SDS PODCAST EPISODE 747: TECHNICAL INTRO TO TRANSFORMERS AND LLMS, WITH KIRILL EREMENKO

Show Notes: http://www.superdatascience.com/747

Jon Krohn:	00:00:00	This is episode number 747 with Kirill Eremenko, the founder and CEO of SuperDataScience. Today's episode is brought to you by Intel and HPE Ezmeral Software and by Prophets of AI, the leading agency for AI experts.
	00:00:20	Welcome to the SuperDataScience podcast, the most listened-to podcast in the data science industry. Each week we bring you inspiring people and ideas to help you build a successful career in data science. I'm your host, Jon Krohn. Thanks for joining me today. And now let's make the complex simple.
	00:00:51	Welcome back to the SuperDataScience Podcast. Today we've got a special episode with a very special individual, Kirill Eremenko. If you don't already know him, Kirill is founder and CEO of SuperDataScience, an E-learning platform that is the namesake of this very podcast. He founded the SuperDataScience podcast in 2016 and he hosted the show until he passed me the reins three years ago. He's reached more than 2.6 million students through the courses he's published on Udemy, making him Udemy's most popular Data Science instructor.
	00:01:19	Today's episode is perhaps the most technical episode of this podcast ever, so it will probably appeal most to hands-on practitioners like data scientists and ML engineers, particularly those who already have some understanding of deep learning. In this episode, Kirill details the history of the attention mechanism in natural language models, how compute efficient attention is enabled by the transformer, a transformative deep neural network architecture. He talks about how transformers work across each of five distinct data processing stages, how transformers are scaled up to power the mind-blowing capabilities of large language models such as modern generative AI models. And he fills us in on why knowing all of this is so helpful and lucrative in a data

		science career. All right, you ready for this exceptionally deep episode? Let's go.
	00:02:10	Kirill, welcome to the SuperDataScience podcast. You've been here so many times, actually, even since I've been host of this show. But many of our listeners probably know you as the founder and the original host of the SuperDataScience podcast. You hosted it for four years and now that we're in 2024, it's been more than three years that I've been hosting, so by this time next year, we'll be neck and neck in terms of total episodes hosted.
Kirill Eremenko:	00:02:40	Yeah. For sure, for sure. Very excited about that. Hello everybody welcome. I'm excited. Welcome me. I'm excited to be back on the show. And yeah, Jon, time flies. Three years that you've been hosting it. How do you feel?
Jon Krohn:	00:03:00	I feel a lot more comfortable than I did the very first episodes. When Kirill prepared for me this huge document. It's like 30 pages long, which was amazing. This was before ChatGPT, so he typed that.
Kirill Eremenko:	00:03:14	For sure, yeah.
Jon Krohn:	00:03:17	And it had just tons of useful tips for me getting started on the show. And then we did one episode co-hosting together, and then all of a sudden that was it. I was thrown in the deep end. And so, those first few episodes I was certainly a bit nervous, but now it's two episodes a week, so I'm pretty comfortable. Every once in a while, we have a guest on that I'm intimidated by. I'll tell you that.
Kirill Eremenko:	00:03:41	Yeah. Yeah.
Jon Krohn:	00:03:42	You're not one of them.
Kirill Eremenko:	00:03:47	Good. Good, man good.

Jon Krohn:	00:03:50	But, yeah, so great to have you here. Where are you calling in from, Kirill?
Kirill Eremenko:	00:03:54	Australia, Gold Coast as usual for me these days. And you're in Canada, right? With family?
Jon Krohn:	00:04:00	I am in Canada with my family. So, we're recording this one over the holidays and I've had a wonderful time here. Yeah. Yeah. Yeah.
Kirill Eremenko:	00:04:08	Three more days left of 2023. It's been a crazy year with a lot of AI ML breakthroughs and LLM is the new big thing, right?
Jon Krohn:	00:04:18	Yeah, for sure. LLMs, they have been a big thing for a while, but certainly the end of 2022 was when with the release of ChatGPT people I think became aware of at least the technology, the user interface of this technology. And so yeah, it's exciting to have an episode now dedicated Kirill to large language models, which no doubt throughout 2024 going to continue to be a huge deal. And for years beyond. I don't think there's any doubt that I think the thing that replaces LLMs is probably some ways away, it'll probably be things that kind of combine. I think what's next in AI will probably involve still LLMs. So, everything that you're going to cover in today's episode is going to be relevant for a long time. I'm super excited about the content that we're covering today. I can't wait to learn from you.
Kirill Eremenko:	00:05:14	Thanks, man. I'm super pumped as well preparing for this podcast. I guess I'll start a bit earlier. Last time we appeared on the show with Hadelin, we were talking about cloud computing-
Jon Krohn:	00:05:29	Episode number 671. 671 it was in April 2023.

Kirill Eremenko:	00:05:36	Yeah. So about eight or nine months ago. And we're talking about the necessity of cloud computing for machine learning and data science moving forward and for AI as well. And I just wanted to reiterate that that's still the case. We are still very focused on that. We're helping a lot of people since that episode 'cause that was when we were launching our platform CloudWolf. Since then, over 400 people have signed up, and most of them are data scientists, machine learning engineers. Dozens have received the AWS certification. And so, we are very pumped about that, and it is still the case more than ever that cloud is important for machine learning AI. But we're not here to talk about that. We're here to talk about LLMs. And why is that? Because initially at the start of this year when ChatGPT came out and things like that, or start of 2023. I didn't think that there'll be a lot of demand for machine learning practitioners and data scientists and AI engineers to learn the nuts and bolts of transformers, large language models, GPT models, how they are constructed.
	00:06:47	I thought it would be focused on prompt engineering, but I was very wrong. We've had a lot of demand for a large language models course, and also, we're seeing a lot of employers out there starting to hire LLM engineers. It's a very hot space. It feels to me like... Remember in 2012, data science was the new big thing, and data science salaries were through the roof because there was lack of talent. Nobody knew what it meant, hiring managers didn't know who to hire. It feels that all over again. This had been a 10-year cycle. Now the new hot thing is LLMs and all these companies, the ones that want to be at the cutting edge of technology and implement LLMs inside their operations, inside their... I don't know, business, whatever else, however they're doing their business.
	00:07:38	And it's not just tech companies. We're talking about real estate companies, banks, insurance companies,

consulting companies, and of course all the tech companies. And the salaries for LLM engineers are through the roof, like going up to all the way up to 600,000 for LLM manager or 500,000 for LLM engineer. What do you reckon? Do you think we're at that phase again with data scientists we were like 10 years ago?

Jon Krohn:	00:08:04	I definitely agree. I think it's a complement to the data scientist skillset, and I think that because of innovations like LLMs, we have had continued interest and increased interest in data science roles, machine learning engineer roles. It's one of those things where people worry about automation taking jobs, but automation throughout all of history has changed the kinds of jobs that people have. And this is a perfect example where the ecosystem around data science keeps developing and the tools that we have grow. And so, even though large language models make data analysis easier, making models easier, allow us to have all kinds of automations, the demand is greater than ever because of this ecosystem that evolves and because of the kinds of applications that people expect more and more to incorporate these kinds of technologies. And so, the only... I guess kind of bit of nuance that I would add is that I don't think it's that 10, 12 years ago data scientist was super popular, and that's waned. It's just that this particular niche within data science has now exploded as well.
Kirill Eremenko:	00:09:30	Yeah. Yeah. I completely agree with you. And I think there's this... One something new like this happens or if we learn, I love what's his name, that investor in the US I forgot his name, the one who looks at the past to predict what's going to be... It's natural, right? If we look at what happened there-
Jon Krohn:	00:09:52	I only trust investors that look into the future.

Kirill Eremenko:	00:09:57	I know he has a book, Ray Dalio, he has a book recently about economies and so on. If we look at what happened with the data science trend, it won't wane. But what will happen, I believe with LLMs is there will be more, like right now, I look today there's 1,300 jobs on Glassdoor open job positions that specifically require LLM skills. Some of them like a preference, but a lot of them is actually you need to know LLMs, you need to do this, you need to be doing engineering, you need to be doing scaling, you need to be doing data collection, whatever else. Managing an LLM team, 1,300 doesn't sound like a lot, and that's why the salaries are so high.
	00:10:35	But as time progresses, there'll be more candidates who know LLMs because people will learn. And also, there'll be more companies who will want LLMs. So, the demand and the supply will increase, and eventually the salaries will taper down. That's my prediction that we're not going to be seeing $600,000 salaries for an LLM engineer one year from now. Maybe two years from now, I think there'll be more of a better equilibrium in the market for LLM jobs.
Jon Krohn:	00:11:06	Yeah. Maybe. Yeah. I'm not sure. I 100% agree. I guess because I think it ends up being at the time that data scientists say there were only 1500 of them.
Kirill Eremenko:	00:11:18	Yeah.
Jon Krohn:	00:11:20	There were probably some that were asking half a million dollars or whatever, but there were also, it was probably more normal to be like 200,000 or 100,000. And as time has gone on, the number of data scientists out there has become so large that there's still... I think the shape of the distribution is probably... I don't know if it's about the same or I think there's probably more people who are data scientists making those very large salaries than ever before, even though the demand and supply would have been... The supply would've been more constrained for

sure 10, 12 years ago. But there has also been a lot of seniority developed, a lot of niches developed. And so, I suspect that in terms of total number, there's more people commanding those kinds of half million dollar or more salaries in data science than ever before.

Kirill Eremenko: 00:12:22 Interesting. Interesting. So, in absolute terms, maybe you're right. But if we think in relative terms, if you have LLM skills right now, you're much more well positioned than if you will have LLM skills two years from now. In relative terms, I think you'll be much more competitive.

Jon Krohn: 00:12:54 So, your business wants an LLM to transform customer service. Well, you could iterate on disconnected platforms, struggle with the lifecycle of open-source tools, or iterate freely without creating a single ticket. Register for a free 30-minute workshop and watch a RAG Retrieval-Augmented Generation-powered system be built in record time using Kubeflow, MLflow and KServe, prebuilt notebooks and battle-tested machine learning tools. All in one place is part of HPE Ezmeral Software and powered by Intel Xeon Scalable Processors. Head to hpe.com/ezmeral/chatwithyourdata to join this free cutting-edge event. We've got the link in the show notes for easy access.

00:13:19 Yeah, yeah. It's definitely better to build those skills now, but I think people typically... Once you kind of ratchet up to that kind of higher salary, I think it's kind of rare for it to come down. Instead, you'll develop more skills, you'll be even more on the cutting edge. You'll have LLMs plus some kind of a specialized LLM skill that comes next, and you can add in and you can be like, I've been publishing LLM papers since 2024. So yeah, you'll be this kind of like senior LLM leader by the time.

Kirill Eremenko: 00:13:50 Especially if you listen to this podcast. Jon talks about LLMs a lot. I've been following, of course, I've been following. So, if you want to be an LLM expert, make sure to subscribe. Subscribe to this podcast you're listening to, quite a lot of guests come on to talk about LLMs. I think.

Jon Krohn:	00:14:09	Yeah. I mean, we try to find the biggest and best guests out there. And we don't typically dictate what guests should be talking about, but the most interesting guests in the world, you bring in people from Berkeley or Stanford or other top labs, Google DeepMind. They're either directly or indirectly, typically right now working on LLMs or generative AI in some way. And typically, also getting their own startup going that's involved in some kind of generative AI startup as well on the side going. So yeah, there's a huge amount of opportunity here, it's because it's such a powerful technology, an unprecedentedly powerful technology. And so, there's an abundance of opportunities for us to be making amazing applications, making a huge difference to people and to businesses. And so, yeah, certainly it's been the topic of the year in the year past in 2023.
Kirill Eremenko:	00:15:08	For sure.
Jon Krohn:	00:15:09	Yeah. Wouldn't be surprised if we've got a lot of LLM episodes in 2024, but this is a different episode because typically when I have guests on to talk about LLMs, they are going into detail on some specific technology related to LLMs that they have some particular application. But here today, you're going to be doing a survey and you're going to be introducing the whole area to us from the ground up. Yeah. So, I guess you've been working a ton on an LLMs course, right?
Kirill Eremenko:	00:15:40	Yeah. Yeah. For sure. It was funny, it was in September this year when we realized, okay, we've done quite a big march forward with the cloud platform that we're building, and people are quite happy. We've got a lot of content in there. Let's take a break for a month and build

this large language models A-Z course in line with our A-Z kind of like... not franchise, what's it called like line of courses that we have Machine Learning AI A to Z. Okay. LLM A to Z take us a month. Four months later, I'm only finishing up. There's this 30 intuition tutorials in there. As usual, I do intuition. Hadelin does the practical, and we're doing fine-tuning of an LLM to... He's doing fine-tuning of LLM to a medical dataset, which is very exciting.

