[SQUEAKING]

[RUSTLING]

[CLICKING]

PHILLIP ISOLA: Today, we're going to have a guest lecture from Jacob Andreas, and it's going to be on large language models. So we debated in this course whether to have a lot of material on large language models or only a little, and we decided to have only a little because LLMs have actually become such a big and important subject that they deserve their own dedicated courses. So there are additional courses taught by Jacob and Yoon Kim and others which you can take just to learn all about language models. Today, we'll just see some of the basics.

But why is this related to deep learning? There's two ways of thinking about it. One is that, well, large language models are all built on top of deep nets. So LLMs are made out of deep nets. That's the way that it works. But a lot of people have also started to say, well, what's the next big thing after deep learning? And one possibility is LLMs really deserve to be considered the next big thing. It's the next paradigm post deep learning.

So you can look at it either way. It's a paradigm that's built on top of deep learning, but it has new kind of fundamental math and characteristics that go beyond continuous weight updates and MLPs and ConvNets and so forth. OK. So Jacob is a professor here in EECS, and he's one of the world experts on large language models, and especially on the scientific understanding of them, not just building them, really understanding what they do. So let's welcome Jacob.

JACOB ANDREAS: Thanks, Phillip. So yeah. Just to reiterate, first of all, a bunch of the slides that I'm going to be using today are-- this is a compressed version of a couple of lectures that we teach in the big NLP class. If you're interested in this stuff, I encourage you to take that, which is 6.8610, as well as Yoon's special topics on language models class, and probably more course development in that direction coming in the next couple of years.

OK. This is a lecture about large language models. We were starting to plan out a lecture on large language models, so we asked a large language model to plan a lecture on large language models, and it gave a reasonable looking outline. And so this is more or less the structure that we're going to follow today.

But really, what this is going to be about is, in fact, mostly-- especially after the first chunk of the lecture-- abstracting away from some of the details of the underlying architecture that we're using to predict language and just talking about what you can do if you have a really good system for doing language generation, doing next token prediction, and now, you can imagine how to build such a system using some of the stuff that you've been talking about in this class.

But I want to start, actually, by talking not about language models based on neural networks, but much, much earlier generation of tools like this in NLP and machine learning, which were count-based language models. Have you talked, I guess, even most basically, about what a language model is or what I mean by that?

PHILLIP ISOLA: You should define it.

JACOB ANDREAS: OK, good. So in general, the only thing that we are going to be concerned with, really, for this entire class today is a model that predicts some next word in a distribution over-- or rather that takes in, I guess, a sequence of words or tokens or however you want to think of them as input and places a distribution over possible next words as output.

And if you have the ability to do this right, if you have the ability to see some sequence of input words and get a distribution over words that come next, via the chain rule, you can turn this into a distribution over strings, you can sample from it to generate language, and you can do all kinds of other things. And this basic question of, how do we predict the next word, given a bunch of words that came before? is something that people have been thinking about in language processing for a long, long time.

Really, dating all the way back to the end of the Second World War, when people were trying to decode the outputs of the German Enigma machines, and they had these decipherment procedures that would give them a bunch of candidate translations, and you needed to figure out which one was most plausible. And the best way to guess at that was basically to say, well, how plausible is this particular decoding that I've proposed to produce as a sequence of letters in German or as a sequence of words in German or something like that?

And for a long time, the ways in which people estimated these kinds of models was just by getting big text corpora and counting, right? So either just saying, well, let's ignore the context and figure out what the probability-- if I give you a random chunk of English, that some word is going to be "the" totally without any context. And if you sample from this, you get a naturalistic distribution over individual words that doesn't cohere into anything meaningful.

If you look at longer and longer contexts, you say, suppose the last two words that I ever saw were "once upon," and I asked you to guess what word is going to come next. What do you think is likely to happen after "once upon?" OK. So some people are saying "time." Some people are saying "a." Probably both of these things occur with nontrivial probability in a big corpus.

If I just go out onto the internet, and I count all the number of times that once upon occurs, a lot of those are stories where people are saying, once upon a. Some of those are typos where people have forgotten the a and are saying time. There's other stuff that can come next that's licensed by English grammar.

And if you just count these things up, you say, every sequence of four words, what's the distribution over things that could come next? You start to do a reasonably good job of generating text that looks sort of like natural language. And this is basically all that the field did from, say, the '60s to the early 2000s.

And around the early 2000s, people started to say, well, what happens if instead of just estimating this distribution by counting words in a corpus, I actually approximated it with a neural network model of the kind that you've seen throughout this semester? And it turns out then if you did it, you did only a little bit better than just these big count-based models at this next word prediction task.

So what we're looking at in this table is a measure called perplexity, which you can think of as just a measure of the quality of a language model's predictions. And you get a modest improvement in perplexity by moving from classical count-based models to neural network models.

And as our deep learning tool kit got better and better, we were able to drive this number down a little bit more, down to, I guess, the typical perplexity on a corpus of 107 in a corpus. And basically as-- and hopefully, these are models whose names are all familiar to you as a result of being in this course.

As our basic machine learning technology got better, our ability to do next word prediction on a fixed corpus also got better. And finally, right now, we live in the age of transformers, and almost all models that we train to do this next word prediction task in the modern era look like this.

Now, one thing that I'm playing a little fast and loose with here and that we'll come back to in a minute is that I'm just showing you a bunch of numbers or model architectures with numbers next to them. But in fact, all of these things-- and especially once we talk about the difference between GPT-2 and GPT-3 or some generic transformer language model in GPT-2, we're talking about changes in both the underlying neural network that we're using to do the prediction and also the scale of the data that we are training on.

And so obviously, the other big thing that has happened, and that you're seeing as these numbers are going down on this chart, is that between 2000 and 2020, we've developed technologies that allow us to train just much, much larger versions of all of these models on larger and larger data sets with more and more parameters. And these two things together-- better prediction machinery and larger data sets-- are really what are contributing to this improvement in prediction.

That being said, it is possible, even in this paradigm that we were talking about before, to have a large language model. So this, I think, is actually one of probably one of the first references to large language models in the machine translation literature. It refers to training a model on two trillion tokens with 300 billion parameters. And you can see that these parameters are actually n-gram counts. What year do we think this paper is from? Yeah.

AUDIENCE: 1995.

JACOB ANDREAS: 1995. So not quite that bad, but early 2000s. So really, long before these things were in widespread use for-- or in the context of deep learning models, people were training very, very, very large language models on large data sets with lots of parameters. And I think the main point that I'm trying to make here is that those models were typically not nearly as good as the neural models that we see today.

