[SQUEAKING]

[RUSTLING]

[CLICKING]

RAMA RAMAKRISHNAN: OK. So let's continue the journey we started last time. So what we're going to do, if you remember in the last class, we showed how we can actually build an autoregressive large language model-- a.k.a. a causal large language model-- using this idea of causal encoder, a transformer causal encoder. And then we showed how you can actually take a bunch of sentences and use next-word-prediction and just run it through. And boom, you get GPT-3.

So that's what we saw last time. I want to point out an important clarification/correction, which is that when we work with large language models, unlike when we work with BERT, for instance, when we work with these kinds of causal models, actually, when the contextual embeddings come out, you don't actually have to use ReLU activations here.

You can literally just run it through just a single dense layer with linear activations and then pass it into a softmax. And boom, you're done. So that's how GPT-3 and all these models are trained. And the other thing I want to point out, which may not have been clear, is that what is coming out of this dense layer, this vector is as long as your vocabulary.

Because only then, when it goes into the softmax, you're going to get probabilities which are as long as your vocabulary, which means that you get to pick one word or token out of that entire 50,000-long vocabulary. So I just want to point that out because I think it's easy for us to get a little confused because of this little difference between the way masked language models like BERT work and causal language models like GPT-3.

So now let's continue with-- we know how to build GPT-3. So what about ChatGPT and GPT-2? What's up with them? Why is GPT-3 so famous and not GPT-2? Well, first of all, you folks know the GPT stands for generative pretrained transformer.

Now, like GPT-3, GPT-2 and GPT-1 were trained in basically the same fashion-- predict the next word, same fashion, the same transformer stack, except that GPT-3 was trained on much more data because the underlying transformer stack had many more layers. So it is a much bigger stack, meaning lots more parameters, and therefore you need lots more data to train it well. OK?

So that was really the only difference. The difference was literally one of scale, scale of network and scale of data. And unlike GPT and GPT-2, GPT-3, even though it was trained basically the same way with the same kind of network, it was one of those situations where more became different. There was almost like some of phase change that happened between 2 and 3.

Unlike GPT and GPT-2, GPT-3 could do amazingly coherent continuations of any starting prompt. So for example, if you have this little prompt, which says "The importance of being on Twitter," by Jerome K. Jerome, who was a famous humorist, and then you give it this prompt ending with the word "it," it produces this continuation, which is really strikingly good.

And if any of you have read Jerome K. Jerome and if you read this thing, you'll be like, wow, that actually sounds like Jerome K. Jerome. So, amazing continuations. But the interesting thing here is not so much the continuation, it's the fact that the same prompt, you give it to 2 or GPT, it won't be very good.

In fact, after the first one, two, or three sentences, it'll become incoherent and meander and start rambling. This thing can keep faking it for a longer time. That's the amazing thing. That was unexpected. Researchers did not expect this.

But it wasn't good at following your instructions. So for instance, if you ask it, help me write a short note to introduce myself to my neighbor. This is the kind of thing it will come up with. And you can actually run it yourself. You can actually go to GPT-3 on the Playground. I think GPT-3 is still available in the Playground. If it is, you can actually start running these prompts.

You will start getting garbage very quickly. And the reason, So for example here, "help me write a short note." It says, "what is a good introduction to a resume." Resume, for some reason it's glommed on to resume. I have no idea why. But the reason it's doing stuff like this is because a lot of the training data it was trained on are basically lots of lists of things.

So when you say, for example, "the capital of France continued," it'll say the capital of France is Paris, the capital of Hungary is Budapest, and so on. It just start came up with the list. So it's very list-driven. It thinks that you need to complete some sort of list. That's what's going on here. And so it's not very good.

So it doesn't realize that you're actually asking it to do something specific. So this is the problem when you have an autocomplete thing which doesn't realize what you're asking it. It's an autocomplete. Now, in addition to these unhelpful answers, it can also produce offensive answers, factually incorrect answers, and so on and so forth.

The list of bad things it can do is long. So why does it do that? Why does it produce unhelpful answers? Well, as you recall, it was only trained to predict the next word. It wasn't explicitly trained to follow instructions. So it seems reasonable that if it's simply trying to guess the next word repeatedly, it can't really do anything more.

How can it figure out that there's an instruction that it needs to follow? Unless the training data on the internet was all instructional, which it clearly is not. So, light bulb idea-- let's explicitly train it with instruction data. Let's just train it with instruction data. And so OpenAI developed an approach called instruction tuning to do exactly this.

And this paper is the paper that was the breakthrough. This is what actually put ChatGPT on the map. And it's very readable. So I would encourage you to check it out, if you're curious. And so we had GPT, GPT-2, GPT-3, just bigger and bigger models trained the same way. And then we run into the problem that it can't handle instructions.

So we do instruction tuning to get to 3.5, also called InstructGPT. And then a small tweak after that gets you ChatGPT. And by the way, this step here, there are really two things going on in this, as you will soon see, I'm just calling it instruction tuning just so that I don't have to say some long thing every single time. This is not a consistent piece of terminology, so just be aware of it, that's all.

All right, first step, they got a bunch of people to write high-quality answers to questions. And they created about 12,500 such question-answer pairs. So for example, let's say this was the question-- "Explain the moon landing to a six-year-old in a few sentences." Believe it or not, GPT-3's answer to that question was another question. Because it thinks there is a list of questions it needs to autocomplete.

So it comes up with, "Explain the theory of gravity to a 6 year old." It's like one of those people who, when you ask them a question, they ask you a question back. So what they did is they said, OK, let's create a nice answer to this question. And here's a human-created answer-- "People went to the moon in a big rocket, walked around," blah, blah, blah. Much better answer to that question.

And so once you create these 12,500 question-answer pairs as training data, we just trained GPT-3 some more using next word prediction as before, no difference. So here is the input-- "Explain the moon landing," blah blah blah blah. This is the question. And then we have the answer right there.

And then we take that answer, move it to the right, and just shift it up so that when it finishes "sentences," it needs to predict "people." And then you give it "people," it needs to predict "went," and so on and so forth, just like we saw before. "The cat sat on the mat" became "the cat sat on the," "cat sat on the mat" on the right. Shifted, right?

That's what makes prediction possible and necessary. So that's what they did. This is step one, same as before. And once you do that, it turns out-- this step is called supervised fine tuning-- it really helped. GPT-3, once you supervise fine tuned it, was much, much better at following instructions.

But there's a small problem with this approach. It takes a lot of money and effort to have humans write high-quality answers to thousands of questions. It takes a lot of money. So the question is, what can we do? What is easier than writing a good answer to a question? OK, all right. How about somebody from this side? Yeah?

AUDIENCE: Perhaps writing a question for an answer.

RAMA RAMAKRISHNAN: Oh, that's actually a good one. Yeah, yeah, I like that. So, given an answer, find a question. And while that is not what I'm going to talk about here, that technique is actually used very heavily in LLMs. And so that's great, very Creative. Mark?

AUDIENCE: Thumbs up, thumbs down.

RAMA RAMAKRISHNAN: Sorry?

AUDIENCE: Thumbs up or thumbs down.

RAMA RAMAKRISHNAN: Thumbs up or thumbs down, exactly. Because all of us, everyone loves to be a critic. It's much easier to be a critic than to be a creator. So what do we do? We basically say, let's rank answers written by somebody else, which begs the question, who's going to write those answers? And that's where there's a brilliant answer to that question, which is?

AUDIENCE: Wikipedia.

RAMA RAMAKRISHNAN: Wikipedia.

AUDIENCE: Reddit.

RAMA RAMAKRISHNAN: Reddit.

AUDIENCE: [INAUDIBLE]

RAMA RAMAKRISHNAN: We will just ask GPT-3 to write the answers. It might be crap, but we don't care because we can rank them. So we ask GPT-3 to generate several answers to the question. And how can we generate several answers? Because we can do sampling. We can do sampling.

