SDS PODCAST EPISODE 759: FULL ENCODER-DECODER TRANSFORMERS FULLY EXPLAINED, WITH KIRILL EREMENKO

Show Notes: http://www.superdatascience.com/759

Jon Krohn: 00:00:03 This is episode number 759, with Kirill Eremenko, the founder and CEO of SuperDataScience. Today's episode is brought to you by Ready Tensor, where innovation meets reproducibility, by Oracle NetSuite Business Software and by Intel and HPE Ezmeral Software.

00:00:20 Welcome to the Super Data Science 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 Super Data Science podcast. Today, we've got another special episode with an extremely important individual to this show, Mr. Kirill Eremenko. If you don't already know him, Kirill is founder and CEO of Super Data Science, an e-learning platform that is the namesake of this very podcast.

00:01:07 He founded the Super Data Science podcast in 2016, and he hosted the show until he passed me the reins a little over three years ago. He has reached more than 2.7 million students through the courses he's published on Udemy, making him Udemy's most popular data science instructor.

00:01:23 We've made today's episode for you because, well, you demanded it. Kirill was most recently on the show for episode number 747, in which he provided a technical introduction to the transformer module that underpins all the major modern, large language models, like the GPT, Gemini, Llama, and BERT architectures. We received an unprecedented amount of positive feedback from that episode demanding more, so here we are. That episode, number 747, as well as today's, are perhaps the two most technical episodes of this podcast ever. So they probably appeal mostly to hands-on practitioners, like data

scientists and machine learning engineers, particularly those who already have some understanding of deep neural networks. In this episode, Kirill reviews the key transformer theory that we covered back in episode number 747, namely, the individual neural network components of the decoder-only architecture that prevails in generative large language models, like the GPT series models.

00:02:16 Kirill then builds on that to detail the full encoder-decoder transformer architecture that is used in the original transformer by Google in a paper called Attention Is All You Need. And it's also used in other models that excel at both natural language understanding and generation, such as T5 and BART. And then he goes on to discuss the performance and capability pros and cons of full encoder-decoder architectures relative to decoder-only architectures like GPT, and encoder-only architectures like BERT, not to be confused with the encoder-decoder BART. And then at the end, we talk about how you can learn everything on transformers and LLMs from today's episode with accompanying video instruction. So not just this audio-only format, by heading to Kirill's LLM course, which is at superdatascience.com/llmcourse. All right, you ready for this insane episode? Let's go.

00:03:16 Kirill Eremenko, welcome back to the Super Data Science podcast. I know where you are 'cause you're in the same office that you're usually recording in. Well, although sometimes you record from home.

Kirill Eremenko: 00:03:27 Yes, that's true.

Jon Krohn: 00:03:28 So, I know that you're in Brisbane, but maybe you can tell us, so since telling us where you are isn't going to be that interesting or different, maybe tell us why you sometimes record from the office versus from home.

Kirill Eremenko: 00:03:40 Well, first of all, thank you, Jon, for having me again, within the span of 40 days. I'm actually in Gold Coast, which is near Brisbane, it's about an hour away from Brisbane. And sometimes office, sometimes home. Good question. I guess it depends on the day. Air con is off in the office on weekends, so I really need the air con. But otherwise also, home is generally a little bit tidier behind me in the background, so it [inaudible 00:04:16] easier.

Jon Krohn: 00:04:16 That's true. That's funny. Last time, before we started recording, I was like, "Maybe just move like that thing and that thing, those are out of control," but I think it's authentic, which is a big part of who you are.

Kirill Eremenko: 00:04:28 Thanks.

Jon Krohn: 00:04:28 You are as authentic as they come, and so it's nice to see whatever you have in there in your office behind you.

Kirill Eremenko: 00:04:35 Thanks. What about you? You have a table? That's a new table?

Jon Krohn: 00:04:39 Yeah, that's different. People watching the YouTube version get to see a table behind me. Wow. So sometimes, Natalie, who is operations manager for the podcast, when we have lots going on, this is a table that's usually in my living room, but I bring it in so that there's lots of space to be working collaboratively when there's lots of admin items to get through. So that's the plan. We're recording on a Monday and it was a weekend of intensive admin and Monday evening is going to be more of it. So yeah, it's in here. It's weird. It's funny that you even notice it. It's white. Maybe just people wouldn't even see it. Anyway, this isn't very good banter. This is low quality banter. Apologies to all our listeners.

00:05:25 So yeah, you've been on the episode. This is your show, for people who don't know, the Super Data Science

podcast, founded by Kirill Eremenko. The first 400 or so episodes were hosted by you, and you have been my guest since I became the host over three years ago now since I've been hosting. And most recently, you were on the show for episode number 747, and in that, you gave an introduction to large language models. And as we were recording that, we took sometimes some breaks and I said to you, "This is getting out of hand, man. This is too technical. This is too long. You've gone too far."

Kirill Eremenko: 00:06:11 People are going to hate it. I was really worried people would tune out and not enjoy the episode as much.

Jon Krohn: 00:06:16 And it was the opposite, opposite scenario. It ended up being that people loved it. We got more individual feedback on the episode than I can recall getting on any other episode. We don't have a rigorous quantitative evaluation of that. And so there could be a recency bias of memory in that, but I'm pretty sure that we've never had so much positive feedback. And certainly, what I can tell you quantitatively, is that it was an absolute barn burner in terms of listens. So, we internally, our key metric of performance on episodes is how the episode has performed after 30 days, so that gives us a snapshot that we can compare. As soon as 30 days have passed, you have this single metric that you can track across all episodes. Because of course sometime, obviously if episodes that were published many years ago, they otherwise have an advantage over recently published episodes, but if we use that 30 day mark. And yeah, it was one of the most popular episodes ever at that 30 day mark. So, seemed to be right on.

Kirill Eremenko: 00:07:18 Great. Great. Yeah, and it was a different format as well, very long, very technical, and I'm glad people enjoyed it. We also noticed quite a lot of, I noticed quite a lot of interest. I saw those comments on the YouTube version

and the eight likes on the SoundCloud version of the episode, which was very nice.

Jon Krohn: 00:07:41 Well, yeah, because often SoundCloud, you don't get much engagement there-

Kirill Eremenko: 00:07:44 Yeah, that's true.

Jon Krohn: 00:07:45 ... is the point you're making. Yeah.

Kirill Eremenko: 00:07:45 That's true. And a lot of people came to the SuperDataScience community afterwards, because on the episode we mentioned SuperDataScience Large Language Models A-Z course, and people came to check out the course to sign up. And I guess slowly leading into today's episode, what we noticed, what I noticed what people were coming for was, yes, of course the Large Language Models A-Z course, where we talked about the whole LLMs and transformer in detail. But interestingly, and we actually ran an event, a live event where I was like, "Okay, I'll answer any transformer questions, any technical questions you have." Because a lot of people watched the episode, had taken the course by that time, and I said, "Come, bring your technical questions."

00:08:28 And so these people came to this live event, which was online of course, for an hour. And I was expecting all these super technical questions about attention mechanism or input embedding and positional encoding, things like that. And most of the questions, I was so surprised, most of the questions people were asking about business applications, how to use LLMs in their specific challenges at work. I think it was zero technical questions. People were like, "All right, how do I use LLMs? What are the concerns? How do I fine-tune models, how to apply it in my specific business situation and so on?" And then also, I had at least one person very thoroughly asking in the community about the full transformer

architecture. Because on the podcast we only talked about the large language model part, which is the decoder-only architecture, and there was at least one person who was very curious about the full transformer architecture.

00:09:34 And so that's what we're going to focus on today. We'll do our best to cover off as much as we can in the time that we have. We'll focus on the full transformer architecture for the first part, which is going to be very technical. Then we're going to focus on fine-tuning and understand what fine-tuning is, and the different types of fine-tuning they are, the different outcomes that you might want to get from fine-tuning. And how to do fine-tuning, the different methods or techniques there are for fine-tuning. And then in the final part, again, if we have time, we'll talk about different additional technical considerations. So things that I didn't mention in the previous podcast episode or in this one, that extra super technical stuff for people who really want to dive deep into it. So, first part is going to be technical, middle part is going to be slightly less technical, final part, if we have time, will be super technical. That's the plan. Yeah.

Jon Krohn: 00:10:34 Research projects in machine learning and data science are becoming increasingly complex, and reproducibility is a major concern. Enter Ready Tensor: a ground-breaking platform developed specifically to meet the needs of AI researchers. With Ready Tensor, you gain more than just scalable computing, storage, model and data versioning, and automated experiment tracking. You also get advanced collaboration tools to share your research conveniently and securely with other researchers and the community. See why top AI researchers are joining Ready Tensor - a platform where research innovation meets reproducibility. Discover more at readytensor.ai. That's readytensor.ai.

	00:11:16	And we'll see, even this full transformer architecture thing, there's a chance that it's going to end up being a huge amount of time, in which case we'll figure something out. We will get all that content to you in some format or another. We'll try to get it all into this episode. And in case it isn't obvious, we are going to do a quick recap of what was covered in Kirill's preceding episode. So in episode number 747, that extremely in-depth introduction to transformers. We're going to be building on that right now. Well, you're going to get the recap and then we're going to build on it, but if you want the full recap, go back to that episode number 747 from last month.
Kirill Eremenko:	00:12:05	For sure. That would be very helpful. But yeah, let's dive into it. So, starting with the recap of what we discussed last time, we used an analogy. So, by the way, people commented, after the previous episode, people commented that it was very helpful, if you're in front of a computer, to open the research paper titled Attention Is All You Need from 2017, you can find it on arXiv. Just type in Attention Is All You Need research paper, it'll come up. It's a foundational paper for large language models. It was published by a team at Google in 2017. Jon, I checked this out recently, it's got over 100,000 citations. That is extremely high. Even 10,000 is huge for a niche research paper. This one's got 100-
Jon Krohn:	00:12:51	10,000 is insane.
Kirill Eremenko:	00:12:52	This one's got 100,000.
Jon Krohn:	00:12:53	10,000 is insane. I didn't know that. That's absolutely crazy.
Kirill Eremenko:	00:12:57	Yeah.

