。