00:16:35 I'm doing the theory of transformers from the ground up, GPT models, everything LLMs, and there's like 30 tutorials I counted yesterday, 31 tutorials. Took me four months to put all together because I had to scour the whole internet, read research papers. Of course, the attention's all you need research paper plus all of the... A lot of the other ones, blogs, videos, you name it. And get all these details and from experts in the field from, I don't know, people who are just dabbling in it, blogs that were original a while ago 'cause transformers were introduced in 2017, so there's a lot of information out there on them.

00:17:14 And yeah, finally, I can with great confidence say that we are finishing up. This week our plan is to finish it up on 31st December. And the course is available. It's exclusively available on the SuperDataScience platform. Not available anywhere else. Not available on Udemy or anywhere else. So, if you want to check it out, go to superdatascience.com/llmcourse. And I can confidently say that from what I've seen, I genuinely believe this is the best course on LLMs you can find out there. Yeah. So that's the course we've been working on. It's available. You can't buy it separately. You have to become a member at SuperDataScience, but with that, you also get all access to all the other courses, workshops, which in 2024, we're going to be doing regularly at least once a month, maybe more often on LLMs and other things and other perks inside the membership.

Jon Krohn:	00:18:11	Nice. So, workshops are interactive?
Kirill Eremenko:	00:18:14	Yeah. Yeah. Yeah. Workshop interactive. We're already doing them in our cloud platform. People love them. So, we're going to be doing them in our community on SuperDataScience.
Jon Krohn:	00:18:23	Yeah. Yeah. So, when you say cloud platform, you mean your cloud instruction platform CloudWolf?
Kirill Eremenko:	00:18:28	Yeah. So, we have CloudWolf for people who want to learn cloud and get AWS certified, and we have SuperDataScience.com for anybody who wants to learn data science, machine learning, AI. But having said that, I'm here not to just promote the course. I'm here because I am fresh in terms of my knowledge, like past four months I've been doing LLMs. We're going to switch over back to creating cloud content in the next few weeks, and I wanted to really capture this knowledge inside a podcast and share with people.
	00:19:00	So, my goal is that by the end of this episode, you will understand as a listener, you'll understand the ins and outs of how a transformer model works. We'll be focusing on GPT models, which are decoder only models, and by this end of this episode, you will understand all of the components that are inside a GPT model slash a transformer model, what LLMs are and how they work on the inside. You've heard a lot of podcasts episodes on this show, as Jon mentioned about people talking about applications, LLMs or research, and really cool things. Today is the opportunity to understand how it really all works on a technical level. So, I'm very excited, very excited to share the insights today.
Jon Krohn:	00:19:42	But before we do, before we dive into this Kirill, is this something that should be scary or intimidating for

people? Are LLMs harder to learn than other kinds of concepts?

Kirill Eremenko: 00:19:54 That's a great question. I would say that LLMs are not much harder than learning about neural networks. And if you understand concepts such as back propagation, even a gradient descent, even if you don't know them yet, like LLMs are easier than that in my view, because it's an architectural solution. It's a matter of understanding the building blocks that go into an LLM, and if you really know about our neural networks, it's going to be pretty straightforward. The hard part for me was spending the four months and putting everything together because there's no one source or there wasn't until we've released this course, I don't believe there's this one source where you could get all of the information coherently in one place with all of the little details covered and nothing left. I'll give you an example. I thought two days ago. I thought, "Okay. I'm done with the course. Almost done, just a few tutorials left. I'm just going to revise everything I've learned over the past few months for the preparation for this podcast with Jon."

00:20:58 And then boom, literally yesterday I was sitting there reading another piece of information. I couldn't find any answers. So, I was digging somewhere in Stack Exchange on AI not even in the main thread in a chat inside Stack Exchange where some researcher was answering somebody's question, and I had an epiphany. One of the biggest breakthroughs that happened in my learning of LLMs happened literally, yes, because it was so hidden, and it's not obvious. It's not in the research paper. Some people who know how these things work, they assume you understand. Some people who don't know in the most cases, we'll probably miss this part, and we'll talk about it later. It's about training of an LLM, how it actually happens within segment parallelization of training. So, to answer your question, it's not hard if you have all the information, and I'll do my best to present all of it here. Again, if you want to dive deeper, check out the course that Hadelin and I are releasing.

Jon Krohn: 00:21:53 Nice. Yeah. LLMs A to Z available exclusively in the SuperDataScience.com platform. All right. Let's start with the basics of LLMs, Kirill.

Kirill Eremenko: 00:22:03 Okay. All right. Here we go. Super excited. So, the ingredients of an LLM, you first of all need a lot of data. You need tons and tons of data, hence large language models. You need a transformer architecture, which is the core of ChatGPT, Claude by Anthropic like basically even Stable Diffusion, right? It's an image system, but it actually also has GPT transformers in its core. You need pre-training. You need a lot of pre-training, and that's where most of your compute time cost is going to go. That's the most expensive and time-consuming part of the large language model. Then you can also optionally have reinforcement learning with human feedback and also optionally, you can fine tune the large language model on domain specific data if you like. So those are the five core ingredients of large learning models.

00:22:59 And of course, there's other variations, other additions to them, a brief history of large language models. By the way, this is not mine originally. This history over you is something I saw in a lecture on YouTube by Andrej Karpathy. We'll link to it in the show notes if you want to check it out. But in summary, in 2003, Yoshua Bengio and some other scientists published a paper called a Neural Probabilistic Language Model. This was one of the first attempts to model a language with a neural way. Then in 2014, Ilya Sutskever who currently works at OpenAI published a paper called Seq2Seq Learning with Neural Networks. The importance of this research paper was that it combined LSTM networks, two LSTM networks, into a encoder-decoder structure. So, the LSTM networks were originally invented back in the 90s, I believe, by... I forgot. He is a German scientist.

Jon Krohn: 00:24:00 Is it Jürgen Schmidhuber?

Kirill Eremenko: 00:24:05 Jürgen Schmidhuber was his supervisor. This was [inaudible 00:24:11] or I forgot his name. So, he was the student under Jürgen Schmidhuber. So well, the two of them invented.

Jon Krohn: 00:24:18 I think it's Hochreiter.

Kirill Eremenko: 00:24:21 Hochreiter, yes. My apologies for mispronouncing. So, Hochreiter was the student of Schmidhuber, and by the way, it's really interesting to watch Schmidhuber's, I remember back from 2017 watching his YouTube when we were preparing Artificial Intelligence A to Z watching some of his talks and comments on AI maybe it's changed, but back then it was very, very dark. It's got a very dark outlook on the impacts of AI. And even back then, all the bells that people are ringing, now in terms of the dangers of all LLMs, and we're not going to go into detail on that, but he was already talking about the dangers of AI back then, and he's got pretty dark outlook in my perspective, may have changed or maybe other people have different perspectives.

00:25:02 Anyway, so Ilya published this paper, and it was a first paper where two LSTMs were combined in the encoder and decoder structures. So basically, and this was for translation. So, you take in the sentence and then LSTM, if you need a refresher as a type of recurrent neural network. You take a sentence you want to translate. You put it into this neural network one word at a time, and then this is the LSTM. Then at the end of this encoder LSTM you have a vector which transfers all of that information to the decoder, which is another LSTM, and then the decoder now unravels it and makes the translation. That was a design, and I remember we're playing around with this in 2017 for our Artificial Intelligence A-Z course, and the big problem with that architecture, even though it was already showing some promise, the big problem, two big problems with that architecture, first one is it has to take in the information sequentially. So, it's very hard to parallelize and you can't scale it easily for training. Second thing is that because you have this one vector, word vector, or context text vector, whatever you want to call it, between the encoder and decoder, there's one vector that connects the two. You're trying to squish all of this information into one vector and then pass it on to the next one. And so, the longer your input the harder it will be. Effectively, you're creating a bottleneck, and that's why LSTMs they have a lot of merit, very eloquent solution to some of the problems there with the vanishing gradient and so on. But at the same time, they didn't go that far, mostly because of these two issues.

00:26:41 Yeah. So that was in 2014. And again, in 2014, just around the same time there's another paper by Dzmitry Bahdanau who published a paper called Neural Mission Translation by Jointly Learning to Align and Translate. And he introduced a concept on top of this architecture that with the LSTMs, he introduced a concept of attention. Well, actually, he didn't call it attention, he was going to call it something else, but Yoshua Bengio was one of the all co-authors on the paper as well, and he suggested the word attention. And so, the way that this paper works is basically it's like, okay, let's use this LSTM architecture, recurrent neural nets for language modeling, but in addition to in order to solve the bottleneck problem, we're going to now give this extra mechanism of attending towards. So, allowing the neural network to remember certain characteristics, certain information about different words, and store it separately so that it can then be used in the translation part or in the output part.

00:27:47 And this was a really cool solution or really cool way to address the bottleneck problem. So, the way to think about it and the way Dzmitry Bahdanau explains this in an email reply to Andrei so when he saw that, he actually asked Dzmitry Bahdanau how he came up with this idea, and he replied, and this is public on one of the YouTube videos by Andri, the way he explained it is like Dzmitry's first language is in English. So, he had to learn English. So, imagine when you're learning to translate from English, from your language to English or the other way around as a human, you're writing out this translated text.

00:28:29 Let's say you have your language like a paragraph, and you're writing out this paragraph in English, so you're writing it out, and let's say you're on the seventh word. So for the eighth word. You don't just have it all in your head. You don't have the context, the whole meaning of your initial paragraph in your head. You look back. You look back and you look at different words. You combine them again, revisit them, and then you translate the eighth word. And then for the ninth word again, it's not all in the head. So, think about it's like for machines, we were thinking, oh, let's have in the LSTM paper, by Ilya Sutskever, everything is in this one vector. It's like having the whole paragraph you want to translate just in your head and trying to translate. Whereas the addition of attention is like in a human version, where you're looking back throughout while you're translating, you're looking back at the original text to help you translate each sequential work.

Jon Krohn: 00:29:20 Empower your business with Prophets of AI, the leading agency for AI and robotics experts. Whether you seek a captivating keynote speaker, a company workshop host, or even guidance in implementing AI seamlessly into your organization, Prophets of AI connects you directly with the luminaries shaping the future of AI, such as Ben Goertzel and Nell Watson, both of whom have been phenomenal guests on this very podcast. Whether you are a large global enterprise or just beginning your AI journey, Prophets of AI have a solution for you. Their speakers have graced the most prestigious stages around the world, and now you can head to ProphetsOfAI.com yourself to see their full roster or to the show notes where we've got their contact link.

00:29:59 Makes perfect sense. It would bridge conceptually this kind of gap in capabilities between pre-attention. It would be a relatively easy task if it happened to be the case that you could kind of like one-to-one map from one language to another word by word. Where you could just be like the dog runs, and then you translate that into some other language, some target language French, Russian or whatever, Chinese. And somehow it ends up being that it's still just the dog runs exactly three words in exactly that order. Then probably without attention, translation would be easy. But attention, as you're saying, it allows you to consider more than just this linear sequence. You are considering all of the context potentially not just before a given next word, but also context after it in order to pick the seventh word, the eighth word, the ninth word or [inaudible 00:30:58].