Jon Krohn:	00:12:57	And that's Vaswani et al. Although interestingly, something I learned from you, in the episode 747, was that all six authors contributed equally, which is also very unusual. So typically-
Kirill Eremenko:	00:13:12	Eight authors.
Jon Krohn:	00:13:13	Eight authors?
Kirill Eremenko:	00:13:14	Yeah.
Jon Krohn:	00:13:14	Oh my goodness. Yeah, so typically, the first author does the most work. They do the writing, it's kind of their idea. Every once in a while, you see on papers an asterisk next to the first two authors that says, "These two authors contributed equally to this work," but in this case, it's all eight who all brought different specializations. That's really cool.
Kirill Eremenko:	00:13:34	And it's really interesting to read, literally on the first page of the paper it says how they contributed. "This person did this, this person added or find tuned all our models. This person added this kind of speeding up methodology. This person introduced this solution." And then there's at least one YouTube video I watched with one of the researchers and how he talks about the contributions of some other people. And it was interesting how they brought their different perspectives and skills into this paper. So yeah, what I was saying is, if you're in front of a computer, bring up the research paper and just go to the diagram. If you're not in front of the computer, that's totally fine. We'll visualize it in our minds and then you can look at it later on when you get to a computer and then you'll be like, "Oh, that makes sense. That's what they were talking about," how it exactly looks in the research paper.

	00:14:27	Cool. So, we use the analogy of a five-story building, and that's the decoder. So the transformer has two parts. It's got the encoder, which is on the left, and it's got a decoder, which is on the right. For visualization purposes, just imagine left and the right. So we told-
Jon Krohn:	00:14:42	It-
Kirill Eremenko:	00:14:43	Yeah, go ahead.
Jon Krohn:	00:14:44	It doesn't actually, to clarify there, there's not literally a left and right. That doesn't [inaudible 00:14:49]-
Kirill Eremenko:	00:14:48	It's just on the picture.
Jon Krohn:	00:14:48	It's just different functions. Right?
Kirill Eremenko:	00:14:50	Yeah, yeah.
Jon Krohn:	00:14:51	On the picture, in that paper. And so therefore, what a lot of people use when they are representing it. Although also, 'cause we often think, at least in the writing systems that we use in the West, they go from left to right. And so from that perspective, maybe it also makes a little bit of sense, because if you think about an encoder and a decoder together, it's easier to kind of conceptualize, and that's probably how you're going to run through it, the encoder working before the decoder. So kind of information moving left to right.
Kirill Eremenko:	00:15:17	I think that stems from the seq2seq models. Remember the RNN or LSTM-based models, and they were pictured as a sequence of boxes going from left to right, like a train, as in the words that you write. As you said, we write from left to right. That's how the words stem. That encoder was on the left. It was like a flat horizontal structure, and then feeding into the decoder on the right or another flat horizontal structure. Whereas with the

transformer, now the structures are vertical, hence our analogy of buildings, but yeah, they kept the positioning, I guess. So previously, we talked about the part on the right, which is the decoder, and we discussed that it has five levels. So we're going to recap that and then we'll see what happens when we add the part on the left, which is the encoder.

00:16:04 So, decoder, if we start from the bottom, the very bottom level, if you imagine a five-story building, and you're inputting words into this, you're inputting your prompt into a large language model. You're asking it, "What is the tallest mountain?" You put that in. So it all goes into this first level of the building, and the key thing is transformers process inputs, not sequentially, but in parallel. So all those words go in at the same time. And the first level, each of those words gets a input embedding. So an input embedding is a vector. A large language model is neural networks, and neural networks can't work with words, they need to work with numbers. So, we're representing each word with a vector. And these are not just random vectors, they're vectors that have semantic meaning. Semantic meaning is dictionary meaning of the word. So, words that have similar semantic meaning are going to be similar, words that have different semantic meaning are going to be far away from each other.

00:17:00 So for example, an orange is going to be closer to an apple and a banana, than it's going to be to the word car or airplane, or the verb to run, or the adjective beautiful. So all words will get some encoding. And all of these vectors, there are 512 dimensional, because the more dimensions you have, the better you can describe a word. So yeah, that's effectively level one. Right? Anything to add? All good? Just a quick recap.

Jon Krohn:	00:17:27	No, all good, man. Yeah. Oh, yeah. Oh, yeah. You got this down. Yeah.
Kirill Eremenko:	00:17:31	Okay, level two. So now we've converted our words into vectors, and there's many ways of doing this. There's a bag-of-words model, there's the n-gram model and so on. So you can read up about those. That's not the core of this transformer or large language model. It's just like a method that we have to do. Then after that, level two is the next module, which is called the positional encoding module. Now, as mentioned before, all of these words like, "What is the tallest mountain?" They go in at the same time. By the way, quick caveat, as in the previous podcast, we're talking about words, but actually they get broken down into what are tokens, and tokens a bit smaller than words. We're not going to go into detail of that, so we're going to use tokens and words in to chain in this podcast. So, tokens go into level one, they get these vectors, then they go up to level two. And in level two... Oh, you want to say something, Jon?
Jon Krohn:	00:18:26	Well, I should just really quickly say that usually they're smaller than words. 'Cause tokens, you could have word-level tokens, you could have character-level tokens. What's on Vogue today, in which we talked about a lot more in the preceding episode, is subword tokens, which will typically be parts of a word. So if you have a very long word, it gets broken up into several of these subwords. If you have a short word, it might be the same. So yeah, so generally speaking, yeah, exactly, exactly. And I don't mean to nitpick.
Kirill Eremenko:	00:18:56	No, no, no. That's very useful. I love learning something new. I didn't know that tokens could be bigger than words, so that's-

Jon Krohn: 00:19:02 Yeah, 'cause you could even, theoretically, you could have a sentence-level token. You could talk about breaking up a document into, it's kind of like a definitional thing.

Kirill Eremenko: 00:19:12 Yeah, yeah, yeah, cool. Very cool.

Jon Krohn: 00:19:15 Yeah, yeah. But yeah, subword tokens, that's definitely the way things are done today.

Kirill Eremenko: 00:19:18 So again, we have these words, they get broken down in tokens. We're going to forget about tokens, we're just going to use words for now. They go into this level one, they get their individual vectors, which represent their dictionary or semantic meaning. Then from there, they go to level two. On level two, they get a positional encoding. What is a positional encoding? Well, as discussed, all of these words go in at the same time. Previous models, such as the RNN-based, or recurrent neural network models, or even more specifically the LSTM models, they would take in the input one word at a time, so they would inherently know the order in which the words came. Transformers are very efficient at training, at processing data, because they can take input in parallel. So you imagine a whole page or 100 pages go in at the same time, the transformer.

00:20:03 That's why when you ask ChatGPT a question, you put in maybe a whole page of text, it instantly gives you the answer, 'cause it doesn't need to process every word by itself. It processes everything at the same time, but the drawback is that now it doesn't know in which order the words came. So we have to have this level two module where we add a positional encoding mechanism. So for example, we looked at the example in the previous podcast where we said the, oh wait, what are they called? Horses eat apples. And then if you reorder the words and you get apples eat horses, it's grammatically correct, but it's a completely different meaning.

Jon Krohn:	00:20:41	Yeah. Do you remember that while you were talking about that?
Kirill Eremenko:	00:20:44	Yeah.
Jon Krohn:	00:20:46	Maybe I could share them-
Kirill Eremenko:	00:20:47	Sure. Yeah, yeah, yeah.
Jon Krohn:	00:20:48	... for accessing in the show notes, but I wanted to see if... Because it's so unusual to think about apples eating horses, it sounds like it's grammatically nonsense. But I wondered whether if I went into Dall-E 3 in the ChatGPT interface and asked it to create apples eating horses, I thought that maybe it would be such grammatical nonsense, so far out of sample of the training data, that it wouldn't be able to do it, but it did it perfectly.
	00:21:21	And then I sent, in real-time, as we were recording the episode, I Slacked images made by Dall-E 3 of apples eating horses. So, it is really amazing to me how the most modern LLMs that we have at the time of recording are able to, in many cases, and there are constraints where you try to go too far out of sample and it won't be able to figure that out and create the sentence for you or create the image for you, but they're starting to get, with these billions of parameters, they're starting to get unbelievably flexible, such that even this grammatical nonsense, no problem.
Kirill Eremenko:	00:21:59	For sure. That was really cool. I tried creating an image, I think yesterday of, what was it? It was something to do... Oh, how in the quantization, a normal distribution gets converted into quantized... What's it called? Quantized number type. This is for quantization for neural networks. Came up with the craziest picture ever. It was similar to your apples eating horses image. So yeah. But at least it gave it a shot.

Jon Krohn: 00:22:40 Yeah. Anyway, I derail you. You're talking about horses eating apples as being grammatically sensible, and apples eating horses as being like-

Kirill Eremenko: 00:22:48 Nonsense. Yeah.

Jon Krohn: 00:22:50 Nonsense. Nonsense [inaudible 00:22:51].

Kirill Eremenko: 00:22:50 And so basically, if you move the words around in a sentence, the meaning changes, so we have to preserve positional... We have to know what position the words came in, and we have to communicate that to a transformer, and that's what module two does. There's lots of ways of doing that. There's a very elegant solution that's used in the transformer, or in this decoder part that we're talking about it. It's uses cosine and sine functions. We're not going to go into detail on that. It's another technical topic, but it's outside of the core value of what transformers bring to the table, which is attention, and that's the next level. So, after these vectors, after the words get the vectors and the vectors get positional encoding added to them, they go into part three, which is the self-attention mechanism. And the self-attention mechanism, basically from each one of those vectors creates three vectors, the Q vector, the K vector, and the V vector, the query vector, the key vector, and the value vector.

00:23:48 So, the query vector is the vector that looks for something. Let's rewind a little bit. Let's say, "Why do we need this attention mechanism?" Let's start it there. So attention mechanism allows us to add context to the words that we're using, to encapsulate context. Example that we used previously was, the dog did not cross the street because it was too tired. So the word it refers to the dog because the dog was too tired. Now, if we change the last word in the sentence, the dog didn't cross the street because it was too wide, the word it refers to the street

because the street was too wide. So we can see that context of a sentence can change the meaning of individual words. And what that tells us, is that words don't only have dictionary meaning, which is a semantic meaning. They also have contextual meaning. And the huge advancement of transformers, compared to other previous models, is that they're able to capture this context.