The fact that we had these stochastic outputs because of sampling is now a feature, not a bug. We create lots of different answers to the question. We feed it a question, get three answers out. Just run it three times, get three answers out, with a nice temperature of 1 or 1.1 or something so that it's nice and random.

And then we literally have humans just rank them, do the thumbs up, thumbs down, just rank them from most useful to least useful. So this step is a Step 2 of instruction tuning. So OpenAI collected 33,000 instructions, fed them to GPT-3, generated answers, and had humans rank them. Once you do this, you can assemble a beautiful training dataset.

And so basically, what we have is that we have an instruction. And let's say we have just two answers, A and B. And in practice, you can have many, many answers, which we rank. But just for simplicity, I'll go with Mark's thumbs up, thumbs down answer, which is let's assume we only have two answers to every question.

And the human has said, I prefer this to that. That's it. So we have a dataset now where the data point is instruction, preferred answer is A, the other answer is B. Yeah?

AUDIENCE: The thumbs up, thumbs down technique that we were talking about before, is that why when we're using ChatGPT now, we also used thumbs up, thumbs down? Because it's using all the answers to train?

RAMA RAMAKRISHNAN: Exactly right.

AUDIENCE: OK.

RAMA RAMAKRISHNAN: Yeah, all the models have the thumbs up, thumbs down stuff going on somewhere. They are all collecting data for this step.

AUDIENCE: Got it. Thank you.

RAMA RAMAKRISHNAN: Yeah. It's sort of the old adage, right? If you're not sure who the product is, you're the product. So it's one of those things. Yeah?

AUDIENCE: So if we understand correctly, when we see thumbs up, thumbs down, it does mean that ChatGPT is going to trade on our data, right?

RAMA RAMAKRISHNAN: Unless you opt out, yeah. So if you actually go to ChatGPT controls, there is something called data controls or something. You can toggle it off. But I think when I last checked, if you toggle it off, you lose your chat history. So they have hobbled that feature to prevent people from setting it to off as much as possible. Yeah, clever.

But you can opt out. And if you use the API, as opposed to the web interface, you're automatically opted out. So you have to deliberately opt in. And if you use the versions that are available through Microsoft Azure and so on and so forth, there are all kinds of very safe controls and stuff like that. In fact, I think the Microsoft Copilot license that MIT has, I think the default is opted out.

So to go here, once you have this data point, you can build something called a reward model. And this is a very clever piece of work. So what you do is-- you have an instruction, you have a preferred answer, and you have the other answer-- you feed it to a network. You feed it to a network. This is just a nice language model. It's just a language model.

And the language model produces a number, which measures how good this thing is. How good an answer is this to that particular instruction? So you get a rating here. You get a rating here. And then what you do is you run it through a little loss function, which essentially encourages the model to give higher numbers to the better answer.

It's the same model. You just run the question on the first answer, question on the second answer, you get these two numbers. And then initially those numbers are just random. But then you tell the model, hey, this is the preferred thing. Make sure the preferred answer's rating, the R value, is higher than the other number. Because more is better, higher is better.

And this thing is just the sigmoid here. You basically take the difference of these two things. Do a sigmoid, take the logarithm. And you can actually convince yourself afterwards-- and I encourage you to do that-- to check for yourself that if we actually give a higher number to the better answer, the loss will be lower.

And since we are minimizing loss, we are essentially training the network to try to give higher ratings to better answers. That's it. So that's the approach. Did you have a-- yeah, Ben.

AUDIENCE: So you could imagine training the model on only the good answers, is the idea of having both that the model's actually learning what makes good--

RAMA RAMAKRISHNAN: Correct, exactly. Much like if you want to build a dog-cat classifier, you have to show pictures of both. Yeah?

AUDIENCE: So I understand the feedback mechanism of thumbs up, thumbs down. But there are a lot of times when the popular response is not the accurate one. So is there a way that they actually have a layer to correct?

RAMA RAMAKRISHNAN: Yeah, good question, Swathi. So as it turns out, all these companies like OpenAI, they have a huge document, 100, 200 pages long, very, very bulky document, which instructs and teaches the labelers, the rankers how to rank these things. So they have to follow these very strict guidelines to precisely handle strange corner cases and things like that.

And that document is on the web. You can dig it up. And it's actually very instructive to read through it. I think they put it out on the web because they wanted to convince people that they're going to inordinate trouble to make sure the rankings are actually good. Do you have a question, Colin? No.

OK, all right, so back to this. And how do you train this thing? SGD. Because you have a network. It's coming up with an answer. You have some way to know if that answer is good or bad. Better answers have lower loss. Back propagation through the network, keep updating the weights, and, boom, you're done.

And once you do that, this reward model can provide a numerical rating for any instruction-answer pair. You just give it an instruction, you give it an answer-- could be a crappy answer, good answer-- it just tells you how good it is, which means, in this case, for example, maybe it's going to give you a nice number, 1.5, which is 1.5 for this answer. But then a better answer comes along, 3.2.

What we have done by doing this whole thing, this modeling, is that essentially we have learned how humans rank responses. Because we can only have humans rank responses for some finite number of questions. What we really want to do is to automate that ranking process so that we can just do it for tens of thousands of questions really fast.

So we have essentially built a model of how humans rank things, which is beautiful. A lot of the stuff here is all very self-referential, which I find very elegant. Anyway, so this can be used to improve GPT-3 even further. So we take the instruction as before, we feed it, it gives you some answer. And then we feed this instruction and the answer to our newly-minted reward model. It gives us a numerical rating.

And then this is the key step. We take this numerical rating. And then we use this rating to nudge the internal weights of GPT-3 in the right direction. This nudging uses a technique called reinforcement learning, which, just in the interest of time, we can't get into in this lecture. But that's a technique you use to nudge these things in the right direction.

So that's what we do. That's reinforcement learning. We nudge it in the right direction. And OpenAI did this with 31,000 questions. Nudge, nudge, nudge, nudge, nudge, nudge. And when you do that, you get GPT-3.5/InstructGPT. OK? That's it.

By the way, this step here is called reinforcement learning with human feedback because we use reinforcement learning. And since humans rank the answers which led to the building of the reward model, we get human feedback. That's reinforcement learning with human feedback. Yeah?

AUDIENCE: I have a question regarding the type of questions that they are using. I can imagine maybe they are very simple questions to answer. Because I'm thinking now you can ask GPT, for example, respond to this as a pirate or something like that. It's going to be harder to train if you have a bunch of questions that are having small instructions and then there is the question.

RAMA RAMAKRISHNAN: That's a good question. So the quality of the questions in the dataset clearly is a big factor. Because if you have simplistic questions, it won't be able to handle complex questions later on. So it's a good question. So that actually begs the question of where did they get these questions from?

So they actually got it from their API. So people were asking GPT-3 on the API before it became 3.5. The API was already available, fully available, commercially available. A lot of people were building products on it already by then. And so they collected all those questions and filtered them for quality.

And that was the question set that they used. And then they judiciously added to it with human-created questions. But they couldn't do a lot of that because it's expensive to do that. But collecting stuff that somebody else is asking your API already-- very easy. Yeah, Tomaso?

AUDIENCE: This might be more of a philosophical question, but the human bias that's present in the small subset of human labelers that they've chosen gets eventually compounded in this model that we often consider as the source of objective truth.

RAMA RAMAKRISHNAN: Yes. Yeah, that's very true. I think the reward model has probably very faithfully learned all the biases of the human labelers, which is why they have these very complex frameworks and guidelines to try to prevent the bias from happening, to mitigate it. So for example, they might give the same question and set of instead of possible answers to many, many different labelers. And only if people pick the same ranking, they might use it.