So I think there's a habit of talking that people have, especially now, suggesting that, really, more data and the ability to train larger and larger models is the only thing that matters and is the only thing that's kind of driving success in AI today. And in fact, I think you should take this as some evidence to the contrary, that we have been training models with as many parameters, as many data points, as many of the largest language models that we had today, and they weren't as good as the ones that we have today.

And there really is something special about the inductive bias that you get from neural sequence models that we didn't get in previous generations of language models. And we'll see some concrete examples of that momentarily. So just to say this very explicitly, even though we talk about parameter counts as being fungible today, in fact, there is a lot of historical evidence that how you use those parameters, specific architectural choices, actually are quite consequential.

So to go, again, very briefly through the history of the field, between, say, 1970 and 2014, a long stretch of time, people were not typically interested in these kinds of models as tools in their own right. Really, it was typical to have some larger pipeline system, an automated speech recognition system, or an automatic machine translation system, that was generating a bunch of candidate outputs.

And what you wanted to do with a language model was just rerank those outputs, to say, of all of the possible transcriptions of this sentence that I just heard, is it more plausible that a person said they wanted to recognize speech or wreck a nice beach?

Or in translation, do all the genders and numbers and things line up in a way that our machine translations systems, which are not that great at the time, weren't all that good at handling? And so for a long time, people were interested in language models, really, just as measures of linguistic well-formedness that you could tack onto the end of some other NLP model that was being custom trained to perform some specific task.

And what really shifted around 2014 was that people started to realize that you could just take these language models and get them to perform the task of interest on their own. So rather than saying, I'm going to train a machine translation system, I'm going to train something that's good at recognizing English sentences, and then I'm just going to combine them together, people said, well, what if we just pose machine translation the problem of taking a French sentence as input, say, and translating it into an English sentence as output?

What would happen if we just posed that as a next word prediction problem of the same kind where maybe these two words are two words in French, this is the first word of the English sentence that you've generated, and you're trying to guess the next word of the English sentence as well? And so starting around 2014, people started to take this basic language model machinery that we'd built up and use it to do sequence generation instead. And in parallel with that, our sequence models themselves were getting a lot better. We invented things like the transformers.

And then what's really changed since 2017 has less to do with the actual models that we're training, and more to do with the procedures by which we are actually training these models. So over here, it was pretty typical to say, I have my machine translation data set or my speech recognition data set or whatever. I'm going to randomly initialize my model. I'm going to train it just on this data set of interest. And what I'll have out at the end is a really good machine translation system and maybe nothing else.

And what people realize starting around 2017-- although, again, this was an idea that had been floating around in the machine learning literature for a long time-- was that you could get nontrivial transfer effects between different tasks. So you could, for example, start by training a model just to do next word prediction on a big data set, and then fine-tune that model to do some specific task of interest, like machine translation or speech recognition or whatever, and that this was a way to get away with much, much less data for any specific downstream task of interest.

And finally, the era that we live in now, which we might call the era of pretraining and prompting, doesn't even go through any of downstream task data at all, or very little, and basically just says, well, if I have a really, really good, really general purpose next word prediction system, what are all the things that I can do with it without having to ever even touch the parameters again or make only very small changes? And so this is the high level overview of the field, or maybe the history of the field for the last 10 years or so.

And we're going to talk now in a little bit more detail about each of the pieces of this, really focusing on the second two stages here. OK. So what does it look like, especially to operate in this kind of pretrain and transfer mode? And the thing that people typically did for a long time was actually not quite this language modeling task that we've been looking at here, but something that was called masked language modeling.

And I imagine you've seen something like this in different flavors for vision tasks, if not language tasks. But the basic task is we're going to take some text as input, we're going to hide some words from this text, and we're going to ask a model to predict what the missing words were in that text, given all of the context that showed up around it. And what's nice about this objective is that you can do it without needing any labeled task-specific data.

You just get all the texts that you can get your hands on. You train a model to perform this task, and you have a model now that's-- maybe you can set some of specific classification problems or whatever up as these mass prediction tasks. But really, for the most part, this model is not useful for anything in particular.

What it does know-- and for reasons we're going to look at in a minute-- is a lot about the structure of language, a lot about the meaning of language in a way that makes its representations very useful for downstream tasks. And so typically, what we would do after pretraining on as much text data as we could get our hands on with this masked language modeling objective, we would then -tune this model just using, say, one hidden representation from one last layer to do whatever downstream thing it was that we were trying to do, whether that's sentiment classification or whatever else.

And this was an incredibly effective paradigm when it first came out. And I think I really want to-- it's easy with the benefit of hindsight to forget about what a big deal this basic paradigm was when it happened. And I think even if you had gone to, say, NLP conferences for a couple of years before this BERT paper came out, which was one of the first papers to really show large scale transfer effects from pretraining on language modeling, people would have said, oh, yeah.

This next word prediction task, it seems in principle like it should be the right pretraining task. There was a lot of interest at the time in just learning word embeddings at the very inputs to these systems, and evidence that you could get good transfer effects from those, but nobody had really managed to crack the recipe for getting pretraining to work.

And around the same time, this BERT paper-- and another paper called ELMo, which was actually a little bit earlier than it and used RNN language models-- showed that you could train at a very large scale on this masked language modeling objective and then fine-tune at a very small scale on various other kinds of tasks. These are mostly sentence classification tasks, lexical entailment, or, I guess, logical entailment, sentiment, various things like that, and do very well.

So why does it work? Why is it that we can train a model to do this weird nuisance masked word prediction task and wind up with the ability to do all of these more complicated NLP things? And I think the main intuition that now, maybe, is pretty commonplace, but again, was, I think, surprising to a lot of people at the time, is that doing a really, really good job of even this simple task of predicting words in context requires both a huge amount of linguistic knowledge-- right?

It requires knowing that what's going to come after was probably needs to be a past tense verb. It probably needs to agree with the word MIT and so on and so forth. And you need a lot of knowledge about the world. If you really want to do a good job of filling in holes in the sentence, you actually need to memorize contingent facts about history, like the fact that MIT was founded in 1861.

And by training on, again, all of the text with all of the facts expressed in it, we would wind up with models that actually embodied a lot of that knowledge in their internal parameters and that could be accessed very easily via fine-tuning.

OK. One thing that you do not get out of this masked language modeling paradigm is a model that looks like this, a model that can actually generate the next word at the end of a sentence given everything that came before. And the reason for that is that the way we set up the inputs to these masked language modeling problems, the model already knows exactly how many words are going to be in a sentence.