00:24:49 Well, to be fair, there was a paper before the transformers, where attention was introduced by Dzmitry Bahdanau with Yoshua Bengio as his supervisor. And Yoshua Bengio actually came up with the term attention. We were speaking about Yoshio, really want to get him on the show, invite him onto the show. And we've mentioned him a couple of times, would be great to get him onto the show.

Jon Krohn: 00:25:10 Yeah, I think we're getting closer and closer. I think it might happen soon.

Kirill Eremenko: 00:25:14 That'd be awesome. So anyway, so that attention concept was introduced previously, but transformers really take advantage of it in a beautiful way. But what attention does, is it allows to capture that contextual meaning of words. And if you don't capture contextual meaning, if you just capture dictionary meaning, then we're back to 2015 model, 2016 models. All that RNN, LSTM models, they're pretty good, but they can't put together a long enough sentence because they lose that contextual thread. And so this attention mechanism is designed to capture context, as we saw it's important. How do they do it? Every word gets three, instead of one vector, which we had already, it gets three vectors. From that one vector, we create three vector...

00:26:03 Already it gets three vectors. From that one vector, we create three vectors. We get Q vector, K vector, and a V

vector. And let's say we're looking at the sentence, apples are a type of delicious blank. So for every word we're going to create a contextual vector. How do we do that? Well, for example, for the word delicious, we go and take the Q vector of the word delicious, which will contain what the word delicious is looking for. Then it will go and interact with every K vector of every other word, including itself. So it will say, "Okay, this is my Q vector of the word delicious. What's the K vector of the word apples? What's the K vector of the word are? What's the K vector of the word a? Delicious? Fruit?" And so on.

00:26:44 So it'll interact with each one. Interact as in we will compare the Q vectors to the K vector. That comparison is done through a dot product operation. And if two vectors are aligned, their dot product is high. If two vectors are perpendicular at zero, if they're not aligned, it's very low. So, we do that process and through that process we know which vectors, QK pairs are aligned, which QK pairs are not aligned, and the ones that are aligned, that's where we go to that word and we take the V value from that word. So the Q vector is what I'm looking for is the word delicious. The K vector is what every other word including myself has to offer. And the V value is what it actually offers, what context it offers to other words. So, as the word delicious, I'm going to pick the words where my K vector is aligned. There K vector is aligned with my Q vector, and from those words I'll extract the V value.

00:27:36 And mathematically it's a simple weighted sum, and the weights are basically the softmax of the dot products of the QK pairs. We won't go into too much detail on that. That's very thoroughly explained. We went thoroughly through it in the previous episode, 747. But I guess the main takeaway is that these QKV vectors are created. Obviously they're randomly initialized because the weights of the initial neural network are random. But then over time, the transformer learns how to update the

weights in such a way that it cannot take advantage of the mechanism. I think for me, the biggest breakthrough in understanding attention was that we're not telling it what to do. We're not telling it, "Oh, use the Q vector like this. You have to put this value, this information, the K vector. You have to put this in a fashion in the V vector." We're just creating this mechanism for it to be able to take advantage of it and then create, populate the vectors in such a way that then it will, in all of our epochs and epochs of training, that it will be able to attend to different words.

00:28:45 So our job is to mathematically implement the mechanism. The transformer through training will learn to use it, and that's a key distinction. That's why we love neural networks.

Jon Krohn: 00:28:54 That was very nicely said.

Kirill Eremenko: 00:28:56 Thank you. Okay, so once we're done with attention, what we get on the output of this level three is that we have these context-enriched vectors. Now, every word previously it had just a semantic-enriched vector, which we added positional coding to. Now from that, using that QKV mechanism, we will have a context rich vector. So each word now knows the context of the sentence that it's in. Then all that goes through layer level four, which is a feedforward neural network. Why that's important, Jon very elegantly put it last time, it adds flexibility to the learning process because it adds additional weights. And also that's the only place in the whole architecture where we have an activation function. We haven't had any activation functions before prior to this.

00:29:45 And then level five, these vectors go to level five, where they go through a linear transformation to map them from the 512 dimensional space that they're in to the output space. And the output space in our case is the

whole, all of the words in the English language, which depending on how you count, can be 200,000 or more. So we want to go from a 512 dimensional vector to a 200,000 dimensional vector, and then we apply a softmax to that vector to get probability. So we get a probability distribution across all of the words of the English language, and that allows us to select something.

00:30:19 One very important key consideration, something to keep in mind that we talked about in the previous podcast and I want to reiterate now, is that each one of the vectors, let's say we have apples are a type of delicious blank, right? So six words, they all go through this process separately. So they go at the end or the whole way through, we have six vectors. Then each one of the six vectors gets converted into those three QKV vectors. Then again, we get six vectors, six context-rich vectors. Then after the neural network, they go through the feedforward neural network. They go separately. So we have six of them on the output. And then afterwards through the linear transformation, again, we get six vectors, 200,000 values each.

00:31:04 So six vectors, each one is 200,000 dimensional. Then we get six probability distributions, each one with 200,000 values. And then we throw away the first five and we keep the last one. So the probability distribution that we extracted from the context rich vector of the word delicious, and we apply that to all the words of the English language, so to predict what the next word is. That's a very key consideration. These vectors don't get mixed. The only time they can get some information from each other is in that attention mechanism. Otherwise they go through separately in parallel. And that's a very important part of the transformer architecture.

Jon Krohn: 00:31:36 Your business gets to a certain size and the cracks start to emerge. If this is you, you should know these three

numbers, 37,000, 25, 1. 37,000, that's the number of businesses that have upgraded to NetSuite by Oracle. 25. NetSuite turns 25 this year. That's 25 years of helping businesses do more with less, close their books in days, not weeks, and drive down costs. One, because your business is one of a kind. So you get a customized solution for all of your KPIs in one efficient system with one source of truth, manage risk, get reliable forecasts, and improve margins. Everything you need to grow all in one place. Download NetSuite's popular KPI checklist, designed to give you consistently excellent performance, absolutely free at netsuite.com/superdata. That's netsuite.com/superdata.

00:32:23 Very cool. This is a little bit of a tangent and I don't expect you to know the answer. I don't really know it either. But as you were talking about that final layer, how the fifth layer has that linear transform with the dictionary of ... And you say in the English language, and that assumes that this is a single-language LLM. But of course if we're using GPT-4 or Gemini or something like that, it can output in-

Kirill Eremenko: 00:32:45 Good point.

Jon Krohn: 00:32:47 ... hundreds of languages. So it's crazy to think how many tokens or how many words there must be in that dictionary. So one thing I was thinking about is, so what if we're not outputting works, but what if this is like DALL-E Three and we're outputting pixels? I guess the probability map would be some pixel color, the most probable color for a given location.

Kirill Eremenko: 00:33:13 Good question. I don't know the answer to image version of transformers, but I do know, for example, in BERT, you don't want to output a whole word. You just want to output a class, for example. So, you have that CLS token that gets added into the input at the beginning, and then

the sentence goes through the BERT model, which we can talk about at the end. But basically, effectively what in the BERT model they have is they don't have, the mapping doesn't go to 200,000 vectors, but goes to your number of classes, which could be three classes, for example. Positive sentiment, neutral sentiment, negative sentiment, and that's it. So I guess in the image version it would be somewhat similar.

Jon Krohn: 00:33:55 Nice. Anyway, just a little bit off-topic. Today we are mostly, or I guess other than that brief section, we're using examples entirely from text in, text out transformer models.

Kirill Eremenko: 00:34:07 For sure. Okay, so that's how the, what's it called, decoder model works. The LLMs are based basically decoder-only. So that was our quick recap. Now, what are we going to do today? We're going to add, to make the full transfer model. We have to add the encoder, which is on the diagram is on the left. And this is going to be really easy because we have all these foundational building blocks. Effectively we're imagining the decoder is a five-story building standing up on its own. Now we're going to add another building on the left is going to be a four-story building. So four-story building is going to have the same thing, input, embedding, positional encoding. Then it's going to have level three self-attention mechanism. And then in level four is going to have the feedforward neural network. So all very similarly doesn't have that output layer. Now, in order to connect them, we have to make a modification to our decoder model. So let's look at the part on the right again, the model that we were just talking about or the architecture we were just talking about, the decoder part.

00:35:13 So we're going to keep level one, two, and three intact. So up to the self-attention. Now, after the self-attention, we have the feedforward neural network, and then after that

we have the linear output and softmax. We're going to lift the feedforward neural network level four and level five, which is the linear transformation and softmax, we're going to lift them up and under them we're going to slot in another level and we'll call it level 3B or 3.5, as you wish. So we're going to have level 1, 2, 3 is going to be one of those weird buildings where you have level three, and then you have level 3B, and then you have level four and five. And so level 3B is another attention mechanism. It's called cross-attention, and we'll get to that in a bit. But first, what we're going to do is let's talk about the encoder so we know how it works. So it's got those four levels that we discussed. We're actually going to look at an example right away.

00:36:09 So to help us illustrate this, we're going to look at an example. Bear in mind, we're talking about inference, not training. We can mention training towards the end, but it will be very similar to how we talked about training for a decoder-only model. So we're going to look at an example. So transformers are basically the original design of transformers, the full model, was to translate sentences. So that's the use case that we're exploring in the research paper. We're going to look at the sentence, the cat sat on the mat. It's got six words and we're going to translate it into Spanish. So we're going to pretend that this transformer has already been trained on how to translate sentences into Spanish, and we're going to see how it's going to work with this specific sentence during inference. How is it going to translate it? So the correct translation is el ... I don't know Spanish, so I might butcher this, but it's "el gato se sentó en el tapete". So the cat sat on the mat. And here "gato" is cat, "se sentó" sat, "tapete" is on the mat.

Jon Krohn: 00:37:10 Sounded pretty good to me. I don't really know Spanish either, but I buy it.