Kirill Eremenko: 00:30:58 Exactly. Yeah. That's right. And yeah, different languages have different styles and different rules, and it's much more complex than just translating word by word. And interesting point is that we're talking about translation here is because it's the most obvious problem for dummy or practice or training problem or a challenge to solve with a machine solution like ChatGPT not the design is not for translation, right? It's for prompt engineer, or prompts and answers, and text generation, right? But like originally, the way this field evolved is all these papers, they're focused on translating because that's kind of the obvious challenge or obvious problem you can solve. And speaking of this, the next paper, the final one that we're going to mention today, which was published in 2017 by a whole team at Google. I believe it was eight researchers at Google, is called Attention is All You Need.

00:31:56 And again, it was focused on translation, and that's the paper that introduced transformers. So, a cool trivia, a piece of trivia is that transformers were originally designed as a translation mechanism because of all this history, but later, only now... Well, afterwards, they were adapted to BERT models, GPT models. For GPT models for text generation. So what's very interesting and phenomenal about this paper, Attention is All You Need. It's like it's in the name. What they did is they took this LSTM structure encoder-decoder, plus the attention that was introduced by Dzmitry. Then they're like, "Okay, let's throw out the LSTM. Let's just keep the attention part. Why do we even need this?"

00:32:38 The Dzmitry's paper in 2014 solved the bottleneck problem or addressed the bottleneck problem, but then they still had that sequential processing problem that it wasn't easy to parallelize. But when you throw out the LSTM and now you only have the attention part, now you don't have a bottleneck problem and you don't have the step-by-step, sequential input processing and generation problem. All of a sudden, you open a whole world of possibilities as long as you can pull it off, as long as you can create a architecture that works based on just attention.

00:33:12 That's exactly what this team at Google did. This was back in 2017. As you can see, you correctly mentioned, Jon, transformers have been around for a while, but it took five years for it to really break through in the form of

ChatGPT and gain popularization, and now it's all over the place. That's where we are at and that's the paper we're going to be going through today. It's going to be a very interesting challenge because it's an interesting challenge for me to put together an explanation of how a transformer works without any visual aid. If you are in front of your computer, feel free to open up this paper.

00:33:53 It's called Attention is All You Need, you can download on arXiv, and open up the diagram of the transformer. You can follow along. You'll notice that I'm simplifying things and leaving out a few items that are not important, but we'll talk about the most important things. If you're not in front of a computer, no worries. Whether you're driving, running, cycling, whatever, relaxing on the couch, I believe I've come up with a method that will help you visualize it in your mind. Just be careful if you're driving. Of course, don't get too carried away, please.

Jon Krohn: 00:34:21 Nice. I think a key thing here to mention is, you already touched on it there a little bit, but I'll dig into it in more detail is that even though this key paper Attention is All You Need from that team at Google is from 2017, the concepts underlying large language models today in terms of neural network architectures are the same. Seven years later, it's just a bit a matter of scaling up in terms of sizes of the training data available for training these LLMs, and then how many of the attention... These things called attention heads, which I'm sure you'll get into later, but the number of attention heads in the LLM gets bigger and bigger. Also, context windows have been getting bigger. So it's basically just size. It's just a matter of scaling up. But fundamentally, the neural network architecture is the same, it's just bigger.

Kirill Eremenko: 00:35:16 Absolutely.

Jon Krohn:	00:35:17	That's why it makes sense to dig into in so much detail this one particular paper from-
Kirill Eremenko:	00:35:20	Absolutely. In addition to what you said as well, I was watching a video by Andrej Karpathy and he pointed out that not just the architecture is the same, but even a lot of the parameters or hyperparameters, the ones that picked in the paper. There haven't been any super major breakthroughs or super major changes. I don't know the number of the embedding size of the vectors or how they did not batch normalization, but layer normalization. Something we won't be talking about is an additional step or these residual connections. Only a very few improvements have happened over the past seven years. Most of the stuff that we're using these days, of course, to what we know, there's a lot of proprietary things with OpenAI as Elon Musk called it becoming the super closed AI for-profit organization. We don't know exactly how they're structured, but most of the things that we're proposed in this paper still work to this day and haven't been outperformed by any significant modifications that have happened since.
Jon Krohn:	00:36:29	All right. Let's get into your visual of transformers from the Attention is All You Need paper. By the way, we will have links to all of these papers that were mentioned in the show notes.
Kirill Eremenko:	00:36:39	For sure.
Jon Krohn:	00:36:39	[inaudible 00:36:40] in one place. If you're driving, don't visualize too hard, keep an eye.
Kirill Eremenko:	00:36:44	Yeah. I wanted to say if you find parts of this too complex, especially when we're talking about that attention mechanism, it can become a little bit technical. Feel free to skip through a few minutes and get to the rest because later on after the technical stuff there'll be more very

important non-technical aspects as well. In addition to that caveat, I wanted to mention a few other ones before we dive into the components of a transformer.

00:37:14 The first one is a transformer has two parts to it. It has an encoder and a decoder. We are going to be talking about just a decoder. Why is that? Because a GPT model with ChatGPT, for example, is a decoder-only model. This is a good piece of trivia. It's not obvious when you think about it like, yes, ChatGPT uses a transformer but doesn't use the full transformer. You throw out the encoder part, you only use the decoder. Whereas another piece of trivia, if you look at a BERT model, you only use the encoder part, you throw out the decoder part. But we're going to be looking at just the decoder part of a transformer. Another caveat, GPT stands... Yeah, go ahead.

Jon Krohn: 00:37:52 If you don't mind here, I'll quickly also... I'll add to give a bit of context on why you would use just an encoder or use just a decoder. A decoder-only model, like the GPT series, it is designed primarily to be generating text. It's this generation, which is the decoding part. It's easy to remember that decoder part of the transformer. When would you use it? It's when you want a model that is optimized for outputting generating like a generative AI model, like a chatbot, ChatGPT. These series of architectures, decoder-only, the inverse encoder- only ends up being really helpful when you're not doing a generative task. When you want to use the information in some other way, so maybe you're building a classifier or some other machine learning model, maybe a ranking algorithm, you're going to convert all of your documents into some abstract space.

00:38:49 You'll use the encoder part of the transformer to do that encoding. So you don't need the decoder, you're just concerned about encoding in those cases. BERT is

		definitely the most famous example of a... B-E-R-T from Sesame Street. In some of the early, you didn't talk about these particular papers, but there were a bunch of early attention and transformer-related papers where there was like ELMo, and then there was BERT. That was a bit of a funny trend now some years ago. Anyway, sorry, I interrupted you.
Kirill Eremenko:	00:39:30	Sure.
Jon Krohn:	00:39:30	Yeah. Encoder-decoder, and then you're on to the next thing.
Kirill Eremenko:	00:39:33	Yeah. The next thing was another... Just so we're on the same page, another piece of trivia. GPT stands for generative pre-trained transformer, hence that transformer because it's used in GPT models. One final thing I wanted to note is that even though basically in our course that we released, we cover everything from encoder to decoder, all the steps, everything completely, but for simplicity's sake and to keep things concise in this podcast, we'll focus on the decoder only and also because that's what's relevant to GPT models. All right. Cool. Let's get straight into it. I'm super pumped about this.
	00:40:11	The Attention is All You Need paper has a diagram. Obviously, we're going to have to visualize this. We're going to simplify some things. The best way to think about it is imagine five boxes, five boxes stacked on top of each other, all a blank. We're going to be filling them in. They're connected from the bottom box to the next box up with an arrow and then to the next box up with an arrow, and so on. Basically, that's the architecture of the transformer. It starts from the bottom and goes up through these boxes, and we're going to now fill in these blanks.

Jon Krohn:	00:40:44	It's like a five-story apartment building. You start on the ground floor and then you move to the second floor, then to the third floor, all the way up.
Kirill Eremenko:	00:40:51	Yeah. In case you're five-year-old listening to this, Jon has got you covered with his five-story analogy. I'm pretty sure our audience can visualize five boxes. But thanks. That's a good addition. All right. Yes, five stories starting with the first floor. How do you do it in Canada? In Australia, it's a ground floor and then it's first floor.
Jon Krohn:	00:41:24	As we often discuss on air with guests, deep learning is the specific technique behind nearly all of the latest AI and machine learning capabilities. If you've been eager to learn exactly how deep learning works, my book, Deep Learning Illustrated is the perfect place to start. Physical copies of Deep Learning Illustrated are available in seven languages, but you can also access it digitally via the O'Reilly learning platform. Within O'Reilly, you'll find not only my book, but also more than 18 hours of corresponding video tutorials. If videos, your preferred mode of learning. If you don't already have access to O'Reilly via your employer or school, you can use our code SDSPOD23 to get a free 30-day trial. That's SDSPOD23. We've got a link in the show notes.
	00:42:15	Canada and the US, there's no floor zero. The first floor is the ground floor, but in Europe... What side of the road do you guys drive on in Australia?
Kirill Eremenko:	00:42:29	On the UK side of the road, on the left.
Jon Krohn:	00:42:31	On the UK side. So you drive on the left side of the road and you have a floor zero.
Kirill Eremenko:	00:42:37	Floor G. All right. Well, for simplicity's sake, we'll use the US method. Yeah?

Jon Krohn:	00:42:45	Before we go into that. It makes a lot of sense to have a floor zero. I know we're going to go with floor one because it does definitely make a lot of sense. But in terms of just the way that numbers work, you shouldn't just go from one to negative one. You've skipped an integer.
Kirill Eremenko:	00:43:03	Yeah, that's a good point.
Jon Krohn:	00:43:05	It makes a lot of sense anyway.
Kirill Eremenko:	00:43:06	But in some countries, due to, I believe, beliefs or superstitions, they omit certain floors or they use floors. For example, in China, the number eight is very popular. I was in one building where I looked at the elevator buttons and it was like... I don't remember of eighth floor because my friends were staying... They lived on the 19th or 18th floor something. It goes like 16, 17, 18, 18A, 18B, 18C, 18D, 18E, 19, 20.
Jon Krohn:	00:43:39	I wonder if they skip 13 in China. They skipped 13 in my building in New York, my residential building because they'd have to charge less rent.
Kirill Eremenko:	00:43:47	Or when you sell it... As a developer, when you're selling it, you're going to have to convince buyers or sell it for less.
Jon Krohn:	00:43:55	But beware all you 14 people, you're in a very unlucky floor, you just don't know.
Kirill Eremenko:	00:44:01	Well, it might be 12 people if you add the ground floor into that.
Jon Krohn:	00:44:10	All right. Anyway, so we're going to go with the American system for this analogy. We're on level one, which is also perhaps for our European and Australian listeners.
Kirill Eremenko:	00:44:20	Level one. Let's imagine you have... Level one is called input embedding. You have a whole sentence of vector of

words, you want to give them to a machine, a transformer. Obviously, machines can't really work with words, they need to work with numbers. We want to convert each one of these words. Well, when I say words on this... If you've been listening to this podcast, you've heard Jon many times talk about tokens. That's what LLMs work with tokens. They're not necessarily full words, they're maybe half words, parts of words. But for the sake of this podcast, I'm going to be using tokens and words interchangeably. You have this series of-

Jon Krohn: 00:45:04 Really quickly.

Kirill Eremenko: 00:45:04 Go ahead.

Jon Krohn: 00:45:04 For a great intro episode for people who want to dig into those subwords specifically episode number 626 is all about subword tokenization.

Kirill Eremenko: 00:45:15 Fantastic. Check that one out. It's quite an important topic. We're going to be talking about words and subwords and tokens, all the same because that's not the focus of this episode. You have these words. Now for each word, you're going to need to create a vector and not just a random vector, but a context-rich vector. And I'll give you an example. So Jon, let's do a quick riddle. Imagine this equation. If you take king the concept of the word king, you know this one, right? Yeah. You subtract the concept of the word man and you add the concept of the word woman, right? What is your answer King? Minus man, plus woman. What are you going to get?

Jon Krohn: 00:45:57 Let me puzzle on this for a moment. Maybe... Is it princess? I feel like I'm in the right neighborhood.