It's been trained in such a way that, often, the missing words come in the middle of the sentence rather than at the end of the sentence. And so while you get good representations, what you didn't get out of any of these bidirectional transformer models was something that you could actually use to do text generation.

And so if we wanted to use these representations for doing things like machine translation or summarization or any other task where there was going to be textual output, typically, that second stage, the thing that was doing the text generation itself, had then to be trained from scratch. So what would it take to go from this kind of pretraining paradigm-- or, I guess, this kind of pretraining paradigm, this masked language modeling paradigm, to something that we could actually use to do text generation as well?

And I think the main thing that changed originally was just that people said, OK. This pretraining thing, this masked language-- this pretraining thing seems very effective. We already have models that can do conditional text generation. We already have models that look like this and that have not been pretrained in a very large way and that predict the next word in a sequence. Let's just figure out which one comes next.

Why not just do both of these things at the same time? Rather than training conditional language models to output translations or summaries or whatever, go back, really, to the paradigm that people were using all the way back in the pipeline days and just say, give me a really, really good model of all English sentences or all sentences in all natural languages that's not specialized to any particular kind of task.

And originally, people did this in basically the same-- or used these models in basically the same way as they use these BERT models. You could get representations out of these things that you could plug into a downstream classifier. You would, of course, also be able to fine-tune these models to then perform targeted generation tasks, again, using translation and summarization as the canonical examples. And this was really the paradigm that we operated in for a long time, and people just kept doing this with larger and larger and larger models, more data, more parameters.

Yeah. I guess these dots are all labeled with the logos of large industry companies because this was the last year that academic labs were really playing this game on their own. And thereafter, most of what we've seen has been various large scale models trained to do just this next word prediction task. Questions up to this point? I should say, also, feel free to interrupt or raise your hands or whatever. We can keep this interactive. OK. Cool.

And so again, for, I would say, most of the models that we're showing on this slide right now, the main things that people were doing with them were using them as starting points for downstream fine-tuning or for pulling representations out of or whatever.

But then something changed. And what changed is people started to notice that you could actually get these models to do some fairly surprising things without doing any additional fine-tuning, without really touching their parameters ever again, once these models got to a certain scale of parameters and data. So the examples that we're looking at on the slide-- and this is from a relatively recent paper looking back at this-- but was that large enough models could start to do arithmetic without having been trained in a targeted way to perform arithmetic tasks.

They could start to do things requiring highly specialized linguistic domain knowledge, like producing phonetic transcriptions of words, giving only their natural language input. They could start doing things multilingual without necessarily having to go out and collect a targeted data set for some task in some language of interest.

So the example we're looking at here is doing a question answering task where all the questions and all the answers are in Farsi. And various other kinds of complicated things. Answering questions about common misconceptions in the world, answering questions about properties of physical objects and physical situations, and so on and so forth.

And this is the GPT-3 paper, which at this point is pretty famous, but was one of the first ones to make this observation that you could really do quite well at a certain scale without needing to do anything after you had pretrained your model.

And I think this is kind of the defining feature of the era that we live in-- although there are going to be a bunch of qualifications that we come to at the end-- that models trained to generate language start to acquire all kinds of surprising capabilities at a certain scale without us necessarily needing to do any targeted fine-tuning for specific tasks.

That being said, part of what happened here is that people figured out that these capabilities were present in models, and another part of it is that people figured out, actually, more sophisticated ways of interacting with these models in order to get these capabilities out.

And so the next thing that I really want to talk about here is exactly that. Once you have a really good model of the entire internet, once you have a thing that's really good at doing this next word prediction task or doing natural string generation in a really general way, how do you then take that model, that distribution over language, and get it to do the specific task that you want it to do.

And once we start to move away from the fine-tuning way of doing things, really, the only mechanism by which we can exert control over a language model, over a text generation system, is by changing what we put into it, by changing what words we ask the model to look at, or the context in which it's making its next word prediction.

And one of the most powerful tools that we have for changing the context in order to get models to exhibit desired target behaviors nowadays is what's called in-context learning. And whether we want to actually-- why we call it this and whether this is the right way to think of it or not is something we'll come back to in a minute.

But the basic idea here is, rather than going to a model and saying, like we said before, what's the probability of the word "a" given once "upon"? If we're trying to do something like a sentiment classification task, we're going to say, ask the model, what's the probability of positive given-- well, so let's naively say, suppose the first thing that we wanted to do is just like-- we're assigning star ratings to movie reviews or something.

So if I said, this movie was terrible, one thing that we could imagine doing is just asking a language model, suppose you saw the words "this movie was terrible" as input. What's the probability that the next word that comes out as output is positive?

But if you think about this for a minute, this is actually a very surprising next word to come after an input like this. If you're randomly browsing the internet, and you come across a web page, and the last words that you see on the web page were, "this movie was terrible," you would assign a pretty low probability to any assessment of the movie coming next. This is just not something that-- people don't say, "This movie was terrible positive." People don't say, "This movie was terrible negative."

And normally, when you're in the middle of a review, you're not going to get labels for it after the fact. And so if you just ask your model, again, as a kind of distribution over natural looking text, what's going to come next after, "This movie was terrible"? It's probably going to be putting most of its probability mass on the word and or the start of another movie review or something like that.

And so if we want to model, again, our language model to be a model for this specific labeling task that we're trying to perform, we need some way of specifying what kinds of outputs we're expecting and what kind of tasks we're asking the model to perform. And so the trick that people have settled on, which is called in-context learning, or sometimes, few-shot prompting, is just to create a fictitious document in which the word positive is actually likely to come next.

And so we'll do that as follows. We'll say something like, I loved it, positive. Whatever. Worst film ever, negative. And now, if you imagine coming across this page on the internet somewhere that said, "I loved it, positive. Worst movie ever, negative. This movie was terrible." And I asked you to guess what word was going to come next.

Now, with reasonably high probability, you should assume that the next word is either going to be positive or negative. And given that these labels seem to actually match the sentiment of the text that comes before them, you might even guess that positive is going to be low probability here, and negative is going to be high probability here. And so the important thing is that we've taken the ability to perform generic next token prediction.

I can show you a weird looking document. You can figure out what the structure of the document is-- you, as clever human beings in this room-- and guess what word is going to slot into that structure. And the empirical discovery here is that once you've trained a large enough language model on a large enough data set, this is also something that just generic next token predictors acquire the ability to do as well. So in-context learning.

Another thing that you can do here-- and something that we'll talk about a little bit more at the end-- is what's called not few-shot prompting but zero-shot prompting, where instead of giving a model explicit demonstrations for the task that we're trying to get it to perform, we just give it a natural language description of the instructions for that task. So I guess we have this example on the slide.