Kirill Eremenko: 00:37:13 Thanks, man. So we're going to translate that sentence into Spanish. What happens, right? So the decoder encoder, right? Let's talk about the encoder first, the building on the left, the four-story building. All of the English words are going to go into this encoder part. The cat sat on the mat, six words. They're going to go through positional encoding, sorry, they're going to go through vector embedding. So they're going to get their vectors with semantic meaning. Then they're going to go through positional encoding, which we've already discussed. Then they're going to go through the self-attention mechanism. So whole QKV process is going to happen, and each one of these vectors will get a context, each one of these words will get a context-rich vector, so now we have six context-rich vectors. And then in level four, each one of these six context-rich vectors will go through a feedforward neural network. And to add some more flexibility, add more parameters so it becomes even more able to understand the complexities of language and so on.

00:38:10 So after that feedforward neural network, we'll have six vectors, which are contextually super rich. Let's call them super-contextually-rich vectors or super vectors after being contextually rich because they went through the feedforward neural network with activation function. And we'll call them vectors O for output, output of the encoder. So those are six O vectors sitting there, and we'll just put a pin in that. So we have six vectors that went through the four levels of the building on the left, which is the encoder, and they're sitting on the rooftop. That's that. We have now created a good representation, a very rich representation of our English sentence, and we're going to use that when translating.

00:38:49 Now, the main part, in my view, I would call it the main part of this translation process happens in the decoder. So all of the, it's going to be large language models, sorry, it's going to be like a GPT model where we are generating

text from a prompt. The only difference is, the best way to think about it is we're generating text, but with a condition on it. So for argument's sake, let's say we've already generated three words. So out of our sentence that we need to translate into, which is "el gato se sentó en el tapete" let's say we've already generated three words "el gato se", so we already have that as output. Remember, decoders generate one word at a time, so it generated "el", then something happened, then it generate "gato" then something happened, then generate "se", then something happened. Now we're going to find out how does it do it? How's it going to generate the fourth word?

00:39:44 So we have this output so far "el gato se" and the output comes from the decoder part, from the top of the decoder remember where we have those probability distributions, that's where we'll be getting the output. So now we have these three words "el gato se". What's going to happen is they're going to go back into the decoder at the bottom in Spanish. They're going to get their vector embeddings. So each one of these three words "el gato se" will have a semantically-rich vector, so it'll have the dictionary meaning associated with or encoded in that vector. Then they will get the positional encoding on level two as we discussed, that's important. Then they'll go to level three. The attention mechanism will happen among these same three words "el gato se". These words will get to understand the context of what they have so far of this three word sentence. That context will be coded into the vectors for these words. After level three of the decoder, each one of these words will have a context-rich vector. So a Spanish context-rich vector with the Spanish sentence that we have so far of three words.

00:40:52 Now it'll go into level 3B, and this is where the interesting thing will happen. This is like the biggest addition that we are doing today compared to the previous episode 747. So what's going to happen now is there's going to be cross-

attention. So we have these three Spanish vectors going into this level 3B, and at the same time on the building on the left, we can see there there's six vectors, six English vectors sitting. They're context rich. they're ready to go. So if we were not to look at the encoder at all, let's just imagine, forget about encoder, what will happen if these three vectors just keep going? There's no level 3B, they just keep going through the decoder. What will happen? Well, they'll go into the feedforward neural network.

00:41:37 Then they'll go into the linear transformation and softmax. And as before, as with any large language model, or GPT model, or decoder on the model, we will produce the next word. So they'll produce an next word. So based on "el gato se" that's like a prompt for, what's the next word? The next word could be paro, like the cat stood. The next word could be relajo. The cat relaxed. The next word could be escondió, the cat hid, right? Those are all valid next words. It makes sense to put a verb next. And a decoder-only model, like a large language model would just produce this next word, which is great. It could create a whole sentence, but that would have nothing to do with the translation. We don't want it to create a sentence based, it would just give us the next most likely word based on what it's seen in the Spanish language, in the corpus of text that t's seen in the Spanish language.

00:42:37 What we want it to generate, it's not just the most likely next word. We want it to generate the most likely next word based on what it's seen in training data, but conditioned on the inputs in English that we gave it. So this is where this level 3B, the cross-attention comes in. So let's rewind. These Spanish words went through level one, through level two, through level three, through the normal attention, now they come into this cross-attention. What happens here is that from each one of the vectors that we have, we already have context-rich vectors

for the Spanish words. From each one of these vectors, we're going to create three vectors. We're going to create another Q ... Well, actually not three. We're going to create one vector. So normally in attention we create QKV, but at this time we're just going to create the Q vector. So for each one of the Spanish words, we will have a Q vector created. And the KV vectors, instead of creating it from the same Spanish words, we're going to create them from the English words sitting at the top of the encoder, right? We have those six English context-rich words on the left in the image. Instead of creating QKV vectors from each one, we're just going to create the K and V vectors. So now we can-

Jon Krohn: 00:43:41 Okay, I'm just going to interrupt you for a second. This seems like a pivotal moment. So I'm going to recap and you can confirm for me that I'm getting this right.

Kirill Eremenko: 00:43:52 Sure.

Jon Krohn: 00:43:53 So we've got two buildings next to each other, or one structure with slightly different heights right next to each other. So I guess maybe it's easier to think of two buildings next to each other because they can have their separate elevator systems or whatever. And so on the left we have the encoder, which is four stories, and that's completely new in terms of what we've been talking about on this podcast. In episode number 747, we only talked about the right-hand building, which had five stories. Now we've got this building on the left, which has four, but we've also, in order to allow these two buildings to communicate effectively with each other, you've added in an extra floor, 3B, between the third floor and the fourth floor on the right-hand decoder building.

Kirill Eremenko: 00:44:43 Correct.

Jon Krohn:	00:44:44	So we have flowing up the elevator on the encoder building on the left is the English language that we want to have translated into Spanish. And then on the right-hand side, there is Spanish language flowing upwards as well. And then they meet at this 3B, so information flows from left to right, from the fourth floor, the top floor of the left-hand encoder tower. It flows into level 3B of the decoder tower. And that's where this self-attention, this powerful thing is happening where-
Kirill Eremenko:	00:45:29	cross-attention.
Jon Krohn:	00:45:30	Cross-attention where ... Yes, yes, yes. Right, right, right, right. So the self-attention is within one tower itself. So the decoder has self-attention, that's a really key thing here. Thank you. Yes.
Kirill Eremenko:	00:45:41	And the decoder has self-attention as well. Its own self-
Jon Krohn:	00:45:44	Has its own self-attention. And that was level three in both?
Kirill Eremenko:	00:45:47	Yes. self-attention is always level three.
Jon Krohn:	00:45:49	Nice. And then now in this 3B, on the right-hand side decoder, we've got cross-attention, which is blending the vectors, the Q, K and V vectors that we recapped. And again, you can get tons of information on those from the preceding episode that Kirill was on, episode number 747. But it's the Q vector, the query vector comes from the encoder. The left-hand side-
Kirill Eremenko:	00:46:14	No, no, from the decoder. Q vector comes from the decoder.
Jon Krohn:	00:46:14	Oh, from the decoder?
Kirill Eremenko:	00:46:17	Because the Spanish words [crosstalk 00:46:19] need the context.

Jon Krohn:	00:46:20	Right. Right, right, right, right, right.
Kirill Eremenko:	00:46:21	And the Q vector is the query. It's like, "I need this context. Do you have it? I need this context. Do you have it?" That's the Q vector. So that is in the right side.
Jon Krohn:	00:46:30	Nice. Q vector is on the right, on Spanish side and the decoder. And then the K and the V are on the left on the encoder.
Kirill Eremenko:	00:46:36	Correct.
Jon Krohn:	00:46:36	Nice. Okay. And so those blend together. And so it's a beautiful thing. So it's allowing that same kind of structure, these three vectors, the Q, the K, and the V, which previously as we learned about in the self self-attention, allowing us to figure out what the next token should be appropriately. Now we're getting that same kind of mechanism but being applied across these two different sources of information and blending them together so that you're getting context from one side. And how would you describe, could you describe? So if Q is the context. No, K and V is the context,
Kirill Eremenko:	00:47:14	K is the key, and V is the value that contains the context.
Jon Krohn:	00:47:19	Yeah. So then is there a way that we could sum up the Q side as the compliment to context?
Kirill Eremenko:	00:47:26	It's hard. It is hard to say. It's just a mechanism really. I would say the context is inside the V vectors. The Q and K is just an indexing mechanism. It's li ke Q, "This is what I'm interested in. Tell me about ... I'm going to add human interpretation to this." Obviously the transformer doesn't do this, but just for argument's sake, let's say the word we have "el gato se", right? So the word "se" is like, "Oh, I'm curious about what action is being performed. I need context about a verb. Give me context about a verb."

So the Q value in human understanding would be like, "Oh, I'm looking for verbs." And then on the encoder side, we've got the six English words. The cat sat on the mat, and so they have the K vectors.

00:48:17 Again, this is butchering it, but it's very approximate, but it's better for understanding each one of those K vectors is going to have information, "What context do I have?" And so maybe the word cat, it'll have a K vector saying, "I'm a noun," and the cat sat the word sat has the K vector saying, "I'm a verb. I've got a verb inside here. If you want to look for a verb look in here," sat on the mat. And then the word mat has the K vector saying, "Oh, I'm a subject, or I'm an object." The cat might have, "I'm a noun and I'm a subject." The word mat might have, "I'm a noun and I'm an object". And then the word the is like, "I'm an article. If you need information, an articles, look inside here." And so the Q vector for the word "se", which is, as I understand in Spanish, it's the start of a verb, but it's like, what's the next verb? What's the next verb? What's the context for this whole situation? It goes and it's like, "I need a verb."

00:49:10 And it goes across these K vectors like, "No, you're not a verb, you're not a verb. Oh, you got a verb inside. Let me look inside the V vector for the word sat." And inside sat is like, "Oh, I'm a verb for sitting down. This is what it feels like, blah, blah, blah." And that's where the Q kind of query, it communicates through dot product with the K vectors of what it's looking for. When it finds a match, boom, it looks for the V, which contains the context that it needs.

Jon Krohn: 00:49:40 Last month, HPE & Intel together showcased the power of RAG — Retrieval-Augmented Generation — to bring relevant business data to your LLMs. In this month's free workshop, you can learn about the art of fine-tuning embedding models to deliver verifiable conversational

chatbots using the HPE Machine Learning Development Environment powered by Intel Xeon Scalable processors. No matter your experience level, join us to learn practical techniques to build trustworthy chatbots with guaranteed behavior for enterprise applications. Visit hpe.com/ezmeral/chatbots to register today. We've got the link for you in the show notes.