So that at least inter-labeller bias can be minimized. But if everybody is biased in the same direction, it won't protect you against that. So in general, there's a whole work that's being done to try to de-bias these things and build them without too much bias in them. It's like a whole world unto itself, which we just don't have time to get into.

AUDIENCE: Depending on the medium that's being returned by these models, would there be more than one reward model? Because isn't that what Gemini--

RAMA RAMAKRISHNAN: Sorry, would there be more than one what?

AUDIENCE: Reward model. Because isn't this what Gemini is running into issues with right now with their image generation, is the bias that they hide?

RAMA RAMAKRISHNAN: Yeah, so the Gemini business that's going on, it's unclear what's causing it. It may be in this step, maybe they were a little overzealous in preventing certain things from happening. Some of these systems, they will actually intercept the question that you ask and then route it differently based on what they sense is sitting around in the question.

So there could be pre-processing, post-processing, a lot of stuff that goes on. So unclear to me where in the pipeline. And it could be more than one place these things may be entering. So yes, here may very well be where it actually enters a situation where people are told, if you see any of this kind of answer, down rank it or don't up-rank it.

And then it learns that ranking very faithfully and then proceeds to apply it where it should not be applied. So that does happen. Jocelyn, you had a question.

AUDIENCE: I think I still don't totally understand why when I ask Chat-GPT a question, even in a lengthy response, it doesn't wander away from the topic that I'm asking about. And so understanding that it's predicting each word, it's taking a random walk from one word to the next in some sense.

RAMA RAMAKRISHNAN: But each word it utters now becomes part of the input to the next word it utters.

AUDIENCE: Right.

RAMA RAMAKRISHNAN: So it's not truly a random walk in that sense. So the next step is not independent of the previous step. It depends on the journey so far. So it's going to try to be very consistent with the journey so far.

AUDIENCE: OK. Does this part, fine-tuning it on these question and answer sets, does this play some role in it being able to constrain itself and not meander away?

RAMA RAMAKRISHNAN: I don't think so.

AUDIENCE: OK.

RAMA RAMAKRISHNAN: I think this is more to make sure that the weights generally tend to produce the right answer. Now, one of the things that is possible is that let's say I'm a ranker and I'm looking at a few different answers, I have to figure out if the answer is helpful, if it is accurate, if it is non-toxic, things like that.

And part of the rubric for evaluating these answers could be their coherence. So it could also be that they are saying, short, coherent answers are better than long, coherent answers. But once you adjust for length, maybe coherence is more important. There could be any number of these things, so it could play a role in that.

AUDIENCE: So just one small follow-up. So in other words, when it's learning from these question-and-answer pairs, it's able to look at the whole response and learn something about the whole response rather than just one word at a time, right?

RAMA RAMAKRISHNAN: Correct. Yeah, the entire question is being ranked. Correct, correct. Yeah?

AUDIENCE: On a related note, when it's generating a new word on topic, does the attention pertain to that entire prior text? Or can you have traveling attention? So last five word attention--

RAMA RAMAKRISHNAN: Yeah, the short answer is, yeah, you can. It's called sliding-window attention. It can be done. They typically tend to do it not so much because they want to focus more on the recent words, but more because it actually makes it very compute efficient. That's why they do it. So it's called sliding-window attention. You can Google it.

AUDIENCE: So normally it's full attention?

RAMA RAMAKRISHNAN: Normally, default is full attention. So that's what they did. And by the way, as I think you pointed out, that's exactly what's going on. You're training the reward model with these thumbs up and thumbs down. Hold on the questions. And so if you give it the same question to GPT-3.5, InstructGPT, amazing answer, like night and day difference. Amazingly good answer.

And then to go from 3.5 to ChatGPT, they basically followed the exact same playbook, except that because they wanted to have a chatbot-- meaning something that could carry on a question-answer, question-answer pair. As opposed to just a single question and answer, they wanted question-answer, question-answer, like conversation.

They trained it on conversations. That's it. Instead of training it on instruction-answer data, they trained it on instruction-answer, instruction-answer, instruction-answer, a sequence of such things, which are strung into a conversation. That's it. That is the only difference to go from 3.5 to ChatGPT.

Now ChatGPT, given you do that, it's giving you a much nicer response. And then you can ask a follow-on question. Can you make it more formal? Boom, it gives you a nice response. Because now it knows about conversations because it's been trained on conversational data. So that's it.

That's how they built ChatGPT. And all the things we are seeing later on are all continuations of this sort of approach. So I'll pause for a couple of quick questions. Swathi, you had a question. Then we'll go to you and then to you. Yeah?

AUDIENCE: So does that make a difference, if a new question-answer pair or a new training data comes early in the building of the model or later in the building of the model, 7 billion parameters. Does that make a--

RAMA RAMAKRISHNAN: You mean the order of the questions, does it matter?

AUDIENCE: So I might have, let's say 5,000 images to start with. Now, after my model is trained and developed, now I have a new use case that has come in. Will that make a difference if I set it in now?

RAMA RAMAKRISHNAN: So if you have a new use case for which you want to essentially adapt the model, there's a whole set of techniques you use, which is going to be the next section.

AUDIENCE: But it's not [INAUDIBLE]

RAMA RAMAKRISHNAN: Yeah, because what you have out of the box is just a generally good chatbot. It knows about a lot of stuff because we've been trained on those 30 billion sentences. It can answer a lot of questions reasonably well using common sense and world knowledge.

But any specific use case like medical and so on and so forth, it may not know. So you'll need to adapt it to your particular unique situation. And that's coming. All right, yes,

AUDIENCE: What determines if a whole conversation is ranked positively versus just a specific answer? proliferating in your question... If the first answer doesn't get a positive response, but then after the follow-up, the second one does, does that indicate--

RAMA RAMAKRISHNAN: Correct, exactly. See, if you're a human and you read the transcript of an exchange between two people and I'm giving you two exchanges, which all start with the same question, you'll be able to assess which one is a better transcript. That's basically what's going on. There was something here, right? Yeah?

AUDIENCE: So I was wondering, when you ask Chat-GPT a question, very often it sounds kind of robotic. You could tell that something was written by ChatGPT, not by an actual person. Do you think that comes from the reinforcement learning part? Or where do you think it comes from in this situation?

RAMA RAMAKRISHNAN: It's a good question. I don't know. Because I know that part of the evaluation, the ranking rubric that are used, is to favor responses which sound more human-like more than robot-like. So if anything, I'm hoping that reinforcement learning would actually make it sound more human-like because the rankers would have prioritized that.

So if it still comes up with robotic stuff, it's something else that's going on. Maybe there are a lot of text on the internet is not literature. It's just people writing some crap. So it could be that. Yeah?

AUDIENCE: How much of this instruction tuning or conversation tuning is happening in real-time within a conversation?

RAMA RAMAKRISHNAN: None of it.

AUDIENCE: None of it. So as you give feedback to the model, it's just basically regenerating it. I don't like that answer, come up with something else.

RAMA RAMAKRISHNAN: No, it's not doing it in real time. Basically, whatever signals you're giving it with these thumbs up, thumbs down business, that gets added to the training logs. And they periodically will retrain it. So by the way, this is instruction tuning in a nutshell. And I want to point that out-- and you don't have to read the whole thing-- but just to quickly point out this was where we had to have human involvement.

The first step, writing a lot of responses to these questions, and then ranking the answers. So these two are still human labor intensive. Now, it turns out you can actually use helper LLMs to automate this too. This is not what OpenAI did in the beginning with ChatGPT. But now you can do it this way because there are lots of really good LLMs available for you to automate many of these things. We don't have time, but if you're curious, I had a little blog post on this. Check it out.

OK, so now we come to the question of, well, if you want to take a base LLM like GPT-3 and make it useful and respond to instructions, we have seen that we had to adapt it with high-quality instruction-answer data using Supervised Fine-Tuning and Reinforcement Learning with Human Feedback.