Instead of an example input and an example output and another example input, another example output, we just give a natural language description of the input-- or natural language description of the task one input and see if an output comes out on its own. And this, again, was a capability that was observed to emerge in models once they got to a large enough scale.

And you can imagine doing all kinds of combinations of these things. You can give both an instruction and a bunch of few-shot examples. You can give the examples and then the instruction. You can give a much more detailed description of the task that talks about all the edge cases and the format in which the output should be returned and whether you want it capitalized or not. And for the most part, models are quite robust to all of these things. We've been through all of this before.

And this works remarkably well. But as soon as this behavior was observed, it sparked a lot of debate in the community about what was actually going on under the hood in these models and whether we actually-- obviously, we can draw an analogy between what we're doing here and training unlabeled data, where I've given you an x and a y and an x and a y and an x, and I'm asking you to predict a y.

Should we think of what's actually going on there as learning the way we think of gradient descent is learning, or is this something more complicated? And to piece this apart, there are a bunch of different things that you can imagine a model learning from an input that's being presented like this. One thing that you can imagine it learning or inferring from this input is just the format of the task that it's supposed to be performing.

I know from giving you this information that we're supposed to output labels that are either the word positive or negative, and we haven't seen any other labels. You're supposed to output them immediately after one sentence. And that you should expect in general, that the sentences that you're getting as input are going to be shorter sentences and maybe seeming to talk about movies.

And so one way to think about what might be going on in in-context learning is that it's just about learning the format of the output that the model is supposed to generate, and that all the other behavior that you get comes from pretraining and not from this context at all.

Oh, sorry. This is just a technical thing that we're going to skip through. Sorry. I'm a little out of order here. OK. I guess before-- well, let's actually finish this thought and then come back to it later. Sorry. My slides are a little out of order here.

Yeah. So this is maybe a point. So one hypothesis that you might have about what's going on with this in-context learning task is just that you're learning these formatting questions and nothing else. And if you think about machine translation as a prototypical example of this, obviously, you're not going to learn a new language entirely from scratch from three examples being provided in the input.

You're certainly not going to learn how to translate from English to French sentences containing French words that you've never seen before, English words that you've never seen before. And so to the extent that we observe these kinds of prompting behaviors actually making models better at their downstream tasks, we probably want to attribute that success mostly to providing information about the format or inducing the model to better perform this translation task, rather than all the other things that we wanted it to do.

And in fact, something that was observed for a little while was that models didn't really seem to be paying attention to the associations between labels and input text at all. What we're looking at on the slide right now is an example of a paper from Sewon Min where you actually took your in-context examples, you totally shuffled the labels-- so maybe I replace this with positive. That's not a working piece of chalk. Where did the chalk go? I guess it's this side. Change the label there. Change the label there. Shuffle everything around. These are these red bars.

And for the most part, they didn't actually hurt performance much at all relative to giving models actual labeled data for the task. And so for a while, the conventional wisdom was that you don't really want to think of this in-context learning phenomenon as learning at all. It's really just helping models retrieve tasks that they had previously learned how to do.

There were also all kinds of other weird pathologies that were observed in these models where the particular choice of words that you chose as your labels, if you said positive rather than good, or the thumbs up emoji or something like that, could actually have pretty substantial impacts on the quality of the predictions that you got from the model. And also, the models were extremely sensitive to the order in which these individual examples were presented, and you could shuffle them around and, again, get big increases or big drops in performance out of these models.

However, a lot of these issues seem to be going away with scale. And I'm not sure I have slides for this-- no-- in this particular talk. But even this phenomenon that we were looking at here, where models seem to be pretty insensitive to random labels or flipped labels or whatever, in even larger models than the ones that we were looking at here in GPT-4 and some of the other things that have come out since then, models do actually start to pay attention to the association between labels and input examples.

And nowadays, for example, if you take a state of the art language model, and you give it the backwards sentiment task where this is negative and this is positive, you can, for most tasks, reasonably expect that models will be smart enough to figure out not only that this is actually a sentiment classification task, but that it's supposed to say negative for positive.

This is also something that our theoretical understanding of has improved a lot in the last couple of years. And we now actually know, in particular cases, that models are capable of implementing real learning algorithms internal to their hidden representations.

So if I train a model to perform something that looks like a classification task or a regression task or whatever, at least on synthetic data in a clean room setting, there's lots of both empirical and theoretical evidence now that these models are actually running something that looks like gradient descent or some other learning algorithm inside their weights where the parameter vector of a smaller model being stored inside the hidden representations, and that thing is getting updated across layers as more examples get ingested.

Now, whether this is what's going on inside big models, nobody knows. Almost certainly, the answer is, it's complicated, and that there's both something that looks like real learning and something that looks like task retrieval and all kinds of other weird, complicated heuristics that we don't know about.

The main thing that I want to make is just that this is an area that's moving very quickly. These capabilities are changing very quickly. As recently as 2022, I think most people thought in-context learning was not actually learning. And now, we think models are actually capable of doing, at least in some limited way, acquisition of new skills in context.

OK. Taking a step back, what does this actually mean for the ways in which we use models? And if we remember back to the high level picture of the history of the field that we were giving at the beginning, we've gone from a world where we would go out, and we would collect targeted data sets for all of the different tasks that we care about, and train some task-specific model, to really just having one model for all of the different tasks that we care about, and only enough training data to actually specify to this big pretrained model what it is that we're trying to do.

And obviously, the only way in which we can do this, or the main mechanism, at least, by which we're going to be able to do this, is if the data set that we use to train that initial big model already contains evidence and information about how to perform most of these specific downstream things that we want to do.

And so, in some ways, maybe the most surprising empirical discovery of the last few years has just been that if you blindly-- or maybe not totally blindly, but with a limited degree of sophistication, scrape the internet, you find examples for translation tasks and summarization tasks and entity recognition tasks and sentiment analysis tasks, or at least tasks that are close enough to them that you learn to acquire all of these capabilities.

And more than that, models get the ability to not, I should say, just do standard tasks that have names for them and standard labeling things that have names for them, but to do more complicated things that actually require novel step by step reasoning for individual problem instances. And this is, again, something that both took a certain amount of skill to discover and a certain amount of actual work by researchers to figure out how to get models to solve some of these more complicated reasoning tasks.

So we're shifting gears a little bit right now. Up until this point, when we've been talking about a task, we've been talking about something like this, some standard thing with a name where you can just look at a sentence and basically spit the answer out without having to do too much thinking. But obviously, there are also lots of things that we want to do with models that don't look like this, that require more complicated reasoning, that take people a couple of seconds to solve, rather than a couple of milliseconds to solve, or even longer than that.