00:50:20 Beautiful man, you've really got this down now. I feel like. I mean, you obviously had it down even when we recorded 747, but it is very tight for you and your mind. It's like second nature.

Kirill Eremenko: 00:50:28 Thanks man. Thanks. Spent six months on this instead. I thought it was going to take four weeks, took six months. It's literally been six months now. But I'm very glad. I'm very glad that I got through this. Yeah.

Jon Krohn: 00:50:42 Okay. Yeah. So I derailed you as you were-

Kirill Eremenko: 00:50:44 No, no.

Jon Krohn: 00:50:44 ... describing it.

Kirill Eremenko: 00:50:45 That was really good to clarify.

Jon Krohn: 00:50:46 The cross-attention, yeah.

Kirill Eremenko: 00:50:46 Thank you. And I didn't have this example in mind to share about, I just came up with this now about the verb, the noun and so on. I think that was very helpful. What I wanted to add, I love your elevator analogy. So we have the two systems. The building on the left elevator goes all the way to level four. There's a small little bridge from level four of the encoder on the left to the 3B level on the decoder on the right. That's where the values, the KV values walk across. And as you said, they blend with the Q values. A very important point is that the elevator on

the encoder works only once. It's a one-way, one-off elevator. So the English words get into the encoder and they go vroom to the top and they sit there. It doesn't happen every time.

00:51:30 And this is like, I love these podcasts because they force me to think, go outside my comfort area and force the barriers. I figured this out yesterday. I was like, "All right, I've learned transformers for five months. I know what I'm going to say. Do I even need to prepare?" And I sat down, I prepared anyway, and I thought the elevator keeps going up every time and you were just hearing it. But no, it goes up once and they sit there, why would it need to go up every time? It's the same six words. Your input English sentence doesn't change. So it makes sense. So these six words get into the encoder, they go up the elevator, they sit at the top, and they have these six context-rich vectors that are used again and again and again.

Jon Krohn: 00:52:14 I think it's also a really good point to make that about the elevator only going up once because that's the whole idea of the compute efficiency of this.

Kirill Eremenko: 00:52:20 Yes.

Jon Krohn: 00:52:20 So something that you mentioned, going back to that attention is all you need paper. And the eight different authors, they were specifically working together to come up with a computationally efficient mechanism that would work well on modern GPUs. And the elevator only going up once is a great example of how you're going to get more compute efficiency from this process.

Kirill Eremenko: 00:52:40 So that's absolutely correct in terms of compute efficiency. So now let's recap what we have. We have "el gato se", these three Spanish words. On the left we have the English words that went up the encoder, and they're sitting there. That happens only once, up that elevator.

We have these context-rich vectors, we'll call them O vectors. Now, on the right we have these three Spanish words. They go through level one, through level two, then they go through the self-attention mechanism in level three, where they get QKV vectors. Through that mechanism, each word gets a context-rich vector. Let's call those vectors A. Now, at the end of level three, we have three context-rich vectors, which are vectors A, one for each word. Then all of those vectors A go into this cross-attention mechanism, level three B.

00:53:32 On the one hand we have the O vectors from the encoder, the output of the encoder, six of those vectors, the English sentence. Then we have the A vectors for the three words in Spanish. All of that is going to get merged in the cross-attention. By merged, what we mean is that from each A vector, we will create only Q vectors. The K and V vectors are not even computed. Only Q vectors are computed from the A vectors inside the decoder side of things. We'll have one Q vector for every Spanish word. Then on the English side of things, the Q vectors are not computed, but the K and V vectors are computed. We'll have a K and V vector for every English word. As we just discussed, the cross-attention mechanism will happen. From there, the Q vectors will look for the context they need using the K vectors, they'll find it and the context will be taken from the V vectors. From that, we will create new context-rich representations in the decoder side of things for each one of the Spanish words.

00:54:34 Now we will have, let's call them vectors B, because it came out of level three B. We have a B vector for the word "el" a B vector for the word "gato", and a B vector for the word "se". Now, these B vectors, they take into account the context that came from the English sentence. From here everything is the same. As before, the B vectors go through the feedforward neural network, each one of them goes through the feedforward neural network, and

then each one of them goes through the linear transformation and softmax to get the output.

00:55:08 From each one of the Spanish words, so we had "el gato se" from each one of them, we'll get a... What's it called? Probability distribution with 200,000 values, or however many hundred thousand words there are in the Spanish language. We'll throw away the first two, as we did previously, we'll throw away all of the probability distribution except for the last one, which in this case is the one that we derived from the word "se". We'll apply that probability distribution of 200,000, or however many values, to all of the words or tokens in the Spanish language. That will give us the probabilities for them. Basically, we'll know which word, the one with the highest probability, is the one that we're going to use next.

00:55:51 In this case, it's not going to be "paro", it's not going to be "stood" or "relaxed" or "hid", right? Paro, relajo, escondio. We know, because the predict this generation is now constrained or conditioned on the English language sentence, now there's no choice for the decoder, but to say that the next word is going to be "sentó". The prediction is going to be that, "The cat sat on the mat." If input sentence in English is, "The cat sat on the mat," it wouldn't be able to say that in Spanish that, "The cat relaxed on the mat," because that would be contradictory to the context that it got from the English sentence. That's basically, that's how it works. The best way to think about the full transformer model is, imagine it as just a decoder-only model that's generating text. But because of that level 3B that we added in, it's generating text based on what it's seen in training data in that language. But that text has to be conditioned on the input sentence we gave it in the original language. That's all it is.

Jon Krohn:	00:56:54	That's all it is.
Kirill Eremenko:	00:56:58	Ah yes. Yes, yes. A very interesting thing that I wanted to discuss is why is there no masking, right? There's only masking, we talked about masking in the previous episode.
Jon Krohn:	00:57:08	Can I ask you a question before we get to masking?
Kirill Eremenko:	00:57:09	Sure.
Jon Krohn:	00:57:10	Something that we discussed in the preceding episode, and then we've also had in a couple of other episodes, because we've had some amazing authors on transformers on the show in the past. For example, we had David Foster in episode number 687, and we had Lewis Tunstall in episode number 695. Both of these guys, great experts in transformers, even they didn't get anywhere near, I mean, we have never had an episode like your episode 747, or this episode today, where we are going so deep into the details here.
Kirill Eremenko:	00:57:48	I'm sure they know way more than me. It's just we designed this episode to be so technical.
Jon Krohn:	00:57:55	Yeah, I'm not trying to make that kind of comparison at all. We didn't get into the mechanics of each level of a transformer and what's happening. But something that we did talk about in those episodes was, and something that we talked about in your 747, I think if I remember correctly, was that we talked about how an encoder-only architecture, like BERT, is great for things like natural language understanding. This whole episode we've been trying to talk about encoders and decoders together. It's the two build things. But you can have transformers where you just have encoder only the left-hand building. When you do that, you can't be generating text with that. But you can be encoding natural language into a vector. I

mean, we have these output vectors, and so you end up with an output vector out of the encoder that allows you to have a numeric representation of the semantic meaning of whatever words went in.

00:59:00 This can be great for lots of different use cases, like classification tasks or sorting tasks. This is something that we do at my company, Nebula, all the time. It's a core part of what we do. We're trying to help people find great candidates for jobs, say based on a job description. You'll convert the job description, you'll encode it using a BERT-like architecture. Take the natural language of the job description, encode that into a vector. Then we have a database of 200 million people, basically all the working population in the US. We have already pre-computed the vectors using a BERT-like architecture from all the natural language that we could find on each of these 200 million people. You can think of it as a spreadsheet with, say we happen to have... I mean let's say we have 200 length vectors. You could think of it as a spreadsheet with 200 million rows and 200 columns. For each of the 200 million people, you have their location in basically this 200-dimensional space. Given this new job description that we encode in real-time, we can then compare, okay, who is closest to this job description? Who's going to be the best fit from these 200 million people? That's an example of how encoding on its own can be super powerful using a BERT-like architecture.

01:00:24 The decoder, the right tower is very useful for anytime you want to generate text, because it outputs a sequence of tokens. It outputs a sequence of words. I feel like it's very easy for me to understand, at this point in my journey with these things, that encoders are great when you want to be encoding natural language into a vector for whatever kind of task, classification or ranking. Then you want to use a decoder when you want to be generating text. Why would we want to use an encoder

and a decoder together? What is the advantage that that full transformer architecture gets us that I can't get with a decoder alone?

Kirill Eremenko: 01:01:09 Great question. Great question. I wanted to say first, I loved your location in a 200-dimensional space. When you said that I was like, you know, like data privacy and stuff like that, your address needs to be private. It would be funny if we had to make it private where your location in the 200-dimensional space is. That's private data. Just quickly to comment on what you said about the encoder, absolutely correct that you can capture the meaning, or semantics, or whatever you need of a text in a vector. But just for the sake of our listeners, it's not what we did in this case.

Jon Krohn: 01:01:56 Oh, right, right, right, right.

Kirill Eremenko: 01:01:56 We got to the point where each vector in the encoder, those six English words, they have their own context-rich vector. What Jon is talking about is if you throw away the decoder part, and on top of those context-rich vectors, you add the fifth layer that we're used to, the linear transformation plus the... What's it called, softmax function. Then what you would also have, basically in the BERT-architecture, you would have your six words like, "The cat sat on the mat," but at the beginning you would add a separate secret token that is not visible to the naked eye of the person typing. It's just for processing reasons. It's called the... What's it called? CLS token, the class token.

01:02:41 It gets added at the beginning of that sentence. Then now when you're processing that input, forget about the decoder side. Just imagine you only have the left tower on its own. When the vectors go through that part, the CLS token also gets its vector embedding. It also gets a positional encoding. It participates in the whole self-

attention mechanism of the encoder. Then it goes through layer four. It also gets that feedforward neural network applied, and then all of those six plus one vectors now go out the linear transformation to get mapped to a certain, let's say, space, like a three-dimensional space, or let's say in your case, a 200-dimensional space. You want 200 dimensions out of your context that you're giving. You get some values in that 200-dimensional space. Or let's say if you're doing the sentiment analysis, it's a three-dimensional space. Then you throw away all the vectors except for the first one. You keep the CLS vector, you're interested in what class did the CLS vector get. That's how the encoder-only architecture called BERT works.