Kirill Eremenko: 00:46:06 No, it's not princess.

Jon Krohn: 00:46:09 It's Elizabeth.

Kirill Eremenko:	00:46:11	Almost. It's a queen. Basically, if you take the characteristics or the features of the essence of the word king, every word in the English, English language has an essence which can be described with certain features. For example, a king is wealthy. Generally, kings on average are pretty wealthy. A king is pretty authoritative. They can dictate to people what they do in their kingdom. [inaudible 00:46:45].
	00:46:47	Yes, they are royal, right? But they're not just royal, they are in charge of the kingdom. A prince is royal, but a king is royal and is in charge. He has authority of over the kingdom and things like that. You can describe words with different features. If you take the essence of the word king and you subtract the essence of the word man, and you add the essence of the word woman, naturally, intuitively, you get the word queen, even though we're doing arithmetic with words and not numbers. This hints towards the fact that you can actually describe a word with lots of different features and then do arithmetic using this features. So in the sense of transformers, what we use are vectors... They're called embedding vectors, and they have 512 dimensions. Quite a lot of features, 512 to be precise are-
Jon Krohn:	00:47:41	That's specific to this Attention is All You Need paper. That number of dimensions is arbitrary. It could be-
Kirill Eremenko:	00:47:46	Yes, you can change. You make it 256, make it 10 million if you want, but it'll obviously impact.
Jon Krohn:	00:47:52	Yeah, compute efficiency versus, so the more dimensions you add, the more numbers you need to represent the location of a word in this embedding space. If you have 512 dimensions versus 256, it's twice as much information you need. However, that could be worth it. It could make more nuance. As you say, as you increase the number of dimensions towards infinity, the improvement

in quality of your embeddings for whatever downstream application you have doesn't get better at some point. So you need to decide, all right, let's just cut it off here at 512.

Kirill Eremenko: 00:48:36 We're not going to go into detail on how this is done because there's solutions on this. You can look up bag-of-words model. You can look up the n-gram model, you can, for example, look... You could just use already an existing embedding for all the words. There's ways of doing it. Just the main takeaway from this is that the vectors that are created in this first box, the vector embedding of the words, they're not just random vectors. They are vectors that capture the semantic meaning of the word. Because this is language, we'll be talking about different types of meaning and they capture semantics. Semantic meaning is the dictionary meaning. If you open up the dictionary, you'll describe what an orange is, what an apple is, what a horse is, what an ocean is, and these vectors capture them. In your vector space, which you'll have, it's not just a random series of vectors that's pretty easy to create just a random series of vectors, unique vectors for words.

00:49:31 But you'll actually have vectors for every single word in the English language, which can range between 50,000 words to 600,000 or a million words, depends on how you look. Let's say 200,000 words in the English vocabulary. For each one, you'll have a unique vector, but not just a random one. They will be capturing semantic meaning, and that implies that, words that are similar will be close to each other. An apple will be close to an orange, close to a banana, whereas I don't know, like a car will be somewhere else in this vector space in a different location because they're not that similar like the adjective good and great and excellent will be close to each other. The adjectives bad and terrible will be close to each other.

		You'll just imagine this multi-dimensional 512-dimensional vector space, which is impossible to imagine.
	00:50:22	But in this space, vectors representing similar words are going to be similar to each other. That's box one, right? We've converted our text into every single word of our input now has its own semantic meaning or embedded vector, which captures semantic meaning. Good. Moving on to box two. Box two is called positional encoding. Why is positional encoding important? What is it? Let's look at this example. Horses eat apples, right? It's a grammatically valid English sentence. It makes total sense, means something. Now let's take the same words and rearrange them, apples eat horses. It's a grammatically correct English sentence. It follows all the rules. It has exactly the same words, but it means something completely different. In fact, it's nonsensical. Apples don't eat horses. That just illustrates that word order-
Jon Krohn:	00:51:22	You just haven't done enough LSD.
Kirill Eremenko:	00:51:28	Yeah, in some weird dreams maybe-
Jon Krohn:	00:51:31	Carry on.
Kirill Eremenko:	00:51:34	Yes. Or some people it might make a lot of sense, anyways. But the point is that word order in a language is important and it can significantly affect meaning. When the state-of-the-art for language processing were recurrent neural networks and LSTMs, that wasn't a problem because they inherently preserve word order. They take the words in sequentially. Whereas transformers have a major breakthrough in the sense that they take the input in parallel so they're much faster, but at the same time, that creates the disadvantage that they don't inherently preserve word order. Therefore, in order to correctly convey the meaning of a sentence into a

transformer model, you have to have positional encoding. Again, this is one of those things that we're not going to dive into. There's many ways of achieving positional encoding.

00:52:31 The one they use in the Attention is All You Need paper is very elegant. It uses cosine and sine functions. It's deterministic for each position in the input sentence. But in a nutshell, what happens is to each one of these vectors that we have, for our words, we have a word. Let's say it has a sentence or the input, let's say it has 50 words, so it means there's 50 positions to each one of the vectors that we've created, the embedding vectors. We add a small number which represents the position of that word in the sentence, not a small number, like a small vector. We add to each one of those vectors, we add another small vector, which represents the position of that word in the sentence, and that is a mechanic. We are just creating the environment for the transformer during training to leverage this, right? Our job is to encode the positions into these vectors. The transformer will learn how to use that mechanism that we're giving to it through this approach. That's what box number two is positional encoding. All good, Jon? Anything to add?

Jon Krohn: 00:53:46 Nice. I'm keeping track here. Floor one, input embeddings where we're getting each of our words put into a vector space, a location in the vector space. Then when we go up to the second floor, the floor you just finished describing, you've got the positional encodings. Then now we're going to go up to floor three.

Kirill Eremenko: 00:54:06 All right. Floor three is the middle and is the most important part. This is the attention mechanism. That is the heart of the transformer. That's like what's most revolutionary about this model, and that really enables a lot of what we're seeing. A lot of this text generation. If you think back to LSTMs and when they were generating texts, a lot of the time was nonsensical. It was no coherency to it. Cohesion, always get confused between the two. Whereas transformers, a lot of the time indistinguishable from human text, and the main reason for this is this attention mechanism. We're going to talk about attention mechanism, and before we do, I'll give another example. Let's think about this sentence. The dog did not cross the street because it was too tired, right? You know this one, Jon?

Jon Krohn:	00:55:04	No, actually.
Kirill Eremenko:	00:55:05	Okay. The dog did not cross the street because it was too tired. What does the word it mean? Which noun does it refer to in this sentence?
Jon Krohn:	00:55:15	The dog, of course.
Kirill Eremenko:	00:55:16	The dog, obviously. Because the dog, it was too tired. But now let's change one little thing, one word about the sentence at the very end. The dog did not cross the street because it was too wide. Which word does the word it mean now? What does the word it mean?
Jon Krohn:	00:55:31	Nice. Now it's the street. That was a great example.
Kirill Eremenko:	00:55:34	It's a street. Right. This is actually from one of the TensorFlow or Google libraries visualizing transformers. They use, I think the animal did not cross the street because it was too wide. Too tired or too wide. By changing one word at the very end of the sentence, after those nouns, after the word in question, the word it, by changing one word, we can change the meaning of a preceding word of the word it in this case. In one case it means dog, in one case, when it's too tired, in one case, it means street when it's too wide. That shows us that there's not just semantic meaning behind words, there's also contextual meaning. There's lots of types of meaning.

They can be sarcastic meaning, they can be emotional meaning, there's lots of different meanings in linguistics. But the two main ones that we are focusing about here is semantic meaning we already have that in our vector embeddings.

00:56:25 But now there's a new meaning that we don't have captured in our vectors, and that is contextual meaning, and that depends on the words around the word, depends on the whole sentence or the whole paragraph or the whole text that we are processing. Contextual meaning is also important and it can alter the meaning of the word that we're looking at. The goal of attention mechanism is to capture, to create new vectors. We have vectors that embed semantic meaning that then have positional coding. Now we want to enhance them even further and create vectors that contain contextual meaning. That's what the attention mechanism addresses. I'm going to go through how the attention mechanism works. It's going to be a bit more technical than everything else we've had so far in the podcast. It'll be the most technical part of this podcast, of this episode.

00:57:15 I'll do my best to simplify it as much as I can. But if you find it a bit too technical, maybe skip forward a bit and come back to it when you're in front of a computer and you can open up the Attention is All You Need, research paper, or again, you can always come to our course where we dive into these things over 30 tutorials. But let's do our best to dive into it. We're going to be talking about performing operations with vectors. There's three ways to think about it. The most mathematically correct way is to think about them as linear matrix operations. You have a vector, you apply a linear matrix operation to it, and then you get a resulting vector. When we say we want to create a vector from this vector, that means we're just applying a matrix operation. Second way to think about it is neural network mentality or approach is you think of a vector as an input layer into a fully... Or an input series of nodes, let's say, 512 nodes.

00:58:13 That's the input to a fully connected layer and output is if you want a 512 dimensional vector, it's 512 nodes. You want a bigger small vector, less, more nodes. Both layers are fully interconnected and there is no activation function. That's basically how we modify or create a new vector from an old picture if you think about it from a neural network's perspective. The final way is the most simplistic way to think about it is if you're not very good with matrix operations, if you're not good with neural networks, you just want to visualize it somehow. It's like a matrix. A vector has 512 features. You want to create a new vector also, let's say, with 512 features. Well, the first feature of the new vector will be a weighted sum of all of the features of the original vector.

00:59:01 The second feature of the new vector will be a weighted sum of all the features of the original vector and so on. It's always a weighted sum. The weights are obviously random and different, and they are adjusted through training, through a process called backpropagation. The neural network will learn, or transformer will learn how to improve those weights. Whether you're thinking about as a matrix operation, a fully connected layer, or the simplistic approach, these weights will be adjusted so that the transformer can perform the task we want it to perform in the most correct way, in the desired way. That's a caveat about vectors. So let's dive into this attention mechanism.

00:59:41 The way it works is we have these vectors. Let's say we're going to look at a sentence. Apples are a type of delicious blank. We want to predict the next words. We have six words, apples are a type of delicious, and we want to predict the seventh word, which it's probably going to be fruit, right? But it's a blank at the moment. We want to predict what that is. In order to do that prediction, we're going to enrich our vectors with contextual meaning. We're going to look... As an example, this is done.

01:00:13 This process I'm going to describe is done for each one of the words. Each one of the six words that we have, the attention mechanism is going to be applied. But we're going to look at just one word just to keep it simple, and you're just going to pick one word as an example, and then just assume that the same thing happens for the rest of the words. For each one of these words, from the original vectors that we had, well, the vectors with the embedded vectors with the positional coding, we're going to create three new vectors for each word. It's going to be a Q vector called the query vector, a K vector called the key vector, and a V vector called the value vector, so Q, K, and V.

01:00:50 We'll have a QKV vector for the word apples, QKV vector for the word are, and so on. The QKV vector for the word delicious. Now the Q vector, and they're all created whichever way you want to think about it. Whether it's a matrix operation with a neural network, simple way that we just described. There's three ways. Pick one of those and just think of it that that's how they create. There's basically a matrix of weights for creating Q vectors in this attention mechanism, there's a matrix of weights for creating Q vectors, a matrix of weights for creating the K vectors, and a matrix of weights for creating the V vectors.

01:01:24 Three separate matrices that all need to be learned through training. Now the Q vector is a vector, which is saying what this word is interested in. The word delicious has its Q vector, and the Q vector of the word delicious is a manifestation of what this word delicious wants to attend to. When this attention mechanism is trained and it's up and running, the word delicious will be looking through the rest of the words in the sentence, and it'll be using its Q vector to communicate, "I'm looking for this. I'm looking for this concept. Do you have this concept? Do you have this concept? Do you have this concept?"