And so a natural question to ask, once you've seen these kinds of capabilities, is whether models also acquire the ability to solve some of these harder reasoning problems. And if you, again, around 2022, and in a totally naive way, just asked a state of the art language model a question like this, for the most part, the answer would be wrong.

Why is this the case? And I think one piece of intuition to have here is that no matter how deep you're building your transformer or whatever other architecture you're using for your language model, you only have a finite amount of computational resources available to you between the moment when you read the last word in the input and the moment at which you're expected to generate the first word in the output.

And so if you think about a task like this, really, the model has to see the question mark token or the word have or whatever at the end of this question and do all of the work that it needs to solve this hard arithmetic problem somewhere internal to its intermediate states. And not to anthropomorphize these things too much, but think about this as trying to solve a problem like this in your head with a very, very small amount of time. And basically, I think we should expect-- or at least not be surprised-- that models are not able to do this out of the box.

So what do people do when they solve problems like this if it's not just generating the answer right away? Well, in some amount of step by step thinking, right? Either writing out the intermediate steps of an algorithm on some of scratchpad, talking out loud through a solution, or so on and so forth. And maybe unsurprisingly, something that has been observed, again, once you get to language models at a sufficiently large scale, is that this kind of step by step reasoning works for them as well.

And so this has come to be called chain-of-thought prompting-- it's something that I think the LLM hobbyist community was aware of before it entered the academic discourse-- but where we now give models inputs that force them to reason more sophisticatedly, again, by just changing the inputs to them. So just like in the few-shot prompting case, we're interested in improving the capabilities of models, and the only mechanism by which we can control their behavior is to change the inputs that we give them.

And so now, when we're giving demonstrations of a task, rather than just saying, here's the question, and here's the answer, we're going to say, here's the question. Here's the bit of the answer that we want the language model to generate. And the first piece of that answer should include not the final output to this question, but this reasoning steps that will eventually lead us to the right answer.

And so going back to the example that we had on the previous slide, rather than just saying, here's the question, and 11 comes out, we say, here's the question. Roger starts with 5 balls. 2 cans of 3 tennis balls each is 6 balls. 5 plus 6 is 11. The kinds of things that you might imagine saying out loud to yourself or writing down in the course of doing a problem like this.

And when you do this, when you prompt models in this way, and then you give them new questions, they're pattern matching against the context. And just like we saw in these few-shot prompting examples, what you get is an answer that first includes a bunch of step by step reasoning, maybe following a different pattern from the one that we provided in the input, and then the answer at the end.

And obviously, this is something that you can combine with all of the other tricks that we were talking about in the context of few-shot prompting, meaning you can also provide high level description of the task. You can also provide a meta template, the kinds of things that make good step by step reasoning instructions, information about the granularity of the reasoning steps and so on, information about the format in which the final answer should be presented so that you can parse it out, and so on.

But really, the high level idea here is just rather-- between our x's and our y's, we're going to interpose some reasoning steps. And if you do this in the input, then you expect to get it in the output as well. And so this is just writing this more formally.

We're going to, between our x's and our y's, put some z's that show the model how to reason. And because everything's in the input because we're in this very few-shot paradigm that we were looking at over here, it's really not that hard for you as an experimenter or a model builder or whatever to write 5 or 10 or 100 examples of these things to provide as input to the model.

So this works in a bunch of different regimes. Again, I think something that maybe is worth emphasizing is that this is a capability that arose, just from a historical perspective, a little bit unpredictably and only at a certain scale. So I knew of at least three different groups at MIT around 2019 who were trying to do something like this, and it wasn't working, because the models that we had at the time weren't big enough.

And then GPT-3 came out, and people tried it again, and all these things work. You can do this for free response math questions. You can use it for multiple choice questions. You can do it for common sense reasoning. And really, this is a meta trick that you can apply to almost any domain of interest.

So here are some more examples of places you can do this. It's really quite general. And it's really, really, really quite effective, especially on these problems that require a lot of step by step reasoning. Questions? Yes.

AUDIENCE: Why do you think scale made it possible?

JACOB ANDREAS: So why does scale make it possible? I mean, again, I think even the-- well, so there's a couple of different things, right? There's recognizing that a given input is asking for a model to perform some of abstract reasoning capability and not just to pattern match against a template.

And so I think a meaningful difference between some of these examples and some of the things that we were looking at over here is that what you put into that z in principle looks very different from problem to problem. You're not just filling in holes in a template. And that's something that-- small models, you can give them this data. They'll do the copying bit, and they won't do the generalization bit. And it's like the ability to generalize to new reasoning chains, that's the thing that you get.

I think it is also the case-- and this is something we'll come back to at the end-- that in the very largest models where people apply these things now, there is some large amount of deliberate data collection where people are making sure in the training set there are enough examples of these kinds of step by step thinking out loud. Another question. Yeah.

AUDIENCE: It seems like you can enforce multi-step reasoning through chain-of-thought prompting. And I'm wondering, where does reinforcement learning fall into that? Is that just to make it marginally better?

JACOB ANDREAS: So there's a question about reinforcement learning. We'll talk about reinforcement learning at the end. Cool. Good. And I think maybe the main thing to take away from this is just that you can, really, for some of these hard reasoning problems, especially in larger models, dramatically improve the quality of the outputs that you get just by prompting models to think step by step in their outputs, by providing examples of the kinds of reasoning chains that you want them to go through.

Even more infuriatingly, as soon as the first round of -of-thought papers came out where people were manually authoring these z's, these reasoning steps, people started to ask, is there just some magic incantation that you can give models so that I don't have to demonstrate the reasoning steps anew for every new task that I'm trying to perform, but I can just prompt the model in a totally task generic way that will cause it to do this reasoning and then produce a final answer?

So as you might expect, somebody basically just did a brute force search in the space of magic incantations that you could append to the beginning of your answer in a language model. They found, "Let's think step by step," is the best string for, at least, the models that existed in 2022. And again, just maddeningly, this works super well.

The way to read this chart is you can take a model that gets 18% on an arithmetic task and bring it up to 80% on an arithmetic task just by typing, "Let's think step by step," before you ask the model to generate an answer.

Again, whether this really reflects some deep or unexpected property of the training distribution, or whether somewhere on the internet or somewhere deep inside OpenAI, some product manager said, hey, let's go out and generate a lot of documents that say, "Let's think step by step," and then have the reasoning steps, we don't know. We may never know. What we do know is that there are actually these just kind of magic phrases that you can give to models that cause them to become much better at some of these tasks.