Jon Krohn: 01:03:52 Right, right, right. That is a really key piece of information here. In the encoder-decoder that we've been talking about most of this episode, until I derailed things just now, with that, you have the vectors coming out for each of the positions in the encoded sequence.

Kirill Eremenko: 01:04:10 Yes.

Jon Krohn: 01:04:10 Each of the words, so all of that rich context. Whereas, in the kind of situation that I'm describing where you want to be doing some classifications task.

Kirill Eremenko: 01:04:19 BERT.

Jon Krohn: 01:04:20 Yeah, BERT. Where you want to be doing some kind of natural language understanding task and just have the encoder, you're not worried about generation, then you're using that CLS vector to collapse everything down to just one vector in the end. Otherwise, you'd have a vector for every single word in the input sequence, which would be overwhelming. There might be... I don't know, who knows? Maybe for somebody that is useful. I mean, obviously it is useful when we're passing all that information to a decoder. You could imagine maybe

somehow someone else has come up with some other kind of application where all of that information is useful, but it's probably very rare. For the most part you're using the CLS token to get just one vector out for your entire-

01:05:06 So in my case, when I put a job description in, you don't want to get a vector out for every word in the job description. You just want one vector for the whole job description altogether. That CLS vector, the mechanism that you just described. The last [crosstalk 01:05:18].

Kirill Eremenko: 01:05:18 Well, you technically will get a vector for every word in your job description, but then you'll just disregard them. You will have, let's say you have a thousand words in your job description, now you're adding the CLS at the beginning. Now there's 1,001 vectors. They all have to go through level three of the encoder. They have to go through level one, two, three, because that's where self-attention happens. That's where the vector for CLS will be enriched with the context from the thousand other vectors. Then from there it doesn't matter. The other thousand vectors don't matter. From there, the CLS, you can throw them away. The CLS will have to go through the feedforward neural network, and then it'll go and get to the output like lean transformation and the softmax, and you'll get whatever you need from the CLS from there. But at least up to the self-attention mechanism, all of the vectors have to go up to that point.

Jon Krohn: 01:06:10 Nice. Yeah, yeah, yeah. Thank you for clarifying. Yeah, I was imagining that in my head. That was obvious to me. But it very, very important to be able to make that explicit that you need all of the vectors for all of the words, say in the job description, in order to be able to do the computation of the encoder. But then out of the final layer, all I want is one vector representing the whole job description, not each of the words.

Kirill Eremenko: 01:06:33 Yes, yes.

Jon Krohn: 01:06:35 The mechanism that you described with the CLS token provides for that. Very cool. All right, so...

Kirill Eremenko: 01:06:40 Back to your question.

Jon Krohn: 01:06:40 Yeah.

Kirill Eremenko: 01:06:40 What's the point?

Jon Krohn: 01:06:41 Yeah.

Kirill Eremenko: 01:06:44 Very interesting.

Jon Krohn: 01:06:44 Why an encoder-decoder?

Kirill Eremenko: 01:06:45 Very interesting. I looked into this, and by no means am I a researcher, so I'm not best positioned to answer this question, and I'd be happy to be corrected on this, but based on what I've seen, GPT models are able to perform any task that a transformer model was designed to perform. Even to the point of translation, because GPT models, or I'm just not talking just... Basically I'm talking about decoder-only models. Decoder-only models, because they're seeing so much training data which contains both English, and French, and Spanish, and all these other languages, and examples of translations, and examples of the same text translated into another language, they're able to like- How can you just put into ChatGPT some text and it'll translate? Yes, of course, you can fine-tune a decoder-only model after training to translate better, better to understand the different nuances of translation. But already in their raw format, they're able to do translations. Again, I'm happy to be corrected, but I can't think of a single task that you would specifically need a transformer architecture to do, rather than just using a decoder-only. What do you think, Jon?

Jon Krohn:	01:08:06	Yeah. When you say transformer architecture, you mean... Because the decoder-only is a transformer architecture.
Kirill Eremenko:	01:08:11	I mean the full transformer architecture.
Jon Krohn:	01:08:12	Yeah, the full transformer architecture. Yeah, I think maybe twice there, just to disambiguate. In the last few sentences, when you were contrasting transformer versus decoder, you're saying full transformer encoder-decoder models.
Kirill Eremenko:	01:08:31	Yes, correct. Yes.
Jon Krohn:	01:08:34	Yeah, maybe it has something to do with scale that I'm guessing here, but maybe when you had... Now today, things like a GPT-architecture that are decoder-only, they're so gigantic with supposedly this... The well-regarded rumor regarding GPT-4, is that in aggregate across all of- It's probably composed of about eight different experts. When you add up all those expert models together, it's probably over 2 trillion model parameters in total. I think maybe what's happened is that as you've gotten to such enormous decoder architectures, in those weights, they have to be able to figure out how to encode in some way anyway. All these enormous amount of weights allows that to happen, especially over all of the layers of transformers, which we didn't talk about in this episode. But you talk about, at the end of 747, where you talk about how you have many layers of transformers. You don't just have one. You have even GPT-2, there's different sizes of GPT-2, but there were even GPT-2s many years ago where there were dozens of transformer layers doing...
	01:09:57	I think the idea is that a gigantic decoder-only model like a GPT-3, a GPT-4, it just has so many model weights that it figures out how to encode the important information in

the decoder itself. But maybe if we wanted to have a smaller model with fewer model weights, perhaps that explicit encoder functionality would allow the context to be stored for tasks like content comprehension. If you're going to have some generative task, that depends a lot on content comprehension, maybe the encoder, especially if we're trying to, I'm speculating here, but if we're trying to have a relatively small model in terms of weights, then maybe that encoder part will allow a richer understanding of the context, given a smaller number of model weights, and then pass that "understanding" off to the decoder for its generative capabilities.

Kirill Eremenko: 01:11:02 Yeah, that's a very interesting point. Another way to also think about, while you were speaking, this idea came to me that, imagine you're translating a huge- It's about context window. Imagine you're translating a one-page or ten pages of English text into Spanish. What happens if we were using a decoder-only model? You put that English text into the decoder-only model. You see, I prompt, translate into Spanish, and then it generates the first word. Then it takes all of that, plus the first word, puts it back into itself, generates the second word. Then takes all of that, puts it back into itself, generates the third word, and so on.

01:11:42 As it's generating more words, it's having to rerun the vectors and all of the calculations for the English text. We have 10 pages of English text. Plus, they're going further and further away inside the context window. But if you use the full-transformer architecture, then the English text goes into the encoder, and as we said, that's a one-off elevator. It goes up that elevator once, those ten pages get encoded to words once, into vectors once, and you don't need to re-encode them, they're not changing. Then the decoder model is just generating its word-by-word thing, but it always has these ten pages of vectors to reference

in the English language. It doesn't have to regenerate them. It's also more compute-efficient in that sense.

Jon Krohn: 01:12:28 Yeah. Interestingly though, I think somehow it ends up being the other way around. I think that somehow, and I can't explain this, maybe it's something we can try to figure out for a future episode, but my understanding is that one of the big draws is that, overall, a decoder-only architecture ends up being more compute-efficient.

Kirill Eremenko: 01:12:48 Oh, okay. Yeah. I see what you mean. Yeah.

Jon Krohn: 01:12:52 Even though, yes absolutely, what you're saying is true. The elevator only goes up the encoder one time. The overall effect is that having... The main drawback of an encoder-decoder model seems to be less computational efficiency. However, that encoder allows for extra context, allows for extra understanding. It could theoretically, I guess, outperform a decoder-only model. Especially where lots of context is important, like you're saying. But the decoder-only ends up being so much more compute-efficient that where we are right now is that the biggest, most effective models out there, like GPT-4, use the decoder-only approach because it's so much more compute-efficient, and therefore cheaper for OpenAI to be running those servers and giving us the results. It ends up being this trade-off where you can save on compute by going decoder-only, and you scale up the amount of layers and parameters in that decoder-only architecture. That ends up, somehow, by ways that I don't fully understand, managing in that decoder-only structure to capture enough of the context that it outperforms, or it performs more than well enough despite the encoder not being there.

Kirill Eremenko: 01:14:28 Yeah, no, that's a very good point. I'm sure there'll be a time when we'll uncover that on the podcast as well.

Maybe some guest with research experience will be able to answer that question.

Jon Krohn: 01:14:40 Yeah, that'd be great. Maybe one of the authors of the Attention Is All You Need paper. That'd be a pretty cool guest.

Kirill Eremenko: 01:14:46 Yeah, that'd be cool.

Jon Krohn: 01:14:46 They could turn... Yeah, a little bit of conjecture here, though, some... Yeah again, to caveat, neither Kirill nor I are transformer experts in the sense of publishing papers on transformers, but this seems to be... I have some references up as I'm saying, the things that I'm saying here that seem reliable.

Kirill Eremenko: 01:15:10 You talked about layers. Let's jump into layers for the full transformer architecture and see how that works there.

Jon Krohn: 01:15:18 Yeah, please do. Yeah, yeah, yeah. Oh yeah, because this ties into the masking that you were going to talk about too.

Kirill Eremenko: 01:15:23 Masking. Yeah, we can either talk... Maybe let's cover masking first and then we'll do the layers. What I wanted to say about masking is that actually masking ties nicely into BERT. But we'll start with masking on the decoder side. As we remember from episode 747, there's masking in the self-attention mechanism on layer three. Technically, the full name of that mechanism is the masked multi-head dot product self-attention, seven words to describe that much. Why masking? Well, during training, masking prevents the transformer from looking ahead. What we discussed in episode 747, towards the end, is that why transformers are so incredibly powerful is because they don't learn one sentence at a time. They don't see just one example and they learn to predict the next word. They see, let's say you give it a thousand

words, then because it's a triangular mask, when it's processing the first word, the first word can't see any other words. Then the second word can only see the first and second word. The third word can see only the first, second, third word, and so on. Every word can only see the words up to itself and including itself during that attention mechanism, when it's getting that context-rich vector.