Jon Krohn: 01:02:08 On floor one, when we had the word delicious, the word delicious just has a location in our vector space and that word delicious is the same no matter what. Its meaning is the same as word to that. But by the time we get to floor three here, and we have this attention mechanism, these Q, K, and V vectors that word delicious that you're referring to, you're not referring to that in the abstract. You're talking about a specific occurrence of delicious in a specific sentence, in a specific...

Kirill Eremenko: 01:02:46 Good point, yes.

Jon Krohn: 01:02:47 It has way more... These Q, K, and V vectors reflect all of the extra context that deliciousness has. So going back to the it thing that you were saying, these K, Q, and V vectors would allow us to be able to determine that it is referring to the dog or it is referring to the road, whereas in level one, it is just it.

Kirill Eremenko: 01:03:09 Yeah, exactly. In that example, the Q vector for the word it is going to contain information about what that word it is looking for. In one case, it'll be looking for an animal or a living noun or an actor. In another case, it'll be looking something, an obstacle, a street, or something like that. It's obviously not going to be humanly intelligible because this is all going to be trained through backpropagation and lots and lots of iterations in many epochs. But you can think about it that way. Each word needs some information about the context of this sentence that we are specifically talking about in order to then become context enriched. In our example, apples are a type of delicious blank, the word delicious is looking for certain things. And what it's looking for is going to be we want the transformer to put that into the Q vector. So that's the Q vector for delicious.

01:04:14 Next, every word, as we discussed, every word will have the Q, K and all three vectors. Now the K vector. So the K vector is like an indexing mechanism. It tells us what the vector contains. So the V vector will have the value so the thing of interest. So each word will have something of interest that it can give to other words for their context, and that is stored in the vector V. Whereas, the K vector is going to be communicating what is stored in this V vector. So each word will have a K vector. Like the word apples will have a K vector. The word Are will have a K vector. A will have a K vector. Type will have a K vector. Each one of them will have its own K vector.

01:04:57 And now what's going to happen is we're going to compare, remember we're doing this attention, we're doing this for the word delicious. So the Q vector of the word delicious is going to be compared, and I'll explain what that means in a second, it will be compared to each one of the K vectors of all of the words in the sentence including itself. So the Q vector for delicious. Say I'm the word delicious. I'm looking for certain information. That's what I'm looking for, is described in the Q vector. Now I'm going to go take that Q vector or this form is going to take that Q vector and it's going to compare it to the K vector of the word apples. If those vectors are aligned, that means apples has what I'm looking for, and I need to take the V value from apples and include it in my context. If those vectors are not aligned, then apples is completely irrelevant to what I'm looking for, and I should disregard whatever apples has in the V vector.

01:05:53 Then I'm going to go to the next word, are. Are is a plural verb. Maybe it has some meaning that's relevant to the word delicious that it's looking for. So the Q vector for the word delicious is going to be compared to the K vector for the word are. If they are aligned, then yes, the value inside the V vector of the word are is important for the context of the word delicious and should be taken into account. If they're not aligned, it should be disregarded. That value is not important to the context.

01:06:27 That process keeps happening, including of itself. So the Q vector for delicious will be compared to the K vector for delicious. If they are aligned, that means whatever value is inside the V vector for delicious is important to the context of the word delicious. If they're not aligned, that means it's not important. It should be disregarded. So that's how Q, K and V work. We'll talk about the mathematics of how this actually is implemented just now, but I just wanted to check, how did that sound, Jon?

Jon Krohn: 01:06:56 Yeah, pretty good. I think it's about as well as you could without visuals, like you say. It's a little bit tricky to keep all these things straight, but fundamentally for any given term, any given word in a sentence, so like your sentence, apples are a type of delicious blank. In each of those words, we've got the Q vector, the K vector, the V vector, and together, that information allows us to map positions and context together to effectively attend to these words and understand them in context and allow us to predict what a good final word is at the end of the sentence, apples are a type of delicious blank.

Kirill Eremenko: 01:07:50 Yeah. Yeah, exactly. And so that is achieved ... The word of interest, in our case, delicious. You take the Q vector. And then for each other word in the sentence including itself, you look at the K vectors and the V vectors, so you compare Q and K, and if they are aligned, then you take the V vector as part of your context. If they're not aligned, you don't. So how is this achieved mathematically? Well, it's quite an elegant, straightforward solution. You take the dot product of the Q vector for the word delicious and the K vector of whatever other word you're looking at. So let's say delicious and apples. So you take the Q vector of delicious, you take the K vector of apples and you calculate the dot product.

01:08:32 In case you need a refresher, dot product is basically the multiplication. You take the absolute value of each one of these vectors. You multiply those two absolute values and then you multiply it by the cosine between those two vectors. If two vectors are perpendicular, then their cosine is going to be zero, and so their dot product is going to be zero. And if they're aligned, if they're pointing in somewhat the same direction, then their dot product is going to be non-zero. And the more they're aligned, the more they're pointing in the same direction, the more, the higher their dot product will be. So that kind of reflects the logic that we just described of how we want the alignment of vectors.

01:09:12 And the interesting thing is that we're operating in a 512-dimensional space, and in a 512-dimensional space, two vectors are always going to be perpendicular, unless they share a projection into at least one of the dimensions. But because there's so many dimensions, the chances are that most vectors are going to be perpendicular. And so that allows our transformer, that gives our transformer a lot of room for maneuvering to keep a lot of vectors at zero as a result of the training. A lot of the dot products are going to be zero except for the ones that really matter.

01:09:45 So in the case of delicious, for the word delicious, we want as much context as possible because we're going to be predicting the next word. So obviously, we need something from the word apples because if we had, I don't know, cucumbers instead of apples, then it would be cucumbers are a type of delicious vegetable, not fruit. So obviously, we need context about apples in there. So that vector, that dot product has to be non-zero, and the transformer will have to learn that over training to set up the matrices. Remember we talked about the Q matrix, the K matrix and the V matrix. You'll have to learn to set up those matrices in such a way that in this particular sentence, the dot product of the Q vector for delicious and the K vector for apples is going to be non-zero, so that then we can extract the context from the word apples for the word delicious.

01:10:36 So the dot products are going to be calculated in that way for each one of these combinations, Q delicious and then the K of each one of those words. And as a result, we're going to have six different dot products. Some of them are going to be very low, close to zero or zero. Some of them are going to be high. For example, in the case of apples, it should be quite high. Now we have these dot products. What we're going to do next is we're going to put them through a softmax function.

01:10:59 Just as a quick refresher, softmax effectively takes the exponent of a value and then divides by the sum of the exponents of all of these six values in this case. So basically, it'll give us a probability distribution and we'll have these six elements in this probability distribution. So for example, in the case of apples, it might be 80% are, maybe 5% A type of maybe ... Zero percent, whatever. So we get this probability distribution, and we'll use them as weights for the V vectors of the associated words. So for example, in the case of apples, if the dot product of apples and delicious vectors and then the softmax after it's applied, if that results in 80% or 0.8, then we are going to take that as the weight of the vector V for apple. So basically we're going to take a weighted sum of the V vectors, and the weighting is going to be the softmax of the dot products of the Q for the worded question, delicious and the K of each vector.

01:12:02 So those will be our weights, what we get from the softmax, and we're going to be taking a weighted sum of the V vectors because the V vectors, as we discussed, they contain what that word is bringing to the table, what that word is offering to other words as context. So in the case of delicious, we want the context from apples. We don't care about the context from a, are, a, of. We care about the context from type. And all of that will come as a result of ... Basically it will be evident in the transformer after the training, after lots and lots of training, which we'll talk about later on. We'll talk about training.

01:12:41 After lots and lots of training, we'll get a result where this contextual vector that we're creating as a weighted sum for the word delicious, it will have the right context from the right words that matter in this sentence. And yeah, that's a weighted sum of these V vectors. As you can see, it's a very elegant solution, very elegant mathematical solution to the logical problem that we had at the start.

Jon Krohn: 01:13:05 Got a bit of a dumb question here, Kirill. So everything that you're describing here, these Q, K and V vectors, they're helpful for allowing us to attend to the words that we already have in our sentence, and in particular, for these kinds of generative decoder-only approaches. I think it's always the case that we're only looking back because we're trying to predict what the next token, it's so-called auto-regressive model where we're predicting the next token, predicting the next word, which will mean that we're only ever looking back.

01:13:40 So we have these vectors, these Q, K and V vectors for some number of words looking back, whatever our size of our context window, which like with Claude 2 from Anthropic, it's like hundreds of thousands of tokens down in the context window. So yeah, we have these K, Q and V vectors for all these historical tokens, but how does that relate? And maybe this is what you're going to get to. Oh,

		this is totally where we're going to get to next. Oh yeah, perfect, perfect, perfect. I bet that's where we're going to go with this, is that ultimately, I guess what we're getting with all of these vectors for all of the preceding tokens, all the preceding words, this is going to allow us to predict what a good final word in the sentence, apples are a type of delicious blank. Right? When we get to floor five, we'll finally be able to have a prediction of what that word should be?
Kirill Eremenko:	01:14:34	Yes, yes, exactly. But you made a very, very good question. You asked a very good question about not looking forward, not looking ahead. I want to put a pin in that because I deliberately omitted talking about masking in this explanation of attention. We will introduce masking later on in this tutorial ... in this episode.
Jon Krohn:	01:15:01	It is a tutorial episode of the podcast, for sure.
Kirill Eremenko:	01:15:06	Effectively, yeah. So in this episode, we'll later on introduce masking, but for now, let's put a pin in that and let's just assume that this whole process is done equally for all of these vectors. Although you are right. When we're creating a context rich vector ... So this process, what I described is creating a context rich vector for the word delicious. So that weighted some of those V vectors that we just got to, that'll be the context rich vector for the word delicious. Same thing will be done for the word apple. Same thing will be done for the word are. So we'll end up with six separate context rich vectors, one for each word.
	01:15:44	And Jon made an absolutely correct comment that, for example, the context rich vector for the word type, which is the fourth word in our sentence, we will have to have a mechanism for it not to be able to look forward. So it's not going to see the fifth word of and it won't see the sixth word delicious, but we'll talk about that in a bit. So for

now, let's just imagine that for each one of these words, six words, we have a context rich vector. Great. So that's the end of floor three or box three. Now we're moving on to next floor, floor four, and things are going to get easier from here.

01:16:21 So in floor four, we have a straightforward feedforward neural network. And why do we need that in there? It's a two-layer feedforward neural network so you have these context-rich vectors. They go into a fully connected layer with an activated function. So the 512-dimensional vectors, they go into a fully connected layer. I think it's 2,048-dimensional hidden layer. And then after that, they go back through another fully connected layer without an activation function this time, they go into the output layer. So basically two fully connected layers or two layers, fully connected layers.

01:17:02 The first one has an activation function. That's the point of the feedforward neural network. First one, it adds more weights and biases, basically adds more parameters to the model so to increase its learning capacity, so it can model more complex relationships, and it also adds an activation function. So remember, up until this point, we haven't had any activation functions. We've only had linear transformations, matrix operations without any activation functions because that's how the transformer is structured. This is the first time in the neural network we're adding activation functions. So this will help it learn even better and more complex non-linear relationships in the data, in the linguistics.

01:17:44 And another thing to note, very important, this is not obvious, if you're learning transformers for the first time. This feedforward neural network is applied independently to each one of the vectors. So we have six context rich vectors that we've created, and they go through this neural network independently. They go in parallel of course, but they're not mixed and matched inside this network. The only time inside a transformer that vectors get to somewhat interact with each other is in the attention mechanism where the vectors can look up these V values from each other using the QK indexing mechanism. Apart from that, everything is completely separate. So each one of these six vectors goes through the feedforward neural network and comes out on the other side even more enriched. Their context enrichment hasn't changed because that was in the attention mechanism, but now they've gone through a feedforward neural network with more weights, with some activation function, so they're even more enhanced vectors.

Jon Krohn: 01:18:45 And provides more flexibility.

Kirill Eremenko: 01:18:47 Flexibility for what?