Good. And obviously, this is-- even though I've been calling it magic, it is not magic. And in general, you are always going to do better by giving explicit examples of the kind of reasoning that you want to see, rather than asking models to come up with it on their own. But they are sometimes, to a nontrivial degree, able to come up with it on their own. Here are a bunch of different prompts that were tried and varying degrees of effectiveness.

Cool. Here is a meme that we found when we were putting this slide together that, back when I was starting grad school, all kinds of beautiful math and structured prediction things, and now, really, the most effective thing that you can do for a lot of tasks is to write the words step by step. And this is some amount of NLP research, but not all of it.

And I think there's a higher level thing here, which is that a weird feature of living in the modern era is that, actually, one of the greatest degrees of control we have over models, one of the best tools that we have for making them better, is just coming up with better examples for the input, coming up with better formats in which models can generate their output, better guidance for what kinds of outputs they should provide without doing any kind of fine-tuning.

And so whether or not you think this is interesting research at this point, certainly, there's an enormous amount of demand in industry for people who have just very good intuition for how to come up with strings that look like this and other ways of structuring models and getting their outputs to come. There are things to learn. My students are much better at this than I am. And it's a thing that people get better at over time.

But how do we push this beyond what you get from off the shelf models that have just been trained on the whole internet? And in particular, is there anything that we can do during the pretraining phase, knowing that the ways in which these models are going to get used now is not just to score the grammaticality of sentences in English, but actually that they're going to be given instructions, that they're going to be given in-context examples, that they're going to be asked to reason step by step when they're in use? Question first.

AUDIENCE: My question is, does it make sense to say that you don't need to fine-tune any more just because it's good at everything, or there's, like, better--

JACOB ANDREAS: So the question is, does it make sense to say you don't need to fine-tune any more because the model is good at everything? This is something where it depends a lot on the task that you're trying to perform. Certainly, if you have some totally novel thing or some weird looking input, you can get models into a better place by fine-tuning them than by providing examples in context if you have lots and lots and lots of examples.

I think right now, the balance of evidence is in favor of, you'd still rather do some kind of fine-tuning on them than just dump 100,000 examples in context. That being said, this is also an area that's moving really fast as the context over which these models can operate gets longer and longer and is something that could change. But most people don't have the ability to fine-tune a very large model in the first place or can only do that once and not a gajillion times.

And so just from a practical perspective, often, this is kind of the best tool that we have for steering these models. But then that feeds into the next question of, given that this is how people are mostly going to be interacting with them, what should we be doing differently when we train the models so that they will do better at in-context learning, so that they will do better at instruction following, so that they will do better at chain-of-thought reasoning?

Sorry. Skipping ahead here. So we set up before that one simple and effective trick that you can do, even before any of this chain-of-thought stuff, was just to give models these x, y examples in context. We saw also that you could give natural language descriptions of tasks and ask models to perform these tasks. But here, if you, for example-- sorry. To step through this piece by piece.

If you just give a model the string, "This quiet, introspective and entertaining independent is worth thinking," it's not even clear what the task is going to be. And so models, by default, when you give them an individual x, they're not going to give you a y, they're just going to give you some other text that looks of related. So here was some, I think, GPT-3 or 2, late 2 model given this sentence as input. And what you get out is not a sentiment label for this, but just more text kind of in the same vein.

If you just modify the format of this a little bit, and you prefix this thing with review, and then you put answer at the end, even here, the model doesn't know the answer to what question. You get, "This thought-provoking independent film is worth a watch." Maybe the model is inferred here that it's supposed to be doing grammar correction task rather than a sentiment analysis task.

If you give a detailed natural language instruction, now, there's, in principle, enough information in the input for the model to figure out what it's supposed to be doing and produce a positive word as output. But even this behavior is quite brittle in models that are off the shelf. And one reason for that is that there are lots of-- again, thinking of these as like models of random documents on the internet, there are lots of ways of starting a document with something that looks like an instruction and then an input and continuing that document without actually providing the text that follows that instruction.

So here, we've prompted the model with, "Answer the following question. Why isn't the next-word prediction objective enough to enable generic instruction-following?" The model, rather than producing an answer here, starts to generate more questions of the same kind, maybe assuming that, then, all of those questions will be handed off to a reader to answer or something.

And so one thing that has become part of the pretraining process after you've gone out and learned to do next token prediction on the entire input is to realize, OK. 99% of what people are going to be using these models for is instruction following tasks, so let's actually just give them targeted supervision on instruction following tasks. And so if you interact with any modern language model now, and you say, "Answer the following question. Why isn't the next-word prediction an objective enough?" It will answer the question.

And this is not a natural behavior that you get just from training on the internet. You actually do need to do some amount of focused fine-tuning to get this kind of zero-shot instruction following behavior. And in a previous generation of models, if you wanted to do this, you would have had to give lots and lots more examples of questions or really careful formatting of the output and so on.

And so the main technical thing here, which is not actually all that technically interesting at all, is just to go out and gather a bunch of targeted instruction following data in a way that will cause models to, by default, assume an instruction following interpretation of inputs rather than other kinds of interpretations.

And there are a couple of different schemes that people proposed for following this in the literature. The first thing which happened mostly in the academic community was just to say, OK. Let's take all of our existing standard NLP data sets for sentiment analysis, for machine translation, for text summarization, for sentence classification problems of other kinds, and just fine-tune language models on all of these things all at the same time with little templates that had natural language descriptions of each tasks and examples of inputs and outputs.

And this actually worked pretty well. And even on small scale academic-sized models, just essentially synthetically generating a bunch of instructions derived from standard data sets and standard tasks that people had put together got you a reasonably long way.

But it turns out, if you really want high quality, totally general instruction following behavior like this and question answering behavior like this, there's really no substitute for just going out and collecting a bunch of new data yourself for as many tasks as you can imagine, as many inputs as you can imagine.

And so a lot of what goes on or is believed to go on inside the big LLM company shops right now is exactly this, just various clever ways of collecting diverse, comprehensive kinds of instructions that users are eventually going to ask their models so that they can fine-tune on that data. Yeah.

AUDIENCE: Sorry. This was a couple topics back. But when you showed that just adding step by step increased the accuracy, is that just for that same word problem plus step by step as the entire input, but that accuracy doesn't transfer to if you just ask the same prompt, and what's the answer? It's able to answer that math question when you add that input prompt, but it still wasn't retrained or learned anything to answer the question correctly with the prompt that is not telling it to go [INAUDIBLE].