01:16:46 Because all of these words are processed in parallel, what, by design, the transformer is able to achieve, is that when you give it a thousand words as input, and it's not just learning to predict the thousand and first word, it's learning to predict the thousand first word from the first thousand. It's learning to predict the thousandth word from the first 999. It's learning to predict the 999th word from the first 998. Just pick any sequence inside there. It's learning to predict the seventh word from the first six words. All that happens in parallel. It's able to calculate a thousand training errors in one go. Then from that, calculate the combined loss function, and do back propagation.

01:17:30 That's why we have masking in the first part, first self-attention, which is in the decoder level 3. The question is: Why don't we have masking in the self-attention inside the encoder level 3, and why don't we have masking inside of the cross-attention inside the decoder, which is level 3B? The first one out of these two, the one in the encoder, we don't have masking inside the encoder, because we want the English language words, or the input language words that end up at the top of the decoder, we want them to have context-rich vectors that are fully aware of the whole input sentence or paragraph that we're translating. If you think about it, when you're translating a paragraph, you know the paragraph in advance. Except for if you're doing... What's it called? Simultaneous translation with somebody speaking, and

then you don't know what they're going to say. That's a different situation. Maybe the architecture there should be different. But in most cases, you already know the full paragraph or page that you need to translate. When you're translating it, you have the full context. That's why, in the encoder side, there's no hiding of context from the encoder. It just gets all the context right away and all of the words can see each other in the self-attention. And then in the cross-attention, now let's say we have this sentence which we're translating, "The cat sat on the mat." Now we've got these first three words, "El gato se," we've generated them in Spanish already. So, when we did the self-attention in level 3, there was masking. So, during training, they would not have visibility of the other words.

01:19:12 But when you get to now the cross-attention, think about it when you're translating as a human, and I mentioned this in episode 747, Dmitry Bogdanov sent this in his reply to Andrej Karpathy in an email, which Andrej Karpathy talks about in one of his YouTube videos. So, Dmitry Bogdanov, who came up with this attention mechanism in an earlier paper, says that English wasn't his first language and when he was learning English, the way he would learn to translate sentences is, he would have this original language sentence, and then he would need to translate it, let's say, into English, and he would write, let's say, first word, second, third word, and then you're writing the fourth word.

01:19:52 You're not trying to keep in mind all of the words that were there, or some part of them that were in the original sentence, you have access to all of them. You take a pen and you look through them and you read through the whole context and so on. So, when you're translating from one language to another, you need the full context of the input language as well, so that's why we don't have masking in the cross-attention. So, very important to

remember in the encoder-decoder architecture, there's only one place where there's masking, and that is the first self-attention inside the decoder.

Jon Krohn: 01:20:27 Very good explanation, that makes a huge amount of sense.

Kirill Eremenko: 01:20:29 And by the way, that really ties in well to BERT. So, just as a quick reminder for those who don't know, BERT, interestingly, the abbreviation stands for Bidirectional Encoder Representations from Transformers, and the keyword there, encoder representations, already talks about that it's in the encoder, so you don't need the decoder model. So, if we look at the encoder-only model by itself, like a BERT model, then it has self-attention, but it doesn't have masking, and that's where the word bidirectional in the abbreviation BERT comes from, because the words inside, that you put into an encoder-only model, when they're creating context, they're able to look forward and backwards, so it doesn't matter, there's no triangular mask to limit each word from looking ahead.

01:21:19 It can look ahead, even in training. It can look ahead, it can look at the whole thing, because we are trying to take all the context, we're not trying to generate text, we're trying to create... Our goal is not to generate text, our goal is to create a representation of this text, classify it, or get some 200 values about somebody's job experience. So, you need some sort of data and you want to look through all the data, and that's the power of BERT, that it's bidirectional. And that comes from, the goals are different. The decoder-only model, you generate text, and so GPT, Generative Pre-trained Transformers, it's got the word generative in there, so it's got... Another way of thinking about it also, is that decoder-only models are causal, so there's a cause and effect. So, certain words cause other words that fall in the sentence. You shouldn't

allow the model to see the words that are coming ahead in training, whereas in the BERT model, which is bidirectional, it's non-causal. There is no cause and effect there, you want to look at all of the words. And yeah, that's what the abbreviation BERT stands for, it's because there is no masking that it is bidirectional.

Jon Krohn: 01:22:27 Yeah, nice. So, I'm going to repeat back some of that that you just said on masking, and that was a really great explanation. So, the whole point of masking is so that, when you have a generative task, like you said, a causal task, you're trying to predict what the next token should be based on the tokens that have already been output. And therefore, in that kind of situation, it would be like if you imagined in a time series prediction, where you're trying to predict the stock market price tomorrow, you couldn't train a model that gets access to the stock market prices in the future and in the past and try to use that to predict [inaudible 01:23:13] what the stock prices are going to be tomorrow.

01:23:15 Yeah, because then during training, you have access to future information that, during inference time, you're obviously not going to have information about what the stock market's going to be tomorrow.

Kirill Eremenko: 01:23:25 Yeah.

Jon Krohn: 01:23:26 And so, this is a similar kind of thing where, for a generative task, if you could see ahead what the next word is supposed to be in the training dataset, then you don't need a model for that, it's just perfect information. So, the masking hides what the future stock price is going to be in this case.

Kirill Eremenko: 01:23:44 In training.

Jon Krohn:	01:23:45	In training, yeah, so that way you're getting a model that is going to work well in real-world circumstances. And so yeah, it's something that's the very nature of this causal, of this generative task, is being able to predict the next token, and so it's essential that we mask, that we hide, what the next tokens would be during training, otherwise the model is just memorizing, it's not learning.
Kirill Eremenko:	01:24:16	It's going to be useless.
Jon Krohn:	01:24:19	Yeah. Whereas, in contrast, with an encoder-only architecture like BERT, all of the context can be used, which is great, because in that case, you're not trying to predict what the next token is going to be, you're trying to do some classification task, and that can end up being what the extra power of an encoder, like BERT, can be when you're thinking about... I was describing ranking people for a job, or doing a classification task in that kind of scenario, because you don't have this, you're not trying to be causal, and I'm not trying to predict what the next token is going to be. You might as well make use of all of the context, all of the words that follow, as well as are ahead of a given word of interest in the input text.
Kirill Eremenko:	01:25:08	Absolutely. Great summary. It's interesting, you mentioned GPT, or you mentioned predicting stock prices. In case listeners might find this interesting, I was looking at a few use cases of large language models in business and Bloomberg, in I think March 2023, or first or second quarter 2023, they released their Bloomberg GPT, which has 50 billion parameters and it was the first of its kind GPT model trained on financial data. And yeah, it basically outperforms similarly sized NLP models, or non-specialized GPT models, because it was trained specifically on financial data. So, indeed, that is a very good use-case example.

Jon Krohn:	01:25:54	But in that case, I think, if I remember correctly with Bloomberg GPT, that while it is trained on financial data, it is still a text-generating model, it's not a stock price prediction model.
Kirill Eremenko:	01:26:07	Yeah, I don't know the details of that. You're right, it might be.
Jon Krohn:	01:26:12	The idea would be that you could consult with it. It has the knowledge of all of the greatest Wall Street analysts, and so you can ask it questions like, "I'm thinking of buying Disney. What kinds of factors do I need to look out for as I make this purchase? What are the potential pros and cons?"
Kirill Eremenko:	01:26:32	It's not predicting stock market. That's a good correction, yeah, I don't know enough about it to comment on that. I think you are right in that sense. So, another cool, relevant, and this is going to be very highly technical question, which took me probably a few days to figure out the answer to, is we talked about masking... Oh no, what am I saying, days? It took me a few weeks to find the answer to this. Masking is important during training, obviously. You don't want it to look ahead so it can learn better. Question: is masking needed during inference? What's the point of masking during inference? Can't we just turn it off and save on computation? Why do we use masking during inference? Why do we keep it on?
Jon Krohn:	01:27:21	Oh, man. I don't know.
Kirill Eremenko:	01:27:24	This one, I was breaking my head about it, I was searching, couldn't find an answer to it, I was asking ChatGPT, "Give me an answer. Why do you use masking?" I'm like, you don't need masking during training, because you're throwing away. Let's say you have a six-word input, it gets to the end. We're talking about GPT-only models. Sorry, decoder-only models. It

gets to the end, you have those six 200,000 probability distributions, and you throw away the first five, you only use the last one to predict next word. And the last word has access to all the words by default. Masking doesn't affect the last word anyway, so what's the point of masking during inference if you're throwing away the first five probability distributions of the other words?

01:28:08 And so, I was breaking my head about it, I was thinking, "Why do we need masking? There's no point in this." And ChatGPT was so stubborn, it was like, "No, you do need masking." It's trying to explain to me, but it wasn't doing a good enough job and I couldn't understand. And then, somewhere on some forum and some hidden discussion, I was reading something and finally it hit me. The reason that you need masking during inference is because of layers. That's the only reason you need masking. So GPT, the original transformer, has six layers, the ChatGPT, I think version three had 96. Remember we were talking about 96 layers in the previous podcast?

01:28:46 So, because you have this decoder-only architecture, but then you don't use the output that it generates, you put another decoder-only, and then another, 96 times like that. So, by the time the words get to the end of the decoder-only level... So, you have these context rich vector representation, you don't have that linear transformation and output, you don't create the... So, you basically have layer 1, 2, 3, 4, you have that fit for all dual network, then you don't do the linear transformation and softmax, you just put straight up, on top of it, you put the next decoder on the architecture where you don't have level one, because you already have vector representations.

01:29:23 I'm not sure, I don't really remember, I think you do still need the positional encoding where, when they go in there, that's a small detail. But anyway, so you have

these six context-rich vectors for the six words, let's say, that you have. They go at the end of layer one, then they go into layer two. And as they go through layer two, they are used in the creation of the context for the sixth word. So, if you turn off masking, all of a sudden these six words, these six vector representations you had at the end of the first layer, are going to be different in training and in inference.