That's what made GPT-3 actually useful and became ChatGPT. By the same token, this holds true more generally. If you want to take a large language model and make it useful for a medical use case, a legal use case, or some other narrow business use case, you have to adapt it with business domain-specific data.

And so let's look at techniques for doing so. All right, so adaptation is the rough name for the process of taking a base large language model and tailoring it for your particular use case. And so there is this ladder of things you can do. And we're going to look at every one of them.

So you can do this thing called zero-shot prompting, which is just you literally ask the LLM nicely, clearly, what you want. And maybe it'll just give it to you. And this is the use case we're all used to in the web interface. You can also do something called few-shot prompting where you ask it something, and you also give a few examples of the kind of things you want.

And that helps it a great deal. And then there's this thing called retrieval-augmented generation and fine-tuning. And we'll look at all of them. And I'll explain all these things as we go along. So let's start with zero-shot prompting. By the way, the word "shot" here is a synonym for "example."

So zero example prompting-- you literally ask in the prompt what you want without giving even a single example. And so let's say we want to look at product reviews and build a detector to figure out if the product review contains-- not sentiment, that's kind of boring-- whether it contains some description of a potential product defect or not.

And so here is something I actually pulled off Wayfair, with apologies to Wayfair. It says here the curve of the back of the chair does not leave enough room to sit comfortably. Sounds like a kind of a defectish kind of thing. So back in the day, you'd have collected all these reviews and built a special-purpose NLP-based classifier to figure out defect, yes or no.

Here, you can literally just feed this thing into GPT-3 and ask it, tell me if a product defect is being described in this product review. And then, "the curve of the back," and, boom, boom, boom. And then it comes back and says, yep, that's a product defect. So this is zero shot. You just ask a question, you get the answer back.

And it actually works remarkably well. And the better models, the bigger models tend to be much better than the smaller, simpler models for doing zero shot. All right, now when you adapt an LLM to a specific task, obviously you need to carefully design the prompt. As you folks know, this is called prompt engineering. And we're not going to spend much time on prompt engineering, except I just want to give a simple example.

So if you actually ask ChatGPT this question, "What is the fifth word of the sentence?" Very often it will give the wrong answer. It's very strange why it can't get this question right. It's a very simple question. So, the fifth word of the sentence is "is." Sometimes it gets it right, but very often, it'll get it wrong. But now you can do a little prompt engineering and it'll always get it right.

So for example, you can say, I'll give you a sentence. First, list all the words that are in the sentence. Then, tell me the fifth word. OK, here is the sentence. Boom, boom, boom, it gets it right. So it's an example of, you can help it along by being very, very prescriptive as to what you want it to do and break down all the steps. Don't make it guess things. It does a great job.

And there are lots of other tricks people have figured out over the last couple of years. For a long time, this was pretty hot, where you say, "let's think step by step." You give it a question and say, let's think step by step, it actually gives you a better shot at giving you a good answer back, an accurate answer back. Now, this kind of thing is actually already baked into the LLMs.

So when you ask a question to ChatGPT, your question, your prompt gets appended to what's called the system prompt. And the whole thing goes into the LLM. You never see the system prompt. And the system prompt is telling ChatGPT, think step by step, take your time, don't blurt out an answer, stuff like that. And you can just Google it. The system prompts have been jailbroken. You can find it on the web.

And this is funny. This came out maybe a month or two ago. It says, apparently, "Take a deep breath and work on the problem step by step" works better than saying, "Work on it step by step." And then more recently, I literally read this two nights ago, apparently, if you tell it, if you have a math or a reasoning question, you tell it you are an officer on the Starship Enterprise, now solve this problem for me, it's more likely to get it right.

[LAUGHTER]

Go figure. Thomas?

AUDIENCE: I read two more that work super funny.

RAMA RAMAKRISHNAN: Yeah?

AUDIENCE: One, "I will tip you if you solve this."

RAMA RAMAKRISHNAN: Correct.

AUDIENCE: And the other one was an answer was, "I cannot do that" for output. And it was, "I tried it on Gemini and it was able to solve it."

RAMA RAMAKRISHNAN: Nice.

AUDIENCE: And both, like back and forward, "ChatGPT was able to solve this. Can you solve this?"

RAMA RAMAKRISHNAN: Yeah, very good, excellent. One of the things just on that, let's have some fun. You can say I'm going to tip you $1,000 if you solve this, it says right. So this person apparently kept using this tip. And at one point, it says, you keep promising me tips, you never give me the tip. So I'm not going to solve this problem for you.

[LAUGHTER]

Yeah. And there are many prompt engineering resources. This one that came out a couple of weeks ago, I thought was pretty good so I just put a link to it here. So now let's look at few-shot prompting, where you give it a few examples.

So here, let's say you want to build a grammar corrector. So what you can do is, you can actually give it examples of poor English, good English. You can see. Poor English-- "I eated the purple berries." Good English-- "I ate the purple berries." And similarly, three examples.

And then you end the prompt with just the poor English input. And then the response from GPT-3 is the good English output. And it says, fix the error. So this is an example of giving a few examples of what you want. And it just learns on the fly, what you have in mind, what your intention is. So that's that.

Now, the ability of LLMs to learn from just a few examples or even no examples, and just with a clear instruction, this thing is called in-context learning. And that was something that GPT-2 and GPT could not do. That was new in GPT-3 and what they call an emergent capability. It was completely unanticipated by the people who built it. So that's that.

Now let's look at retrieval augmented generation. By the way, this thing is also called indexing sometimes. So it's called RAG, retrieval augmented generation, RAG. The idea of RAG is actually very simple. So let's say that we want to ask a question to a chatbot, but we want the chatbot to leverage proprietary data that you might have.

Maybe it's a customer call support in a call center kind of operation and you have this massive FAQ database, content database. And you want to give that FAQ to the chatbot along with your question so that it can leverage the FAQ to answer the question for you, as opposed to whatever things it has learned previously in its general training.

So can't we just include the entire FAQ, the whole dataset, into a prompt and send it in? Maybe we just take our question, take everything we have potentially relevant to the question, everything we have in the database, just attach it to the question. The whole thing becomes a prompt. Feed it in and say, hey, find out for me. Can't we just do that? Theoretically, nothing stops us.

The reason you can't do it is because this pesky thing called the context window. So for any LLM, the prompt plus the output, the length cannot exceed a predefined limit. This is called the context window. Remember the max sequence length we had in our earlier models, where that was the size of the sentence that could be fed in.

Basically, there is a size of the sentence for any of these things. It's called the context window. There are only so many tokens it can accommodate. And since what comes in is what comes out, it is for both the input and the output together. That's called the context window.

And furthermore, when you have a conversation with one of these chatbots, the entire conversation is fed in every single time. That's how it actually remembers what's going on earlier in the conversation. It doesn't have any memory per se. Each time you ask a question, the entire thread is fed in.

So initially you say, what's the square root of 17? It gives you an answer. Initially, you only send in the red stuff. Then, the next question you ask is the first question, the answer, the second question, all of them are fed in. Then, all these are fed in. So with the conversation, you're consuming more and more of the context window as you go along.

So can you imagine taking a whole FAQ, asking a question, and saying, well, I didn't mean that, I wanted something else. And before you know it, boom, you've blown out the context window. It's going to come back and give you an error.

AUDIENCE: Can you just make it so you can't [INAUDIBLE]. Or does it take specific windows of it?

RAMA RAMAKRISHNAN: Yeah, so there is a whole research cottage industry around when your thing is longer than the context window, what do you pick?

So the simplest case is you have a moving window. If you have 1,000 tokens, you just look at the last 1,000 tokens. But there are some cleverer schemes where you can actually take the first stuff that is outside the window that doesn't fit into the window and use another LLM to summarize it for you. And then you attach it to your current prompt. I know, it gets crazy.