Jon Krohn: 01:18:48 In the whole thing that we're trying to accomplish by having-

Kirill Eremenko: 01:18:52 Oh, yes, learning-

Jon Krohn: 01:18:54 Yeah. By having a feedforward neural network anywhere when we're trying to map X to Y, a feedforward neural network is always just going to allow you to have some kinds of subtleties be handled, some kind of flexibility in learning that otherwise might not be possible. So it's cool. Yeah, it makes a lot of sense to have that in there.

Kirill Eremenko: 01:19:13 Yes, for sure. Okay. Floor five, the final floor. So what you have in floor five is ... So in the feedforward neural network, we had a 512-dimensional vector go through the first fully connected layer of activation function you get. In the paper, they say they have 2,048-dimensional, so it increases the dimensionality, and then the second fully connected layer reduces the dimensionality back to 512. So as an output of the fourth floor or fourth level and as the input of the fifth level, we're back to a 512-dimensional vector for each word. There's six of them in our sentence.

01:19:52 Now what we want is we want to map that to predict the next word. So we need that to map that to our vocabulary, which is, in the English language, let's say 200,000 words. So in the fifth level and final level, we have a linear transformation plus a softmax function. The linear transformation is designed to map the output that we have so far, the 512-dimensional vector to 200,000 words, to 200,000 nodes. Basically it's a neural network layer that has 512 in the input, no activation function, and it maps it to a 512 ... fully connected layer maps it to 200,000 layer with 200,000 nodes.

01:20:40 So we get those outputs, they're called logits. They're un-normalized, just outputs in this neural network layer. And then after that, we pass them through a softmax function, very similar as we discussed in the attention mechanism, softmax function, and this is our second softmax function for the transformer. Basically what it does is takes all those logits, those un-normalized values. We've got 200,000 of them, one for each word in the English language. Each one takes an exponent of each one and then divides by the sum of those exponents. So it basically creates a probability distribution.

01:21:15 Now we have 200,000 values in a probability distribution, which in total add up to one. And those values are representative of what is the probability for each word of the English language to be the next word in this sentence. But remember, this is key, this is very important part of transformers, something took me a while to figure out or wrap my head around. I had a big breakthrough myself literally yesterday on this. We had six vectors going into this fifth and final level and then we applied the linear transformation. It was applied to each vector independently, and we applied the softmax. It was applied to each of those vectors or whatever we got those 200,000. For each vector, we got 200,000 values after the linear transformation. And so the softmax was applied.

01:22:09 We applied it six times, so we have six probability distribution, one for the word apples, one for the word are, one as a result of the vector that we had for the word apples. We have a probability distribution of 200,000 values. Then we have another one as a result of the vector we had for the word are, another probability distribution of 200,000 values, and another one and another one, all the way up to delicious. We have for delicious. We had a vector which was then converted into a probability distribution of 200,000 values, which add up to one. So that's a very key important thing.

01:22:43 Now, in order to predict the next word in the sentence, we're going to take the probability distribution of the last word, of the word delicious, and we're going to use those probabilities, and we'll just take the word with the highest probability, and that will be our next word. So in this case, it will be fruit. We want the transformer to know that that's ... Once the transformer is trained, the word fruit will have the highest probability in the probability distribution that came from the context rich vector of the word delicious. The other five probability distributions, we're just going to throw them away. We don't need them. They're not used in inference. They're absolutely useless during inference.

Jon Krohn: 01:23:24 But they are useful for training.

Kirill Eremenko: 01:23:26 They are.

Jon Krohn: 01:23:26 Is that what we can-

Kirill Eremenko: 01:23:26 Yes.

Jon Krohn:	01:23:29	Okay. Okay. Why are we doing all that compute for nothing? Someone should have thought of this.
Kirill Eremenko:	01:23:30	I know, right? I know. So imagine if you have an input of a thousand words, like a context. That's why a context window ... It's hard because if you have a thousand words input, you're going to have a thousand probability distribution. You're only going to need the last one of them. Right?
Jon Krohn:	01:23:45	For inference.
Kirill Eremenko:	01:23:46	For inference, exactly. So we'll get to this in a second. So let's just sum up what we have so far, so the levels. On level one of our building, we have word embedding. We're turning words into vectors, which contain semantic meaning. On level two of our building, we have positional encoding. We're adding positional encoding to these word embeddings because transformers ingest all of the input at once, not sequentially. That's their disadvantage. But we're making up for it with positional encoding to let the transformer know in which order these words actually came in the original sentence.
	01:24:24	On level three, we have the heart of the transformer. You can think of it as like it's in the middle. The heart is in the middle. It's the self-attention mechanism. Well, actually, the correct term for it is the masked multi-head dot product self-attention. So we haven't talked about the masked part. We haven't talked about the multi-head part. We'll talk about those separately, but it's called the dot product. We know why. We've discussed why it's a dot product. It's dot product self-attention. It's attending to itself, to the sentence itself.
	01:24:51	So we've got the self-attention. And the reason why we need self-attention is we have vectors that have semantic meaning with positional encoding, but we actually want

vectors that also have contextual meaning because contextual meaning is extremely important in language. That is achieved through the Q vector to do the query and QK combination create the V lookup, if you're thinking of Excel terms or the indexing. If you're thinking of database terms like Q and K combination, it's the index, whether we want it or not. Well, K is the index, Q and K is the answer whether we want that or not in our context. And V is the value that we want to pull from each word into our context. So the Q, K, V vectors together work to create the self-attention. That's done through dot product, then a softmax, and then a weighted sum of V values. That's the self-attention part.

01:25:44 And then after that, on level four, we have the feedforward neural network, which has two fully connected layers. The first layer of the feedforward neural network is, in the original paper, four times larger so far from 512, it goes to 2,048 neurons. That's the first place and only place where there's an activation function. It adds flexibility, as Jon pointed out, to the learning. And then after that, it goes back from 2,048 back to 512 dimensions. The next layer, that's our feedforward neural network.

01:26:18 And then after that, on the final floor on level five, we have a linear transformation plus the softmax. Linear transformation is designed to go from 512 dimensions to the dimensionality of our vocabulary, for example, 200,000 words in the English language. And the softmax is designed to convert that into probability distribution. And all of this, very important to keep in mind always, all of this is done for every single word. So all the way, if we have six words, it's done for all six words. If we do a thousand words, we do it for all thousand words, all of these steps. Okay, that's the summary. All good?

Jon Krohn: 01:26:53 Yeah. Wow. This is, as you know, as we talked about at the beginning of the episode, I've now been the host for over 300 of these episodes, and we've definitely never had an episode this deep in the weeds. It is a record. It's a bit of an interesting experiment. There's probably going to be some people out there that are like, "Wow, this is the most valuable thing that's ever happened to me in a podcast. This is so rich." But as you said at the beginning of this section, when we got into the five boxes of transformers, this might also have been tricky for people to track, especially if they don't have a neural network background. But very well done, Kirill. Certainly for this podcast medium, you are stretching the boundaries of what's possible. And I think for some folks out there, it's going to be a slam dunk.

Kirill Eremenko: 01:27:41 Awesome. Thanks, man. Thanks. Yeah, and hopefully we'll do our best to include as many additional materials in the show notes so if people want some references. I'll add some slides from the course. I'll just take some screenshots from the slides from the course so we'll throw them in there as well so people can follow along as well in maybe some ... There'll be some visual aids. We'll add them in there. If you are struggling to keep up, check out the show notes for the podcast. This is really important stuff. I can't stress it enough. It's so important for your future career. If you understand all these concepts, they're not extremely complex. Once you dig into them and get your head around them, it'll be a game changer for your future career. As we discussed at the start of the episode, I'm glad we did. LLMs are here for at least the next few years, they're going to be disrupting businesses, industries. People are going to be building careers around them. They're going to be more and more pervasive. I don't know if that's the correct word, but this proliferation, there'll be a proliferation of it.

Jon Krohn: 01:28:47 Yeah. Pervasive is right too, for sure.

Kirill Eremenko: 01:28:49 Yeah.

Jon Krohn:	01:28:50	They will pervade more and more and more. Nice. So this could be a good point. If this has already been an intense amount, this could potentially be a good spot in the episode to take a pause and maybe simulate a bit of what has been covered because now we're going to expand on that. So we're going to expand on the foundations, these five boxes, these five levels of transformers that Kirill has gone over to talk about inference and training time, what we do at inference versus what we do at training time. And so yeah, let's dig into that. So we alluded to this a little bit. So talking about, for example, having all those probability distributions for each of the words, but at inference time, only needing the final ones. So maybe let's pick up from there.
Kirill Eremenko:	01:29:43	Okay. Yeah. So just quickly, we're going to start with inference and then we'll talk about training. With inference, indeed, you only need the last one. So let's say you have a prompt and with ChatGPT, you put in a thousand words of a prompt into ChatGPT, and what it will do is it will ingest that whole thousand. It'll go through a transformer. As a result, it'll have 1,000 of those probability distributions, one for every word, one as a result of the vector of each of those words. And we're only going to use the last one, so 999 of them are not going to be used. The last one, the thousandth probability distribution will tell us what the next word is supposed to be. This is because that context rich vector, which we had for the thousandth word, it had context about all of the words that came before it, so the probability distribution should be able to predict the next word. And that's exactly what we're going to do. We're going to pick the word of the English language with the highest probability from that probability distribution. That'll be our first or our thousandth and first word.
	01:30:45	Now, what ChatGPT does next? How does it generate a lot of text? How does it generate a whole paragraph of text?

Well, what it does next is it takes this 1,001 words now that we have, and it puts them through itself again. And as a result, we'll get 1,001 probability distributions. We're only going to use the last one, the 1,000 and first. We're going to discard the first thousand, and we're going to get the next word of the prediction, which is going to be 1000 and second word. Now it's going to take the 1,002 words that we have and ingest them again. And as a result, 1,002 distributions, we'll only take the last one and get the third word or 1000 and third word and so on.

01:31:25 So it will keep generating words like that all the way until it generates an end of sequence token, EOS. That token, the EOS token is also part of the vocabulary. It's one of those 200,000 words that we have. One of them is the end of sequence token. And when the probability for that token is the highest, it'll be selected as the next word in the sequence or in the output sequence, and that will be a signal to ChatGPT or whatever GPT model to stop basically generating there. So that's how inference happens.

01:31:59 And so when you see ChatGPT, when you ask a query and when you see ChatGPT entering one word at a time for you, outputting one word at a time, it's not just a cool thing they've done to make it look cool, like it's generating one word at a time. It's actually exactly how it works. Transformers take in your input all at once, but they generate one word, then they take all the input and generate. So every time, it's re-invoking itself until it gets to the end of sequence token.

01:32:27 And also, the other thing you might notice is when you're speaking with ChatGPT in the same chat for a while, it gets slower. I've had this experience a lot of times. I don't work for OpenAI so I can't comment on this exactly, but my guess is that what is happening is that it's taking all of your previous conversation and putting it in into itself.

So it's generated a paragraph of output. Now you ask another question. It's taking your question plus everything you had before up to the context window limit, which right now I believe is like 32,000 for ChatGPT. It's taking all of that, putting it back into itself, generating more and so on. So the more you talk with it in one chat, the more slow it will get until it reaches that context window, and then it's just going to be the same amount of slow going forward as I understand. So that's how inference works. That's how it generates new words.

01:33:17 I guess the only other comment here is that what we discussed here so far is called greedy sampling where you have this probability distribution, you take the word with the highest probability. There are other sampling techniques. For example, you could take the three words with the highest probabilities and then take a random one of those three words. And when you click in ChatGPT, you click regenerate output, that refresh button, it will give you a different answer, well, often, because it's not using greedy sampling. It's not always just taking the words with the highest probability. It's mixing it up a little bit to give some variability of the output. So that's inference. All good?

Jon Krohn: 01:33:55 Nice. And really quickly there, at the time of recording, the OpenAI model with the largest context window is GPT-4 turbo, also known as GPT-4.5, and that has 128,000 token context window. So that's about 300 pages of text that it can handle in a single round.