01:30:01 You'll still get a result, but in training and inference, there'll be a difference. And the fact that they're used for the context of the sixth word in layer two, is going to affect the sixth word in layer two. And so in layer three, and so in layer four, and then it'll get to layer 96. So, masking can be switched off only in the last layer, but in the previous layers, it's important to keep it so that your training architecture is identical to your inference architecture. So, it's a very deep, technical question, but I think if I was hiring somebody for an LLM position, I would ask that question. I would say, "Do you need masking during inference?" And they'll say, "No, you don't." And then I'll ask them why, and then we'll talk about layers and I'll see how well they understand it.

Jon Krohn: 01:30:48 Yeah, so the end to that, you just said that the interview with your artificial interviewee, they say no, but the answer is yes.

Kirill Eremenko: 01:30:56 The answer is yes.

Jon Krohn: 01:30:57 There is masking, but it's because you need it in everything except the top layer.

Kirill Eremenko: 01:31:02 Yes.

Jon Krohn: 01:31:03 So, in those 96 layers, the 96th layer doesn't need it, but the first 95 do.

Kirill Eremenko: 01:31:06 Yes, that's right. That's right. Because the first five vectors, they're not used for producing the output in the final layer, but they are used in the previous layers for the context in all the self-attentions.

Jon Krohn: 01:31:25 Nice. All right. What's your next technical question?

Kirill Eremenko: 01:31:28 Okay. Now this is an interesting and important one for the full transformer. It's called the SOS token, or the start-of-sequence token. If you look at the full transformer architecture research paper, at the bottom of the decoder, it says outputs shifted right. And that might be a little bit confusing at first, what's that for? Well, the reason is that, how do you generate the first word? So, you have those six words, "The cat sat on the mat," going through the encoder, but then the decoder needs to somehow generate that first word, and then it can build on top of that. How does it generate the first word?

01:32:07 And so, obviously, during inference, there is no first word, there is no output that you need to shift right at the start, but during training, you can't show it, the first word. You need to put some placeholder in that place so it knows, it also learns how to generate the first word. And that output shift to right means, basically, that in the decoder side, we preempt all... Any text that goes into the decoder always starts with an SOS token, which stands for start-of-sequence. And basically, this token is treated like any other token. It goes through the input, embedding, positional encoding. It goes through the self-attention, even though it's kind of pointless, but it still follows the same architecture. And then it gets to the cross-attention, then it gets to cross-attend the English sentence, as we discussed, and then it'll be able to generate the first word. So, just something to keep in mind that outputs shifted right is a SOS token that is used in the decoder.

01:33:11 That's one technical addition. Okay, so masking, we talked about masking. I guess the other important point, something we talked about was layers. So, when we know it's a decoder-only layer, the layers are stacked on top of each other. But how do layers work in a full transformer architecture? Well, layers are taken in the encoder, the layers are stacked on top of each other, so in the original transformer paper, Attention Is All You Need Paper, there's six layers of encoders, they're stacked on top of each other, and the outputs go into the next layer, into the next layer, into the next layer, and so on.

01:33:50 But then, in the decoder side, there's also six layers, and all of the six layers in their cross-attention, they use the output of the final layer of the encoder. So, it's not like like-for-like, where the first layer is connected with the first layer. The encoder first layer is connected to the decoder first layer, it's not like that. The encoder does all six layers first, and then the output of the sixth layer is fed into level 3B in each layer of the decoder. So, that's another technical thing to know that how, architecturally, it looks like.

01:34:21 And I think that's pretty much it. Oh, there's just one last thing I wanted to say, and that is about bottleneck. Remember, Jon, we were talking about in the previous podcast that LSDMs, they create a bottleneck and they were inefficient because of the sequential processing, but also they put all their input into a bottleneck, and basically they put into one vector, and that vector is a bottleneck, because you're trying to squeeze a lot of information into one vector. So yeah, that's pretty much it. Those are the main technical things that I wanted to discuss on this podcast and make sure that everybody's at least aware of them, because it's important. I think it's important for the general understanding.

Jon Krohn: 01:35:05 Nice. Well, Kirill, it has been an awesome journey. Thank you for coming back. At the beginning of the episode, we said there might be too much to cover of all that we wanted to cover, so there was also the hope of talking about fine-tuning these architectures, fine-tuning new transformer architectures. We'll leave that to another time. So, either Kirill and I will make a decision on whether we should come back and have a whole other long Tuesday episode about that, or maybe some one five-minute Friday. Well, it'll be more than... One shorter Friday episode that isn't quite as long as a Tuesday episode. We will get that content to you.

01:35:48 There's been lots of advances, and I get the impression that Kirill probably knows a lot more about these things than I do these days, but we have done some episodes on fine-tuning in the past, for example, episode number 674, in particular, is about parameter efficient fine-tuning with LoRA, Low-Rank Adaptation, which is one of the most popular approaches or certainly the root of many of the popular approaches today. But yeah, there's lots of cool things to talk about. QLoRA and ROHF and fine-tuning versus the regular training. And so, there's all these kinds of things. It sounds to me like it's probably a whole other Tuesday episode, Kirill, but we can figure that out.

Kirill Eremenko: 01:36:29 One day.

Jon Krohn: 01:36:31 In the meantime, in addition to episode number 747, where Kirill gave us that amazing technical introduction to decoder-only transformers, now we've filled out our knowledge with a complete coverage of how an encoder-decoder transformer works. And I think it's pretty amazing. This does seem like a relatively long podcast episode, but if you think about it, this is over the course of roughly two two-hour podcast episodes, 747 and then now this one today, we've managed to distill almost six months of your knowledge. It's pretty amazing. And on

top of that, if you're interested in getting this information from something more than just an audio-only podcast format like this, Kirill, I suspect you have a good resource?

Kirill Eremenko: 01:37:24 Yeah, of course. Please come check out superdatascience.com/llmcourse, where Hadelin and I have published our Largest Language Models A to Z course, and it's not available anywhere else, it's only available in the SuperDataScience platform. So, you'd need to sign up for a membership and then you can get access to that course and all of our other courses, plus all of the Live Labs that we do. Our goal is to do two labs per month at this stage, plus additional live sessions, as in online live sessions. Yeah, superdatascience.com/llmcourse, come check it out, learn tons about large language models.

Jon Krohn: 01:38:04 I think you're touching on something really cool there as well that is worth highlighting, which is that the superdatascience.com platform is now becoming a really vibrant space for people to be getting live interactive instruction and being able to chat together and get help on technical questions as well as career questions with each other, as well as with luminaries, like yourself and Hadelin and others as well. And so, yeah, really cool that superdatascience.com is now becoming an ecosystem all into itself.

Kirill Eremenko: 01:38:39 Yeah, absolutely. That's our goal and we're gradually building that up, so yeah. Would love to see our listeners there.

Jon Krohn: 01:38:46 Nice. All right, so let's wrap up. As you know, Kirill, as you put in the instructions to me when I took over as host a little over three years ago, you very well know that I mmust ask you for a book recommendation before I let you go.

Kirill Eremenko:	01:38:58	Awesome. Book recommendation. So, last time I talked about The Big Leap, a fantastic book. At this time I'd like to recommend another great book I read a few months ago called The Go-Giver by Bob Burg and John David Mann, I believe. Really nice book about how to accomplish success in life, however you define success, by rather than aiming to grab things, and take things, and win things, and always striving for achievement, doing the opposite by giving people as much as you can in all areas of life. And they talk about five areas or five ways of giving, five concepts that underpin this notion.
	01:39:50	And it's really well written in the sense that it's not just a self-help book, where it talks about the tactics and strategies, it's actually a story. It's a story of this guy who's having trouble at his work and his relationship and so on, and then he meets this mentor and the mentor talks him through these five days of giving and it shows how it transformed his life. It's a very heartfelt book, I enjoyed it a lot.
Jon Krohn:	01:40:14	Excellent recommendation, very much up your alley. You are a big giver-
Kirill Eremenko:	01:40:19	Thanks, Jon.
Jon Krohn:	01:40:20	... and we are all super grateful for all of the content, and resources, and laughter, and everything that you've brought to our lives, we really appreciate it.
Kirill Eremenko:	01:40:29	Thanks, Jon. That's so nice, thank you.
Jon Krohn:	01:40:32	In addition to the superdatascience.com platform, where should people be following you before the next time you appear on the show?
Kirill Eremenko:	01:40:40	LinkedIn is a great place, but I think the place I hang out most is SuperDataScience's platform right now.

Show Notes: http://www.superdatascience.com/759

Jon Krohn:	01:40:47	Nice. All right. Thanks so much, Kirill. I'm sure it won't be long before you're on again. And I'm sure our audience loves the experience as much as I do. Thank you.
Kirill Eremenko:	01:40:56	Thank you, Jon. Thanks. It was great.
Jon Krohn:	01:41:04	All right, I hope you found today's episode to be extremely informative, I certainly did. In today's episode, Kirill filled us in on the encoder structure of the transformer and how it combined with the decoder structure through cross-attention. How encoder-only architectures, like BERT, excel at natural language understanding, while decoder-only architectures excel at natural language generation. And full encoder-decoder architectures give you the best of both, that's highly contextualized natural language generation.
	01:41:29	He also talked about how we need masking during self-attention, because it prevents the model from cheating, from looking ahead to what comes next in the training data during generation tasks, while encoder-only models and cross-attention don't need masking, and so you can take advantage of the full context that's the language before and after a given token with those kinds of models.
	01:41:49	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/759. And for a full video instruction version of the content we covered today and more, you can check out Kirill's LLM course exclusively at superdatascience.com/llmcourse. We've got that link for you in the show notes as well.
	01:42:15	Thanks to my colleagues at Nebula for supporting me while I create content like this Super Data Science episode for you, and thanks of course to Ivana, Mario,

Natalie, Serg, Sylvia, Zara, and Kirill on the Super Data Science team producing another insane episode for us today. For enabling that super team to create this free podcast for you, we are deeply grateful to our sponsors. You can support this show by checking out our sponsors' links, which you can find in the show notes. And if you yourself are interested in sponsoring an episode, don't hesitate to check out how by heading to jonkrohn.com/podcast.

01:42:48 Otherwise, please feel free to share, review, subscribe if you haven't already, and all that good stuff, but most importantly, just keep listening. So grateful to have you listening and I hope I can continue to 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 Super Data Science podcast with you very soon.