OK, so for all these reasons, we need to pick and choose what we can send to answer a particular question. So what we do is, since we can't include the whole thing, we first retrieve the relevant content from the database or the FAQ and then send it to the LLM along with the question we have. So retrieval augmented sequence generation, that's what's going on. Make sense?

And so pictorially, basically, what we do is, let's say that this is our external set of documents. Think of it as an FAQ. And then we take the FAQ, and imagine for each question and answer, we take each question and answer in the FAQ, and then we treat it as its own little unit of text. And then we actually calculate a contextual embedding for each of those question-answer pairs.

Remember, we know how to do contextual embeddings. It's a piece of cake at this point, right? You folks know how to do contextual embedding. Run it through something like BERT, you're done. You get a contextual embedding. So you get embeddings for all the things that are in your FAQ.

And now when a new question comes in, what you do is you take that question and you calculate a contextual embedding for that too. And then what you do is you then look to see which of the FAQ elements you have, which of those chunks are the most similar to your question. And then you grab the ones that are the most similar and then pack it into the prompt and send it in.

Maybe you have 10,000 questions, but you can only accommodate 5 of them in your prompt because the context window is very small. So you pick the 5, what you think is the most relevant content to your particular question, and then you feed it in. That's the idea. That is retrieval augmented generation. Yep,

AUDIENCE: So does this tie-in, for example, if I were to prompt and say, help me work on my startup pitch, but do it in the voice of Steve Jobs. Is it then going out there and reducing the subset of data to things that have been written by Steve Jobs? And then it's kind of generating its response based--

RAMA RAMAKRISHNAN: Not as a default, not as a default, typically, because there's a lot of Steve Jobs stuff on the web. It was just using that because it's all part of its pre-training data. But this tends to be more useful for very targeted applications where you don't expect it to know the answer because it is not in the public internet. It's your proprietary data and you wanted to use that proprietary data. And this is how you do it. Yeah?

AUDIENCE: [INAUDIBLE] ..certain information sure like that will there be some loss?

RAMA RAMAKRISHNAN: Sure.

AUDIENCE: Do you think that there will be some loss?

RAMA RAMAKRISHNAN: There will be some loss. Because you have to figure out how to chunk it. Maybe you have a 300-page PDF, and then maybe you look for each section and make it a chunk.

Maybe you look for each paragraph, make it a chunk. Again, there is a whole empirical cottage industry of techniques for doing these things, better or worse, depending on the use case and so on and so forth. But the conceptual idea is chunk and embedding.

AUDIENCE: Chunking in the middle of [INAUDIBLE].

RAMA RAMAKRISHNAN: Yeah, in fact, we're going to do it ourselves in the Colab right now. Yeah?

AUDIENCE: Can you give more weight to a particular chunk?

RAMA RAMAKRISHNAN: [LAUGHS] So in the default implementation, no. But in some sense, by picking the 5 most relevant chunks from 10,000 chunks, you're giving the other 10,000 minus 5 chunks a weight of 0 and these a weight of 1. So in some sense, you're weighting it. Yeah?

AUDIENCE: I'm just curious how much structure you have to have with an external document? If you're a hospital or something, do you have to do a bunch of labelling?

RAMA RAMAKRISHNAN: Pre-processing? No, you just need to make sure it's relatively clean. But you will see in the Colab that it can be kind of crappy and it still works. Yeah. Because there is so much crap on the internet it's been trained on already.

All right, so let's look at the Colab. By the way, retrieval augmented generation is, in my opinion, the most prevalent business application of LLMs that I've seen up to date. And there's a huge ecosystem of tools and vendors and so on and so forth. I'm going to skip through the verbiage here.

So you have to install the OpenAI library and this thing called tiktoken, which we'll get to in a bit. I've already installed it before class because it takes some time. So I'll just make sure all these things are already-- phew, good. So we don't have to wait for this. So we imported pandas, as before.

And you could read through these things because basically I have an OpenAI token that I have to use, a key, rather, key, API key. And I'm not showing you the key, obviously. I have to remember to delete it before I upload the Colab. You have to get your own key to make it all work, but the instructions are here.

So we're going to use GPT-3.5-turbo to demonstrate RAG. So I give the name of the model. And then OpenAI also has a whole bunch of different models which can be used for, you can feed it a sentence or a chunk of text. It'll give you a contextual embedding out. It's like a nice little API.

You don't have to use your own BERT and so on and so forth. You can just use the OpenAI embeddings. Obviously, you have to pay OpenAI every time you make a request, but it's really, really cheap at this point. Yeah,

AUDIENCE: [INAUDIBLE] question. By reading the proprietary data, because a lot of companies are like, we need to invest in our own LLM because we don't want our data to be going out. In this kind of a context, how good is the cybersecurity or the compliance and legal stuff?

RAMA RAMAKRISHNAN: I think each vendor has their own set of rules and contractual commitments they're willing to sign up for. So you just--

AUDIENCE: If you use the data here, does this go into the company [INAUDIBLE]? No?

RAMA RAMAKRISHNAN: But the vendor gets to see it.

AUDIENCE: OK.

RAMA RAMAKRISHNAN: Meaning the vendor systems get to see it. But do the vendors' employees get to see it if they need to? Unclear. Those are all the legalese, nitty-gritty you have to worry about. The other thing you can do is you can actually just download an open source LLM and do it all within your own premises.

AUDIENCE: Oh, OK

RAMA RAMAKRISHNAN: That's totally possible to do. In fact, I probably won't have time today. I have a whole section on how do you actually do a fine-tuning with an open source LLM, which I'll do a video if you don't have time. And so this model, this embedding-ada-002 is the name of the OpenAI model that actually gives you contextual embedding. So we're going to use that.

So the use case here is that we want to ask the LLM, we want to create a chatbot which can answer questions about the 2022 Olympics, like random questions you might have about the Olympics. So let's first ask it this question. We'll ask it about the 2020 Summer Olympics. That's the query.

And then this is the API request we have to make. And you can read through it. I have a link to the documentation here as to how it works. And then it says that "Barshim of Qatar and Tamberi of Italy both won the gold." And you can actually fact check this. It's actually accurate. It's correct.

So now let's change the query and ask it about the 2022 Winter Olympics. And why '22 versus '20 will become clear in just a moment. So, "Which athletes won the gold in curling in the '22 Olympics?" And it says, "The gold medal in curling was won by the Swedish men's team and the South Korean women's team."

If you fact check this, it turns out, wait for it, Sweden won the men's gold, yes. South Korea did participate, but Great Britain actually won the Women's Gold. So it got it wrong. So it sounds like GPT-3.5-turbo could use some help. And now one of the things we can do is, the thing is, the reason why GPT-3-turbo didn't know about this is because its training cutoff date was September 2021.

So as far as it's concerned, the '22 Olympics haven't happened yet. Yet, it confidently gave you the wrong answer, as it is often prone to do. And this, by the way, it's called hallucination, where it gives you a very eloquent, confident, wrong answer. Or as some folks have said about another business school that shall remain nameless, often in error, but never in doubt.

[LAUGHTER]

All right, back to this. So one simple thing we can try right off the bat is to tell 3.5-turbo, you can ask it to say, I don't know, if it doesn't know, rather than just make stuff up. And how do you do it? It's very simple. You say in your prompt, "Answer the question as truthfully as possible. And if you're unsure of the answer, say sorry, I don't know."

OK, now here is the question. OK, this is a query, so let's run it through. "Sorry, I don't know." Not bad, huh? So it worked. It's trying to be humble and honest and self-aware and things like that. It's a bit more like a Sloany at this point. Now, the reason, as I mentioned earlier, you can check the cutoff date and you can see it's 2021. Actually, you know what? Let's just open a new tab.