Kirill Eremenko: 01:34:17 Yeah, that's quite a massive context window. Okay. So now we're going to talk about training. Training is where transformers, where they are super efficient, the state of the art. As we discussed, they're not so efficient in ... What's it called? They're not so efficient in inference. They could be more efficient. They are redundant operations, but in training, they're the best. So what we do is we take a large corpus of text. Just think of, for example, all of Wikipedia, lots and lots of data, lots and lots of text in there. We take this text and we break it down into segments. Typically, segments are just basically chunks of text that are the size of the context window. So you just take all of Wikipedia, put into one huge document, and then you break it up into context window segments that fit exactly into the context window of the transformer.

01:35:15 Then each segment is fed into the transformer, one at a time for training. This, of course, can be parallelized. Now, we are going to look at what happens with one segment when it's being trained. For simplicity's sake, we're going to take an example, a smaller example. We're going to take the sentence we've been working with. We're going to take apples are a type of delicious fruit, which has a total of seven words, and we're going to pretend that's our segment and see how the transformer is trained. So you would think that what would happen in this kind of scenario is that you would take apples are a type of delicious blank. You would take out the last word. You would feed in everything except for the last word, and you would want the transformer to predict the seventh word. That's what we are naturally inclined to think.

01:36:01 Well, at least me with my limited exposure or the amounts of things I've seen about AI, I'm like, "Okay, that's probably the way it works." But actually it's much smarter that architecturally transformers blow everything out of the water. What happens is all seven words are fed into the transformer and then they all seven of them go through these five levels we talked about, or five levels of our building. So the token embedding, positional encoding, the self attention mechanism that feed forward neural network, and the final linear transformation plus softmax. And then we get these probabilities. For each one of these seven words, we have a probability distribution, which results from the vector that that word was represented by.

01:36:48 But let's go back to the softmax, and this is where we go back to the pin that we put in our conversation earlier, Jon. Basically the masking. So what will happen is the transformer has this additional mechanism inside the attention mechanism, which is called masking. And it doesn't allow any word to look at the words that come after it. So for example, in the case of apples are a type of delicious fruit, the word apples when it's doing the QKV mechanism, it'll only be allowed to look at itself. It can't look at any words going after it. The word are, so apples are, so the word are can look at itself and apples. But it cannot reference any of the words that go after it. The word type, which is the fourth word in our sentence, it can only look at apples are a type. So you can only get context. It's allowed to only get context from those words. It's not allowed to get context from further the words come later. And the apples... Yeah?

Jon Krohn: 01:37:54 Yeah a really key thing here though, just to reiterate for our listeners, is that this is for these decoder only models like the GBT series models. If you have some other architectures like the BERT model that we talked about now quite a long ways ago in the episode, that kind of encoder architecture to the B in BERT stands for bidirectional. And so, it can actually use information from the future of the sequence because with that kind of encoding model, we're not interested in predicting the next word. Which fundamentally you can't. If you knew what the next word was going to be when you're trying to do next word prediction, that's cheating. You don't need a model to do that. You just know what it is. But when you're doing an encoder type of transformer, that extra context of what's happening ahead of a given word can be useful and can be used for some downstream task.

Kirill Eremenko:	01:38:47	Yeah, for sure. And you're right, masking is only applied in the decoder part. In the original paper where you have encoder and decoder, the encoder it's used for translation. So the encoder can look at everything. When you're translating a sentence, you're able to see the whole sentence and then you generate word by word your translation. So in that case as well, there's no masking in the encoder. Good point. That's only a decoder thing. So you have this masking, so depending on the word position, all the other words to the right are the later words are masked. And it's also called triangular masking. The reason why it's called triangular masking is if you write out all these words in a matrix like as the columns and the rows. So you take apples are a type of delicious fruit, you write them as the rows and you write them out again as the columns, you'll have a matrix of seven by seven.
	01:39:39	And on the left the columns are going to be which word you are creating a context-rich vector for. And the columns represent which words that a context-rich vector can reference, is allowed to reference. Well, the way it works is the first row will only have the first column. Only apples, it can only reference itself. The second row can reference two words, apples are. The third row can reference three words, apples are a, and so on. So basically you'll notice that in the top right corner you have this triangle forming, which is those words are blacked out, they grayed out, they're not allowed to be referenced by the word that we are creating a context-rich vector for which is in that associated row. And so that's why it's called a triangular mask.
	01:40:32	And that's how we create these context-rich vectors. And the mathematical way that it works is it's very simple. It's basically once we calculate those dot products, and before you apply the softmax, for the words that you're not allowed to reference, you add this triangular mask. And it

basically where the words you are allowed to reference, the mask has a zero. Where you're not allowed to reference, it's a negative infinity. So effectively you're replacing the dot product. You calculate the dot product for each one of the words, but where you're not allowed to reference that word, that dot product is replaced with negative infinity. And when you apply the softmax, it will become a zero. So when you do the weighted sum, it'll be a zero. So it won't allow you to get to the V value behind the word that you're not allowed to reference.

01:41:24 So that's how the masking technique works. And effectively what we get in the end, so in our case, we have seven words. Apples are a type of delicious fruit. In the end, we will get after the fifth level, we will get a probability distribution from each one of these vectors. And if you think about it, because let's take the word for example apples, it was only able to reference itself. It didn't know in the attention mechanism, it didn't know what else is happening in the transformer on the next words. And moreover, as we discussed, the vectors in the transformer never get mixed between each other. The attention mechanism is the only time when the vectors can somewhat reference each other. So the word apples along the whole time, the vector for the word apples, has only been exposed to the vector for itself, the V vector, the KQV vectors for itself.

01:42:17 So effectively this probability distribution that we have from the word apples is a probability distribution as if our segment only had the word apples. And it predicts what the next word should be. And if we take the apples are, and we look at the probability distribution that we got from the context rich vector for the word are, that context rich vector was only allowed to reference the words apple, apples and are, itself. It wasn't allowed to reference anything else. So the probability distribution that we got from the word are is as if we were doing a segment with just two words, apples are. So it's predicting the next word. And then we keep doing that. So for example, apples are a type. For the probability distribution of the word type well, the vector that it was based on only ever referenced apples are a type.

01:43:07 It never saw what's coming next in the sentence. So it's as if that probability distribution would be the same if our whole segment was only apples are a type. So it's predicting the next word. So now all of a sudden, these seven probability distributions that we have, we in one go instead of one training, we can do seven trainings, we can calculate seven errors. Because the probability distribution that we got from the context rich vector of the word apples, should be predicting. We wanted to predict the word are. The probability distribution we got from the context rich vector of the word are should be, we wanted to predict the word a and so on and so on. So on the one we got from the word type, the fourth word, should be predicting the fifth word.

01:43:51 The one we got from delicious should be predicting fruit. The one we got from fruit is still useful to us because it should be predicting end of sequence. The next token after is end of sequence. And because that probability distribution has context of the whole sentence. All of a sudden we can calculate not one error, we can calculate seven errors and we can back calculate the loss function of seven errors back propagated through the whole neural network and adjust our weight. So this was a simple example. Now imagine you have, like we discussed at the beginning, you take the whole of Wikipedia and you break it out into segments which are exactly the size of the context window of for normal ChatGPT 4, 32,000. Now in one go the transformer calculated 32,000 probability distributions. And in one go of the transformer, you get to train it 32,000 times.

01:44:35 You get 32,000 errors. You calculate the loss function, you back propagate it. So effectively this is the mind-blowing power of transformers. You're getting within segment parallelization, you can parallelize it across batches. You can do a batch of segments at the same time across different machines. And then calculate the loss across batches and back propagate it that way. But also within each segment you are getting parallelization. So transformers, and this is extremely powerful insight. Transformers are super powerful, not only because they're efficient and matrix operations and they're elegant in their solutions, so on. But one of the biggest advantages is that they have this double parallelization within segment parallelization that we just described. And also the parallelization you can achieve through hardware, which is batch parallelization.

01:45:27 And through this double parallelization, that's why you can throw a lot of money at it, a lot of data. And you can train the more basically GPUs you can put towards it, the more money you can throw towards training it, the more you can get out of it because it just parallelizes so well. And this is a very inherent architectural by-design solution of transformers that we're taking advantage of these days.

Jon Krohn: 01:45:53 Nice. So now going back to my question from a while ago. When you were getting to the end of your five boxes of transformers, your five levels of transformers conversation. I was talking about, "Oh, why would we create all of these vectors for every one of the words, certainly for inference, it doesn't seem like we need it." And you agreed. Now the whole picture makes sense so that during training we're not just trying to predict some of the words. We use, all of the training, the data that we have. We might as well, like you were giving the apples are, apples are a type, apples are a type of delicious... You might as well... You have all these training data. And so you can use each one of these steps in the sequence of tokens.

01:46:45 So if you have all of Wikipedia to train on, then you might as well train at each next word in Wikipedia. And if you have all the internet to train on, then train on each next word on all of the internet, that's going to give you the most amount of training data possible. Very cool. And then so just one kind of question for you is, I guess the correct token... So in that seven word sentence that we've been working with throughout this episode, apples are a type of delicious fruit. I guess the correct token to be predicted after that is the end of sequence, that's special EOS token, yeah?

Kirill Eremenko: 01:47:26 Yep, exactly. Yeah, that's the correct token. Because that's our segment. There is a bit of additional tweaks that you need to do. If you're taking all of Wikipedia and you're breaking it down by context window, well your context window might just end up being in the middle of a sentence. So the end of sequence token might not be the correct one to be predicting afterwards. It's architectural choices, you might break it down differently. You might stop it a bit earlier before the 32,000 or the context window size add a full stop or at the end of the paragraph. Or you might choose not to use the prediction of the last token because you're not sure that that's the end of sequence token. So that's kind of like more detail nuanced things. But in a nutshell, yes, in our example, it would be the end of sequence.

01:48:17 And the other thing you pointed out there, which is very valid as well, very important. Is that by doing it this way, we're exposing the transformer to variable lengths of input. So we don't have to feed it five word sentences, a 100 word sentences, a 1,000 word sentences separately. We just give it the context of the maximum we can. And inherently by design is going to train on one word, sentences on two word sentences or three word sentences on four word, all the way up to your context window size. It's an additional automatic advantage of transformers that you don't have to think through how you're going to train them on different size of data. Again, you can make that more nuanced if you want to, but ultimately it's inherently built in that it's already training on all variability or all variations of input sizes.

Jon Krohn: 01:49:09 Nice. All right, so you've given us an amazing introduction to transformers, to attention and therefore to large language models. Why ultimately are transformers so powerful? Why when we take this attention mechanism that you've described, these five levels of the transformer, and we scale it up using all of the paralyzations that you recently described in this episode. Why is it the case that all of this ends up being able to be at this time of recording, by far the most nuanced mind-blowing tool that humans have ever created?

Kirill Eremenko: 01:49:55 Yeah, indeed. Indeed. It is really mind-blowing. It's very cool to watch talks of the creators of transformers. I found one of them on YouTube and it's really cool how to, these eight people, it's really cool to see how they were thinking when designing this. And each one of them had separate roles and they identified in the research paper. And by the way, the order of names on their research paper is not alphabetical or in order of contribution, it's at random because they all feel they equally contributed. And it's very interesting to observe their thinking about how they were building this architecture while having in mind speed. Speed and parallelization, all of the solutions they were coming up with, they were considering that. And I think that's probably one of the biggest reasons why transformers are so powerful because they use matrix operations which are really fast.

01:50:55 There's no like in LSTMs there's no gated units and all those kind of other solutions to the vanishing gradient problem, that had to be put into LSTMs. There's no bottleneck that LSTMs had. Transformers are non-sequential as opposed to LSTMs. So they don't take the input one by one. They can take the input all at once. So that's why if you put into a ChatGPT like, I don't know 2000 words of input, you get an answer right away. It starts typing. Your prompt is 2000 words, but you start seeing the results right away. Because it's not processing them one by one, it gets ingested all at once.