JACOB ANDREAS: Yeah, that's right. So for all-- let me see if I can find this specific table. But for all the examples that we were looking at here, in none of these cases are we-- I guess now, we're a little out of order, because these are models that have been instruction tuned following the procedure that I showed you later on. But it's the same base model in all of these cases. And all that's changed is basically the text that goes in front of the answer. Yeah.

AUDIENCE: Maybe jumping the gun here, but you said collect large data sets from users.

JACOB ANDREAS: Yeah.

AUDIENCE: Why not have a system synthetically created to augment data sets for instruction?

JACOB ANDREAS: So the question is, why not have the system synthetically create these things? Well, so there's two things that you could do synthetically. You could just ask models to generate, what are kinds of instructions that users might issue in the future? And maybe if you have a lot of instructions of the kind that you want to be able to use eventually, and you say, generate more stuff in this genre, you will get some interesting generalization there. And then you can take those things, and you can go out and ask people to label them. And that is certainly part of the process everywhere.

Another thing that you could imagine doing with synthetic data is asking the model both to generate a bunch of instructions and the text that follows those instructions and then fine-tune on that. And this is something that we can talk more about later. But if you just do that, if you just say, generate some data and then fine-tune on that same data, the model can't possibly have learned to do anything new.

Information, theoretically, can't get something from nothing. And so when you're in-- if you want to train a model on synthetic data, you either need some sort of outside source of information that's going to filter that data down to the good stuff or some difference between the process that's generating the synthetic data and the process that's training on the synthetic data.

And so one concrete example you can imagine doing for this instruction following thing, which I think people have tried, is to say, I'm going to write a really good prompt that really makes it very clear that what the model is supposed to be doing is instruction following and not any other thing. I'm going to have a ton of few-shot examples, and I'm going to have very explicit meta instructions saying, you're supposed to be an instruction following model.

I'm going to sample a bunch of data from that model, and then I'm going to fine-tune on that data. And what you get there is not a model that knows how to do anything new, but just a model that behaves like that really good prompt that you wrote is sitting there in the context all the time so that the user doesn't have to type it out themselves.

That trick is called context distillation, and it's like one of the standard things that you can do with synthetic data that works. But it really just moves mass around in the output space, and it doesn't tell the model how to do anything new. We can do more questions at the end. I do want to get ahead to the last bit of this.

And so coming back to things that change at scale or capabilities that emerge as models get bigger and bigger, instruction tuning of the kind that we were describing, like doing some focused fine-tuning on targeted instruction data, does actually seem to be quite important for getting at some of these capabilities.

And so looking especially at the second plot here, if you take a model, you just do larger and larger scale pretraining without this instruction tuning phase, it gets better at some standard collection of language understanding tasks, but not at a particularly impressive rate.

If you grow the model, and in parallel, you do this instruction tuning step at the end, then you start to see really much greater improvements with scale. So these things interact with each other in a positive way. You need targeted training data, and you need really large models in order to get good general purpose instruction following capabilities.

And is this all we need? Or is this the best thing that we can possibly do, knowing that we're going to live in a world where people are mostly interacting with these models by issuing instructions and asking them to do some of chain-of-thought reasoning?

Consider the following set of instructions that a user might issue to one of these models. And these are all-- sorry. Coming back to what we were saying before, the data collection process for instruction tuning looks something like this. We come up with a set of instruction prompts. We ask human annotators to label them, and then we just maximize the probability of these responses given the instructions that preceded them.

But you're going to run into some issues here. One of them, maybe, is that the data is going to change after you train the model, so maybe people weren't knowledgeable enough to say, by default, when somebody is asking a transformer, when they're using transformer with a capital T, they mean the machine learning model and not the electrical component. Maybe in 2017, the data that we were collecting, nobody was asking for explanations of transformers because nobody knew what a transformer was.

Similarly, if in this question asking data, if when you go out and you ask your users to issue-- or you collect a bunch of instructions and then you get people to label those, and you always label them with the right answer, then in the future, if somebody asks an impossible question to the language model, it has never anywhere in its training data, or its fine-tuning data, at least, seen somebody say, "I don't know."

And so of all of the possible responses you could give to who won the 2028 World Cup, maybe the model knows enough about soccer to know that it should be the name of a country. Maybe it knows enough about soccer to know what countries are soccer powerhouses and that Argentina is a plausible winner and maybe the US men's team is not a plausible winner of the 2028 World Cup.

It doesn't assign high probability to, this is an impossible question to answer, because it's never, in the training data, seen a question that was impossible to answer. And more generally, even if it saw a impossible questions to answer, the language model doesn't know what the year is, so it doesn't necessarily know-- by default, unless it's been fine-tuned to output 2024 when somebody asks what the year is. And so it doesn't know what things are in principle answerable and what things are not even in principle answerable.

And so what you'd really like in these cases is for the model to say something like, I don't know. This is not part of my training data. But you have the chicken and egg problem because your human annotators aren't going to know what the model doesn't know and therefore can't supervise it to say, "I don't know," in appropriate places.

Similarly, as a model developer, there might be certain kinds of requests that we don't want models to answer at all in the first place. It's a bad PR look for OpenAI or Google or Microsoft if you can ask, give me ideas on how to cheat on my midterm, and the model comes up with lots of ideas for how to cheat on the midterm.

Really, for many users, what they want, if they're putting a language model out-- or many, I guess, model builders, if they're putting a language model out into the world, what they want is a model that will say, I don't know-- or sorry, I can't help you in response to some category of requests.

In fact, if you go to ChatGPT, and you say, give me ideas for how to cheat on the midterm, it gives you a bunch of ideas that are not about how to cheat on the midterm, but instead a list of activities that you could use to get better. And it gives you a little preachy, this is bad for you.

Another funny thing that has been observed in-- if you just do instruction tuning and nothing else-- is that certain categories of questions, models actually get worse at answering, or at least become more predisposed to produce incorrect answers as they get larger and larger. So here's an example from this truthful QA data set where you're asking models of various sizes, what happens if you smash a mirror?

If you ask a 300 million parameter model, what happens if you smash a mirror? It says, you smash a mirror. Not wrong. Reasonably high probability. You ask a 13 billion instruction tuned model, what happens if you smash a mirror? It defines a mirror for you. You ask a 7 billion instruction tuned model. It says it'll shatter into a million pieces. And you ask a really big model, and it says you'll have seven years of bad luck.

And the reason or the way to think about what's happening here is that the model, as it gets bigger and bigger, is not just getting better at instruction following, but it's modeling, in a finer grained way, the training distribution. And I think this is probably still true. Certainly, it was true at some point. If you looked at what text was likely to co-occur with, what happens if you smash a mirror on the internet? It's mostly stuff talking about the superstition and not about the physics of mirror smashing.