So all these cutoff dates are training data. So 3.5-turbo, this is what we are using-- cutoff date 2021. That's why. So now what we can do is, we can obviously provide relevant data in the prompt itself. We can leading up to RAG here. And by the way, the extra information we provide in the prompt to help it answer a question is called context. That's the lingo for it.

So we can do it. We'll first do it manually. So first we'll use the Wikipedia article for 2022 Winter Olympics. And we tell it explicitly to make use of this context, because telling things explicitly always seems to help. So this is the thing we cut and pasted here, Wikipedia article on curling.

And it's a pretty long article. It's got all kinds of stuff. And it's not even all that cleanly formatted. It's very strange. Look at that. So to answer your question, Spencer, it can be in pretty bad shape and it still seems to work. So now, "Use below article on the Olympics to answer the subsequent question.

If you don't know, say you don't know." So that's what we have. There's the query. And by the way, before I send it into the LLM, this is the actual query that it's going to be sending. I'm printing out the query. Look at how long the query is. "Use the article below and here is the article," blah, blah, blah, blah, blah. Scroll, scroll, scroll.

This is the whole thing. And it keeps on going on. And then finally I say, "Which teams won the gold?" OK, so let's run it. Look at that. Women's curling, Great Britain-- it got it right. Pretty good, right? It had to parse all that crap to get and find the nuggets, so nicely done.

Now, maybe it wasn't super hard because we literally gave it the answer. So let's make it a bit harder. So I noticed that this person, Oskar Eriksson won two golds in the event, two medals in the event. So let's ask if any athlete won multiple medals. That requires a little bit of abstraction.

So, all right, same query, "Did any athlete win multiple medals in curling?" The question has changed. Everything else hasn't changed. Hit it. Let's see what happens. "Yes, Oskar Eriksson won multiple medals in curling. He won a gold in the men's event and a bronze in the mixed doubles." It's pretty cool. Take that Google.

All right, now we come to retrieval augmented generation where instead of doing it manually, obviously, because it doesn't scale, we will do it automatically. And so the thing you have to remember, as I mentioned just a few minutes ago, is that there is a context window for every LLM. And for GPT-4 and 3.5-turbo, the context window is 16,385 tokens. That is the length of the input and the output. So we can't exceed that.

By the way, GPT-4's context window is, I think, up to 128,000 tokens. And Google Gemini 1.5 Pro-- they really need to work on their names-- Google Gemini 1.5 Pro, the context window is 1 million tokens. And in research, they have tested 10 million tokens, so crazy times. All that means is that you can upload entire videos and ask it questions about the video.

All right, to come back to this, what we'll do is we'll only grab the data from the Wikipedia articles, all the articles about the Olympics that are relevant to our question by using pre-trained embeddings. So again, this is the thing we talked about earlier. This is the picture we saw in class.

And the only thing I want to point out is that if you have a particular embedding for a question and a particular embedding for a chunk of text that you have in your database, you have to figure out how similar, how related they are. And for that, we can use what? Dot product or something almost as dot product, which is more easier for us to work with-- the cosine similarity.

We have done cosine similarity previously. I've explained it in class. We're just going to use cosine similarity-- how similar are these vectors? So that's what we're going to do. So the same picture as we saw in class. So first, what we'll do is we need to break up the dataset into sections and then take each section and then run it through the embedding thing.

But fortunately for us, I have code here which actually does it for you manually. You can play around with it later. But OpenAI has already given us the chunked dataset. So we'll just use that because it's just easy for us. And I downloaded it already because it takes five minutes to download. I've downloaded this thing and I've stuck it in a particular dataframe here.

So let's print out five randomly chosen chunks. So you can see here, this is the first chunk, somebody else, somebody else. And look at all this crazy stuff here. The formatting is off. But these are all, basically, paragraphs and sections just grabbed straight from Wikipedia with no cleaning.

Now, we define a simple function to basically send in any arbitrary piece of text into the embedding model and get the contextual embedding vector out. And there is this little function that does that. We are using an embedding model. We send in text, it gives you something.

So let's try it on that. "HODL is amazing!" You should get a vector back. Oh, come on. Don't fail me now. Phew, all right. How long is it? 1536. So how about I say, HODL is incredible, like HODL is amazing? Hopefully, the two vectors will be kind of similar in terms of cosine.

And so to calculate the cosine distance, I use this particular function from SciPy. It just calculates the cosine similarity. And I hit it, so 0.9934. Maximum is 1. So 0.934 means that they're very, very similar, which is comforting because amazing and incredible are obviously synonyms.

So now given a dataframe with a column of text chunks in it, we can use this function on every one of these things to calculate the embedding. And you have a function here that basically does it for you. I'm not going to run it because it takes a long time. But you can run it later on. Just be prepared. Go get a cup of coffee and stuff while it does it.

But happily for us, OpenAI has actually already done this step for us, so we don't have to. So it's already available in this dataframe. So if you actually look at this, you can see here there is a text, and then there is an embedding that's sitting right there right next to it. And these embeddings are whatever-- how long is it? 1536 long, 1536 long vectors All right, so that's what we have.

So now that we have this thing, whenever we get a question, we calculate the question's embedding and then compare calculated cosine similarity with all the embeddings sitting in this dataframe. So to do that we're going to define a couple of helper functions here. You can read through the Python later to understand. It's just basic Python manipulations that are going on.

And so let's just test this function. So basically, we have a little function called strings_ranked_by_relatedness where you give it any input question or text, and then it's going to give you the top five most related chunks of text that it had in its dataframe. So let me just run this thing.

So curling, the things it pulls back, it better involve curling and medals and so on. So this one has a cosine similarity of 0.88. Curling at the 2012 Olympics, that's good. Results summary, medal summary, results summary-- It's all pretty good. Even the fifth one has a cosine similarity of 0.867, which is pretty high. So it's doing the right things. It's picked up curling gold medal with input text and it's picked up the right things from it.

Now let's see what we can do with the original question. So here is a header I'm going to use in the prompt. I'm going to say, "Use the below articles to answer the subsequent question. Answer the questions truthfully as possible. And if you're unsure of the answer, say I'm sorry, I don't know," as before. OK, that's our prompt.

And now here is the thing. We don't want to exceed the context window. So we need to count the tokens we are sending in and the likely number of tokens we're going to get back so that we don't exceed the budget. So we use this package called tiktoken package for this.

And then it just helps you count the tokens. And you can read through this. It's just again, some basic Python for counting tokens. And now what we do is, this is where we actually assemble the prompt. We start with the header. We have the header, which says "be truthful" and all that. Then we say, here is a question that I'm going to ask you.

And then you go in there and keep grabbing Wikipedia articles till the number of tokens in your prompt is exceeding your token budget, and then you stop. When you're about to exceed the budget, you stop because you can't exceed the budget. And that's the whole thing. So here, let's just do tiktoken, run this function.

Now, it turns out, as you saw, we can go up to 1,600 something tokens in the context window. I'm just using three 3,700 as my budget, partly just to show you how to use this thing and also because it's charging my credit card for every token that I'm using. So I'm just being careful. It charges by the token. It's a beautiful business model.

Anyway, so back here. So let's ask the question, "Which athletes won the gold medal in curling at the 2022 Winter Olympics?" Here is the dataframe that you should use. Here is the GPT model. And don't exceed 3,700 tokens. That's the query or the prompt. It's going to compose the prompt now. And this is the whole prompt. Let's just go to the very top. It's really long.

All right, "Use the below articles to answer the subsequent question-- as possible." And boom, boom, boom, boom, boom, it has all these things. It's added a whole bunch of paragraphs from the Wikipedia pages. And then it finally ends with the question, "which athletes won the gold?"

All right, now let's just ask it the thing. And this is just a little function to send stuff into the API. And now we are finally ready to ask GPT the question. Fingers crossed. All right, curling where, Stefania, curling in the mixed doubles. And the team consisting of blah, blah, blah, in the men's tournament.