01:51:32 And of course it's double parallelization. I don't know if this is a completely correct technical term, it is something I used to describe it for myself. The parallelization within segment that happens through this wonderful just genius solution of triangular masking during the attention mechanism. And the inherent architecture of a transformer plus parallelization, which you can achieve through batches, which is hardware based. I think when you combine all of those things, it just like any model that is able to process all of the language that we have online on the internet, it's like a perfect storm. We have so much language that we've been creating since the '90s that's available on the internet. Any model that can process all of it, is going to be powerful. It just so happens that transformers were built with speed and parallelization in mind. And that made them the king in this moment.

Jon Krohn: 01:52:26 Nice summary. Very nice.

Kirill Eremenko: 01:52:28 Cool. Can we do a quick bonus trivia? I got a few extra points.

Jon Krohn: 01:52:31 Go for it Kirill.

Kirill Eremenko: 01:52:32 All right, so GPT model what we discussed those layers. Let's start with the attention head. So inside the attention block, remember how we said we create three vectors, the QKV vectors, there are 512 dimension each. So from the vector that has semantic meaning positional encoding, we create the Q and K and the V vector for each word. Well actually you can, that's one attention head. In reality the transformer, the original paper has eight attention heads. And how is that done? Well, the original vector that you have with the semantic meaning and positional encoding goes through a matrix or is transformed with a Q matrix.

01:53:14 But that matrix creates not a 512 dimensional vector, it creates a 64 dimensional vector. And there's a K matrix which creates a 64 dimensional K vector and there's a V vector matrix which creates a 64 dimensional vector. But then there's eight sets of those matrices. This is three matrix QKV 1, and then there's QKV 2, there's QKV 3, there's eight sets of those matrices. And so you get eight attention heads which are running in parallel. So the transformer can learn different things about the same sentence.

01:53:49 An important thing to keep in mind is that if I was doing an interview for somebody applying for an LLM job, I'd ask this question. When you're doing multiple attention heads is your input vector for this attention mechanism, which is the vector with the semantic meaning plus positional coding, is it split up into eight? Is it like cut into eight parts? The answer is no. It's not cut into eight parts. It goes through a matrix or linear transformation, a matrix operation which spits out the 64 dimensional vector of the result of the Q1 matrix, the 64 dimensional vector. It's actually a combination of all 512 dimensions. So you're not separating the dimensions of the input vector, you're combining them to a smaller output. So that's number one, multiple attention heads. The original

		paper has eight, you can have as many as you want. I believe the GPT-3 model had 96 of them.
	01:54:40	In addition to all of that, imagine all everything we did, we had a building of five floors. And on the level three we've just discussed, there's going to be eight attention heads or 96 attention heads. Great. But now take that building with five floors and put another one on top and another one on top and another one on top, 96 times. So the GPT model, it's again your choice how many you want to do. The original, the transformer architecture has six layers, so these are layers. So after the output of these five levels, instead of predicting the next word, those outputted vectors that we got the context rich vectors after the feed forward neural network. They go as inputs into the next level, into the level one, of the next building.
	01:55:24	And then we get outputs again and then go in. So that happens 96 times in the GPT-3 model. And why is that? Well, that gives it even more opportunity to learn, to learn more complex relationships in the language. Understand more complex concepts like how can a ChatGPT answer questions on chemistry. That's a level of sophistication, or poetry. That's a level of sophistication that might not be achievable with one layer of one decoder. But when you put 96 decoders on top of each other, well it looks like it's able to do really phenomenal things.
Jon Krohn:	01:56:01	So making sure I'm getting this right. So 96, you said that number twice for GPT-3. So there's both 96 attention heads?
Kirill Eremenko:	01:56:12	In each decoder.
Jon Krohn:	01:56:13	As well as in each decoder. As well as 96 layers of decoders.

Kirill Eremenko:	01:56:16	Yes. So there's 96 squared attention heads if you calculate all of them correctly.
Jon Krohn:	01:56:21	Wow.
Kirill Eremenko:	01:56:22	That's not the original design. And with new models, things might be changing and I would urge people to double check this number. But the original design was six layers of encoder decoder architecture with eight attention heads inside each layer. In GPT, in ChatGPT specifically, they took it to the next level with 96 layers. And as far as I remember, 96 attention heads inside each layer. And I guess the other part of bonus trivia is we already talked about that the original paper was written for translation tasks. GPT is just the decoder part, which is for generation. And of course like GPT can also do translation, interestingly enough. But we're not going to go into detail on that. The other thing I wanted to point out is, for me diving deep into this really helped to understand that ChatGPT doesn't have any reasoning or logic behind it. It doesn't understand and cannot think. It cannot answer your questions through thought. It can only answer your questions by predicting the next token, but it still stands to up to debate.
	01:57:39	It's good to understand, it helps me better perceive these large-language models and what they're doing. That they're not going to, on one hand, they're not going to take over the world because they can't think for themselves. They can't have a reasoning. On the other hand, like Ilya Sutskever who is, who's in OpenAI and who was part of these research papers that we talked about at the start. There's a YouTube video of him talking to an interviewer about this. And the question is, maybe this is a new kind of intelligence where you don't need reasoning. But predicting the next word is sufficiently sufficient in order to behave like a human, in order to think like a human, in order to create things like a

human. And so it still stands up for debate, but at least knowing the difference in how we think and how we reason versus how a GPT or a large-language model reasons for me, it's been really helpful.

Jon Krohn:	01:58:41	Very cool. Very cool, man. What an episode. It has been epic. Truly.
Kirill Eremenko:	01:58:48	Thanks man.
Jon Krohn:	01:58:49	And I've learned a ton. Kirill, I can't believe how much you and probably Hadelin now know about attention, about transformers, about large-language models. Wow. And so yeah, your course must be incredible. I am now itching to take it. So yeah, Large language models A-Z available in the superdatascience.com platform exclusively.
Kirill Eremenko:	01:59:15	Exclusively.
Jon Krohn:	01:59:16	Fantastic.
Kirill Eremenko:	01:59:17	Check out-
Jon Krohn:	01:59:18	Kirill, before I let you-
Kirill Eremenko:	01:59:18	Oh yeah, sorry Jon. Just quickly check out superdatascience.com/LLMcourse. You'll see there the curriculum, the description, some images, so it just gives you a quick overview before you decide if it's the right thing for you.
Jon Krohn:	01:59:34	Nice. And of course, we'll be sure to have that link in the show notes. Kirill, before I let you go, as you know having created that document for me to interview people from. Before I let guests go, we must have a book recommendation. Do you have a new one for us since you were last on the show last year?

Kirill Eremenko:	01:59:52	I'm not sure if it's a new one. I may have mentioned it on the show before, but I think it's fair for me to mention it because I am rereading this book myself for the second time. It's not a technical book. Hadelin and I first time listened to it in a road trip actually through Slovenia. We were on a road trip with Ivana and her husband, Mitja. Ivana is one of the people who contributes to this podcast.
Jon Krohn:	02:00:17	Yeah, Ivana is the podcast manager. She is the most important person on the show. She is the one, two episodes a week. They don't happen every single week at exactly the right time for many years without an unbelievably diligent person. And yes, Ivana-
Kirill Eremenko:	02:00:31	Absolutely.
Jon Krohn:	02:00:32	Ivana in Slovenia is that person.
Kirill Eremenko:	02:00:33	So yeah, we were on a road trip there with Hadelin. And that's when we listened to it the first time. And recently I felt the need, the time, I felt the time came to re-listen to this book. The book is called The Big Leap by Gay Hendricks. It's over 20 years old. It's a fantastic book that the main two kind of takeaways is where are you operating in your life. Whether it's your personal life, your romantic relationships, your work life. Are you operating in a zone of incompetence, a zone of competence, a zone of expertise, or a zone of genius? And he talks about how to identify that zone of genius for yourself in all these areas of your life. And why it's so important and also what prevents us.
	02:01:14	And the second takeaway, the big takeaway, that I remember from back in 2017 is the thermostat principle that whenever things are going well in life, like your body or your mind operates like a thermostat. Like if you're used to a certain level of happiness or finance or whatever

		else, romance in your life, and things go worse, you will find ways to get back to where you were. But interestingly conversely, if things get better, you'll also push yourself back to where you were. And that's why, for example, we see the stats around lottery winners. More than 80% of them lose all their money within two years. Because their psychology takes them back to where they used to be.
	02:01:56	And I think I can tie it into the times we live in entering into this new age of where there's going to be a lot of change happening with 2024 bringing in even more probably AI developments. I think it's extremely important to be conscious of your psychology. And to enhance your psychology because as they say, success, however you define it, however you measure, it's only 20% mechanics. It's 80% psychology. So it's really important. I feel in my life, I am ready to take the next step in many levels, in many layers. And I feel like that's why I felt the need to reread this book again, to help me prepare my psychology for that step.
Jon Krohn:	02:02:39	Your life has many levels, many layers, and many attention heads.
Kirill Eremenko:	02:02:43	96.
Jon Krohn:	02:02:46	Squared. Awesome, Kirill. So yeah, thanks so much. If our listeners aren't already aware from your many previous appearances on the show and your countless courses and everything else you've done online for millions of people. How should people be following you if they are new to you?
Kirill Eremenko:	02:03:09	Great. LinkedIn, great place to follow if you like. And also SuperDataScience membership. We've just revamped it and we've got a community which introduce community aspect. And so I'm hanging out there probably twice a week at the moment. I'm going to be hanging out there

		almost on a daily basis. You can hit me up, chat. We will be having these interactive sessions, get togethers, parties, whatever else, workshops. Definitely if you want to interact in person, that's the place you can find me.
Jon Krohn:	02:03:41	Nice. In person, you mean live?
Kirill Eremenko:	02:03:43	I mean live. Sorry. Live. It's becoming-
Jon Krohn:	02:03:47	Virtual.
Kirill Eremenko:	02:03:47	So normal. Virtually-
Jon Krohn:	02:03:49	Virtually in person. Nice. All right thanks so much, Kirill. It's been so awesome catching up with you and having all of these insights, these deep, deep insights that you and Hadelin have dug up about LLMs, about transformers, about attention. Thank you so much. And I'm sure we'll be catching you again sometime soon.
Kirill Eremenko:	02:04:10	Absolutely. Thanks a lot, Jon. I really loved it.
Jon Krohn:	02:04:18	Well, I hope you enjoy that extra technical episode. And Kirill filled us in on the development of attention and natural language processing, the five stages of transformer data processing, specifically input embedding, positional, encoding the attention mechanism, a feed forward neural network, and linear transformation and softmax. He talked about how encoder-only transformer architectures like BERT are efficient for natural language understanding tasks. And how decoder-only architectures like the GPT family are best suited to generative tasks. He also filled us in on how transformers are used for both training and for inference and how they're scaled up to enable the powerful emergent capabilities of LLMs.

02:04:57 As always, you can get all the show notes including the transcript for this episode, the video recording, any materials mentioned on the show, the URLs for Kirill's social media profiles, as well as my own at superdatascience.com/747. Given how unusual and highly technical today's episode was, I'd particularly appreciate if you reached out to me to let me know. So perhaps by commenting on the social media posts that I make about today's episode, to let me know what you thought about having so much technical content in an episode. We're definitely trying something different and so let us know whether we should do it again, or maybe not.

02:05:34 All right, so thanks of course to Ivana, Mario, Natalie, Serg, Sylvia, Zara, and Kirill on the SuperDataScience team for producing another exceptionally deep episode for us today. For enabling that super team to create this free podcast for you, we are deeply grateful to our sponsors. Please consider supporting the show by checking out our sponsors' links, which are in the show notes.

02:05:55 And if you yourself are interested in sponsoring an episode, you can get the details on how at jonkrohn.com/podcast. Otherwise, please share this podcast, review it on your favorite podcasting platforms. Subscribe if you aren't a subscriber already. But most importantly, just keep on tuning in. I'm so grateful to have you listening and I hope I can make episodes you love for years and years to come. Until next time, keep on rocking it out there and I'm looking forward to enjoying another round of the SuperDataScience podcast with you very soon.