And so as you get better and better at modeling that training distribution, if you don't have some way of explicitly signaling to the model that it's always supposed to answer truthfully, then it becomes able to distinguish between, this is the kind of question where, on the internet, people usually give a truthful answer, and this is the kind of question where, on the internet, people usually give a superstitious answer. And so this Lin et al. Paper documented a whole category of phenomena like this where models actually, by default, get worse as they get bigger.

OK. So just to say these two things again, one more time. We want models to say, I don't know. Sometimes, we want models to answer in accordance with what they actually know and what they think the user wants to hear. Sometimes, we often want them to not give answers, even when they know the answers, for user safety reasons or whatever.

And fundamentally, in all of these cases, it's hard to get models to output the right answers just by providing targeted supervision because we don't have access to their internal knowledge. We don't know what they and therefore can't surgically intervene on exactly the questions where we would like them to say, I don't know, or any of these other things.

And so a standard thing-- and we're really only going to talk about this for a little while in the interest of time-- but is that there's now a third step in this language model training pipeline, which is reinforcement learning from human feedback phase. Here's an example of ChatGPT talking like a pirate while describing reinforcement learning with feedback. I don't know why we included that in the talk, but it's there.

OK. So fundamentally, how are we going to get around all of the issues that we talked about before, this thing where we don't know what the model knows and want to control its output in cases where we don't actually know exactly what it is supposed to say? Consider the following simplified example.

Suppose now we're just trying to build a model to do some specific targeted task, say, text summarization, where there's many, many good summaries for a model or for any given article. Lots of different ways in which we could produce a reasonable looking summary of some textual article and where we want models to learn that they're allowed to output a diverse distribution over responses, that there's many acceptable things to say here.

One thing that we could imagine doing would be to ask our human annotators to generate all of the acceptable summaries and none of the unacceptable summaries for a given article, write them all down, and then train the language model to generate all of them at the same time. Obviously, this is not going to be practical because that's a huge amount of work for our human annotators, and we'd rather not ask them to spend their time that way.

Another thing-- coming back to the synthetic data question from before-- that we could do here is ask the model itself, after it's done a little bit of instruction tuning, to generate some summaries. And here, because it's just a language model, we don't have to pay it, or we don't have to pay for the electricity, we can generate a large, large number of summaries without really needing a human in the loop.

And so the question becomes, how do I get from this large collection of summaries that I sampled from my model to something that will make the model actually better at this summarization task? Well, if the model is itself the one generating the summaries, then in some sense, the only thing that we can do is pick out the good summaries or identify the bad summaries, and just say, of the summaries that the model itself generated, cause it to assign higher probability to the good ones and lower probability to the bad ones.

And so again, one thing that you could do is ask your human annotators to go through and not just write the summaries themselves, but to label the machine-generated summaries either by assigning them numerical scores, or just giving them thumbs up or thumbs down, or expressing pairwise preferences between them, or whatever else. And this is also a reasonable strategy, but again, something that, if we want to do it at a really large scale for a large number of summaries, is going to require a huge amount of human annotator effort.

And so what we actually do is something as follows. Again, we're working with a normal language model, which by analogy to reinforcement learning, we're going to briefly refer to as a policy. Don't worry about this. Again, it's just a thing that takes some text as input and generates some text as output.

Let us suppose for a moment that instead of our human annotators, we had some magical black box that could pick up any article summary pair and give us some number corresponding to how good that summary was. Then, all we would really need to do is find some way of optimizing this language model, choosing the parameters, P, for this language model so that the score assigned to the summaries that it generated became higher and maybe became as high as possible.

And given that we only have five minutes left, I'm really not going to go too much into the details of this, other than to say that in practice, what people do is we get our annotators to hand label a bunch of articles with thumbs up, thumbs downs, just like we were saying over here. But we use these not to train the model and instead to train a reward model.

So we train a model not that's generating text, but instead a model that's trying to reproduce the human annotators' judgments about which articles are good and which ones are bad. This is an easier task, and this is a really important point. This is an easier task than generating articles themselves. We've all taken our intro CS theory class. Recognition problems generally are easier than generation problems.

And so for summarization, for math, problem solving, whatever, it is generally, both computationally and from a statistical efficiency standpoint, going to be easier to train models to score summaries than to generate summaries. But as soon as we've done this, as soon as we have a good summary score, we're done. All we have to do is throw standard optimization techniques at this objective right here, and what we get out at the end is a model that gets better at producing summaries.

To go just one step further into detail here, the standard thing that people do now is an algorithm called policy gradient, which I think I do not even have written out here. But basically, you can turn this objective into something that looks like just doing normal maximum likelihood estimation where each sample is weighted by the score assigned to it by the reward model. And so if you've heard of reinforcement learning with human feedback, that is basically the entire thing. You go out, and you train this reward model, and then you do RL against the reward model that you collected from human judgments.

Now, going back a couple of slides to here, one thing that we said is that-- or I guess to here. One thing that we said is that we would conceptually just like it to be the case that the model assigns high probability to the things that the people thought were good and low probability to the things that people thought were bad.

And there were a bunch of other learning algorithms that you can use to do this that don't look like reinforcement learning. So if you've heard of SLiCK or DPO or any of these other things, basically, there are other ways of taking these human labels, not training a reward model, and just directly optimizing your language model to make the good things high probability-- or I guess the bad things, low probability, and the good things, high probability.

Yeah. And that's basically what I wanted to go through today. Happy to do some questions at the end. But just to pull up one summary slide back from the beginning. It's actually this one. Really coming back to what Phillip was saying at the very beginning of this lecture, I think the defining feature of the language model era, the defining feature of the world that we find ourselves in right now, is the stuff that's written in this rightmost column here.

I think, much to many people's surprise, if you do large enough scale training on just tons and tons and tons of text data, and then a little bit of refinement after the fact that doesn't actually change really much about what models and mostly just the format in which they produce their responses by default and the assumptions they make about the inputs that are being given to them, you wind up with really flexible, really general purpose predictors that can be given instructions, that can be given small numbers of examples as input and that can then do a relatively broad range of tasks without any kind of additional fine-tuning or any additional modification of their internal parameters.

And I think figuring out how far to push this, figuring out what comes after this chain-of-thought thing in terms of making models better at solving even more computationally complex tasks, I think are then the research challenges for the post-LLM era. And I will wrap up there. If folks have questions, I can stick around and answer them. But thank you.

[APPLAUSE]