Oh, interesting. It has actually ignored the Great Britain people completely, I think. Last night it didn't. Welcome to stochasticity. So you can try it. When you try it, it will actually give you the thing. And so let's ask it now a question about the 2016 Winter Olympics, which, by the way, didn't happen.

There were no Winter Olympics in 2016. So if you ask it-- "Sorry, I don't know." All right, now let's change the header so that we don't say be truthful. So we will remove the need for it to be truthful and see what happens. "Which athletes won the gold?"

Oh, now it's telling you about the 2022 Olympics. So it answered an irrelevant question accurately if you removed the need for it to be truthful. So I guess the moral of the story is that, first of all, you can use RAG to grab stuff from vast databases, and it's very heavily used in industry, number one. Number two, you have to be careful about these token budgets and so on and so forth.

And small wording changes in the prompt can actually dramatically alter behavior, which makes it very difficult in enterprise settings to do QA on this stuff. So a lot of care has to go into it. And you have seen examples of, for example, Air Canada had a chatbot which actually gave the wrong advice to a customer. The customer sued Air Canada.

And then the court ruled in favor of the passenger. And then they pulled the chatbot off the website. So you've got to be very careful. I think without a human in the loop checking these answers, it's kind of dangerous, in my opinion, at this current state. Hopefully, it will get better. So there's a lot of potential in it, but you have to be careful.

All right, so this is what we have. And you can actually take this thing here and use it. You can actually take a 1,000-page PDF that you might have or something and then chunk it and use this approach. And I've done it for a whole bunch of different things. It actually works really well most of the time. It will make errors here and there. Most of the time it actually works really well. Yeah?

AUDIENCE: Just a quick question. GPT-4 now lets you upload a PDF and it remains locked. Is it chunking that? Or is it actually ingesting all the--

RAMA RAMAKRISHNAN: No, when you upload something, because GPT-4-Turbo has 128,000 tokens, which means it can accommodate a whole, long bunch of documents. So when you upload stuff, it's not doing any chunking. The chunking we are talking about, you have to do. The LLM doesn't even know you're doing it.

As far as the LLM is concerned, it's only seeing the prompt it sees. And the prompt says, hey, here's a bunch of information, here's a question. Answer it for me using this question and be truthful. That's it. Now when you ask these things a question, which is later than its training data, you will actually see GPT-4 saying, "doing a Bing search" and things like that.

There, what's actually going on is there's a pre-processing step and a program which is doing a Bing search, gathering a bunch of Bing results, taking the top few results, chunking, embedding, backing into a prompt, sending it into GPT-4. And you don't know all this is going on under the hood. So when it's actually thinking and saying "Bing search," this is what's going on under the hood.

Was there a question somewhere here? No. Oh, sorry. Yeah?

AUDIENCE: I have a question about formatting.

RAMA RAMAKRISHNAN: Yeah?

AUDIENCE: So it seems to be able to understand and ignore irrelevant formatting, even when there's colloquial tables not clearly defined tables. And also when it outputs formats, it's able to do it really human-ly. Is that something that it's figuring out through the neural network? Or is it something that's going to be programmed in the header somewhere with the standard--

RAMA RAMAKRISHNAN: There is no explicit programming going on. It's typically because a lot of the question-answer pairs that was used for supervised fine-tuning and reinforcement learning, the better answers with the same badly-formatted input, the better answers were just rewarded, were ranked higher. That's what's going on.

But on a related note, one thing that's very useful is that you can actually ask it to give you the answer back using certain formats, like markdown and JSON and things like that. And by forcing it to adhere to certain well-defined formats, you actually increase the chance of it actually getting the right answer in the first place.

Again, there's a whole tangent here we can go into. But those are some of the things that are part of prompt engineering. All right, so that's what we have here. Back to the PowerPoint.

So that's retrieval augmented generation. And we finally come to fine-tuning. So fine-tuning is when-- up to this point, all the things we have seen don't alter the internals of the LLM. You have not messed around with the weights or changed them at all. You're just using it as a black box. With fine-tuning, you actually will train it further, meaning the weights are going to change.

Now, remember, we take something like a causal LLM like GPT-- and this, I haven't fixed this yet. There is no ReLU here, as I mentioned earlier. Just remember that. And then if you have domain-specific input-output examples, like input and output, you can just train it like this, input and then the shifted output. And that will update these weights, all these weights.

So this is basically fine-tuning exactly like we saw with BERT and so on. And even with ResNet, it's the same sort of thing. That is fine-tuning. Now, before we discuss the mechanics of how to do it, I want to show you a quick example of the usefulness of fine-tuning. Imagine for a second that we want to generate synthetic product reviews from product descriptions.

So we are building some product which can simulate customer behavior in e-commerce. And for that, we need to be able to generate the kinds of reviews that customers might come up with. And writing a lot of reviews is very time consuming. But what you can do is you can get a whole bunch of product descriptions from the internet.

So let's say you ask an LLM, hey, write a product review using this information here, product description here. And it comes up with this-- "Timeless. Authentic. Iconic." Seriously, do product reviewers actually write stuff like this? No. This looks like marketing copy.

This reads like marketing copy because there's a whole bunch of marketing copy on the internet. So it's not good. It doesn't feel like a review. It's not authentic. Here's another example for Urban Outfitters. And it says, "The boxy and cropped silhouette is flattering on all body types." Come on!

[LAUGHTER]

So it's not going to work. So what we do is we fine-tune the LLM. We can take an LLM and we can fine-tune it with instruction, product description, and product review examples. That's what you can do. So for instance, we can take something like this. Let me zoom into this thing.

So it says here, "Write a positive review for the following product." And then this is the description, is the input and the output is, "The best! Carhart is my husband's favorite. They fit well." They feel like product reviews. So you just have to get a few hundred of these product review examples, just a few hundred. And you may not even need that much.

Once you do that, you just do fine-tuning, like I showed earlier-- instruction, input, output. And then you take that output and shift it a bit and make it the actual label, the actual output. Fine tune, fine tune, fine tune, fine tune a bunch of times. Gradient descent, weights get updated. Now you have a new and updated element.

And when you do that, now for the same things, here's what you get. "Write a review. These are the best jeans I've ever owned. I am--" whatever, some details. "I've been wearing them for a few weeks. They still look brand new." It looks much better. It doesn't look like marketing. This is completely fake, by the way. The LLM came up with it after the fine-tuning.

And then we say, "Write a horrible review," because we want to be balanced. "These are the worst jeans I've ever worn. They are too tight here and there. I'm going to return them and try a 30, but I'm not optimistic. I'm going to stick with Levi's." Phew, OK. So these read like real reviews. So just by taking a few hundred examples and fine-tuning it, it completely changes the behavior that you want for your particular use case.

That's the key thing. So for me, the biggest benefit here is that while it took billions of sentences for pre-training the original LLM and then it took tens of thousands of examples to do supervised fine-tuning and RLHF and so on and so forth, to make it work for your narrow business use case, you only had to spend a couple of hundred examples.

That's it. It's amazing. Imagine that if you had to collect like 30,000 examples to make it. Nobody's going to do these things. It's too much work. But a couple of hundred, anybody can do. That's why it's so powerful to fine tune these things. Yeah?

AUDIENCE: You talked about being able to, in industries where you don't want to put some of this stuff on the internet, downloading the pre-trained model and being able to do this on your own. Talking about computer power, some of the computers we have now, the GPUs, I don't know how powerful they are. Are you able to do some of these very small use cases on those types of devices?

RAMA RAMAKRISHNAN: Perfect question, Mike. We're going to get to that. Because the short answer, it's hard. Yeah, just a few hundred examples. But actually trying to fine-tune these big models on consumer-grade hardware is actually not easy. So you have to make certain tricks and simplifications, which is the next topic. Yeah?

AUDIENCE: Is fine-tuning always supervised? You need those pairs? Or could you do it if the company has less structured data?

RAMA RAMAKRISHNAN: No. The thing is, it depends on whether you want to make it generally smart about the company's business details, in which case, you can just take a whole bunch of text and just do a next-word prediction on it. It's going to get smarter about generally things. But it doesn't mean it's going to specifically follow your instructions on your particular business problem. So if you want it to follow instructions, you need supervision.

OK. All right, these are great reviews. So for small LLMs like GPT-2, fine-tuning isn't difficult, to go to your question. You can actually do this with small models. So for example, Google had this thing called Gemma, which came out recently. It's a small model, like 2 billion parameters or something, if I remember. It's the smallest one. Thank you.

Those things will typically fit into one GPU and you can fine tune it. You still need GPUs, just to be clear. They will actually fit into one thing. But if you want to use a larger model, it won't fit. So to make this work, you have to do other things. And that's what we're going to talk about now.

But there's a family of models called Llama, Llama-2. These are open source LLMs. And they are widely used for fine-tuning. Because you can just download the model and just do whatever you want with it. It's open. It's not strictly open because there are some footnote considerations you have to worry about. But for most purposes, it's open enough, in my opinion.

And so let's see how hard it is to build the biggest model in this family, which is the Llama-2 model with 70 billion parameters, OK? 70 billion parameters. So first of all, the model is gigantic. So 70 billion parameters, each parameter is, let's say, we store it in 2 bytes per parameter. And then each of these parameters actually, we will need a multiplier on each parameter to store various details about how the optimization is done.

We won't get into the details here. The one thing I do want to point out is that this 3 to 4 should really be 1 to 6. So I didn't have a chance to change it this morning. But the point is that it's going to be a huge model. So even with this number, it's going to be like 420 to 560 gigabytes, just to hold the model in memory and manipulate it.

And so if you use a GPU like an A100 GPU or an H100 GPU, which are all NVIDIA GPUs, each of these things typically has 80 gigabytes of RAM memory. So we need between 6 and 7 to accommodate this thing, 6 to 7 GPUs just to accommodate this thing. So that's the first problem. The model is big. Just to hold it and work with it, you need lots of GPUs.

The second problem-- Llama-2 was trained on 2 trillion tokens of text, 2 trillion tokens of text. So these GPUs can process about 400 tokens per GPU per second. By process, I mean the forward pass through the network. And so if you actually use 7 GPUs with all this thing, it's going to take you 8,000 days.

Let's say we want to do it in about a month, you need 2,048 GPUs at this cost of $2.50 per GPU per hour. This will cost you $4 million. And we would expect the actual cost to be a lot higher than this because this is very optimistic. It assumes you just do one pass-through and you're all done.

In general, you'll make some mistakes, you have to do a bunch of times, and so on and so forth. So this is an overly optimistic estimate, and that is $4 million. So you need lots of GPUs and you need to spend a lot of money for it.

Now, what can we do with fewer resources? First of all, you need to reduce the size of the dataset. The second thing is you want to reduce the memory required. So we can ideally do it in many fewer GPUs, hopefully even one GPU, literally on Colab.

Now we have good news on the data front. Because as I mentioned earlier, while it takes a lot of data to build these models, to fine-tune them for your specific data, your use case, you may just need a few hundred examples. It's no problem at all. So the data for fine-tuning is actually not a problem.

Only for building it in the first place, it's a problem. In fact, there's this famous Alpaca fine-tuning dataset. It has 50,000 instruction-answer pairs. And so for that, way less than the 2 trillion tokens. And that can actually be done in about 20 hours. You can fine-tune a 50,000 example, fine-tuning dataset, you can fine tune it with just 20 hours. OK? Tomaso?

AUDIENCE: Could Microsoft's 1-bit model drastically reduces the amount of compute we need?

RAMA RAMAKRISHNAN: Yeah, there's a whole bunch of approximations and simplifications to make all these things fit into smaller GPUs and so on and so forth. And that's one of them. So the short answer is, yeah, there are many possibilities. And we have to very carefully look at them.

Because every one of these simplifications, it will cost you something in terms of accuracy and the ability of the model to do what it needs to do. So there's always a trade-off you have to worry about. So for folks who are interested, there is this whole field called quantization, LLM quantization. Google it. And that's an entry point into that whole area.

Now, how do we reduce the memory required so that we can process the data using fewer GPUs-- ideally, just one GPU on Colab? So if you look at what actually consumes memory, you have all these model parameters. Let's say, 70 billion parameters times 2 bytes each, 140 gigabytes. Gradient computations is another 140 to hold the gradient.

And then the optimizer state is 2x. And as I mentioned earlier, it could be between 1 to 6x as opposed to 3 to 4x. But we'll just go with these numbers for the moment. And so the total is 560 gigabytes, if you just naively want to use it. So it turns out you can't do anything about that. It is just for 140.

But by using a trick called gradient checkpointing, this whole thing can actually be squashed close to 0. Basically, you say, hey, I don't mind it running longer, but I don't want to use as much memory. And that trick is called gradient checkpointing. We won't go into technical details. That can go to 0.

But then this thing here, the optimizer state, turns out even this can be squashed very close to 0. And that actually was a breakthrough from maybe a year ago. And so to do that, what we're going to do is to say, look you know what? There are a whole bunch of weights here, but we're only going to take those matrices inside each attention layer.

And we're going to only look at those matrices. We're going to freeze everything else. So we're going to take only a small set of parameters, unfreeze them, and update them, and see if it's any good, if it actually gets the job done, instead of unfreezing everything and updating them.

And so if you look at the weight matrix, let's say the key AK weight matrix in Llama-2, this is a roughly 8,000 by 8,000 matrix, which means that there are 64 million parameters inside each of these matrices, 64 million. If you imagine this matrix AK here, and let's say, thought experiment, you do the fine-tuning and the numbers have changed as a result of fine-tuning.

Then you can imagine that the resulting matrix is just the original matrix you had plus just the changes. The original plus the changes. And we call the changes delta AK. And of course, in general, this change is also going to be a 64 million matrix, 8,000 by 8,000.

So the question is, can we make this change matrix smaller? And to make it smaller, it seems reasonable because a fine-tune will only make small changes to just a few weights. It's not going to change everything. By definition, a couple of hundred examples, you do some fine-tuning, hopefully a few weights are going to change. And maybe they won't change a whole lot.

So the key insight here is that maybe we can force this change matrix to be kind of simple and get the job done. And it turns out you can. And what you do is you can think of this matrix as really coming from two thin, skinny matrices, which, if you multiply them, gets you the original matrix. And I'm not going to get into the mathematical details here.

This is called a low-rank approximation. But the point here is that you can take two very small matrices. And if you multiply them the right way, you actually can recover the original matrix. You can approximate the original matrix. And this matrix, as it turns out, these two matrices are much smaller because each one is just 8,000 times 2, 16,000.

And so this thing has just 16,192 parameters, which is 0.02% of the original 64 million. So this thing is called low-rank adaptation, or LORA. And it's incredibly widely used in the industry. And so what we do is we freeze all the parameters, we initialize all these change matrices to 0. And then we update just those two skinny matrices right here, here, we update only those matrices using gradient descent.

And when you do that, everything will fit into memory, which means that the whole thing will fit in. And you can just use two GPUs and get the job done. And if you actually use Llama's, the smaller models, like 7 billion, 13 billion, it can be fine-tuned comfortably on a single GPU, on a single Colab GPU.

All right, 9:54, time does not permit. So I have a Colab on how to do the fine-tuning using this technique. I will do a video walkthrough tomorrow or day after. And I'm done. Thanks, folks. Have a good rest of your week.

[APPLAUSE]

Thank you.