Introduction to Large Language Models (LLMs)

Prof. Tanmoy Chakraborty, Prof. Soumen Chakraborti

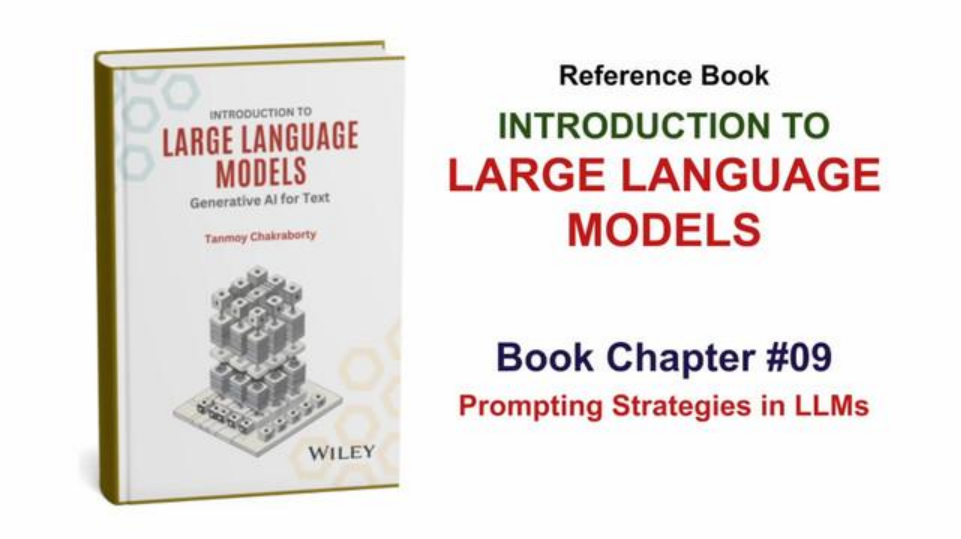
Department of Computer Science & Engineering

Indian Institute of Technology, Delhi

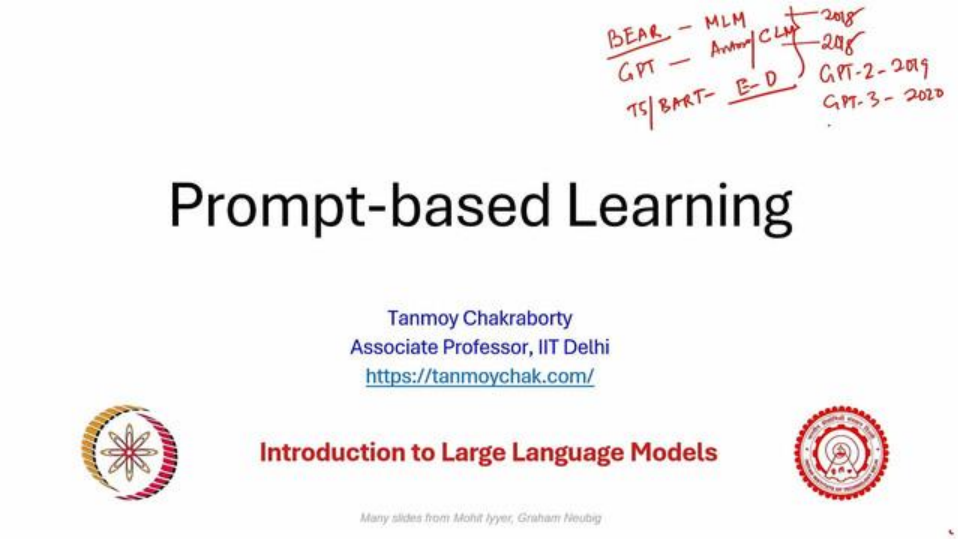
Lecture 22

Prompt-based Learning

Welcome back. So in today's lecture we are going to talk about prompts, right and you know when it comes to you know chat GPT kind of models, large language models, the first question which comes to our mind is how to write a prompt, right. What is going to be the optimal prompt for a specific model, right. So in this lecture we will discuss different types of prompting techniques, right, how prompts affect the accuracy of the models, right? And how with the scaling, with the increasing size of the models, how it basically affects the accuracy and how a specific prompt will be responsible for producing accuracy across different types of models. We'll also discuss prompt sensitivity, right? We'll see that most of these models are highly sensible to simple perturbations of prompts, right? And that would affect the accuracy, that would affect other aspects of the model and how we can quantify them, right? So far, we have discussed different pre-training strategies.

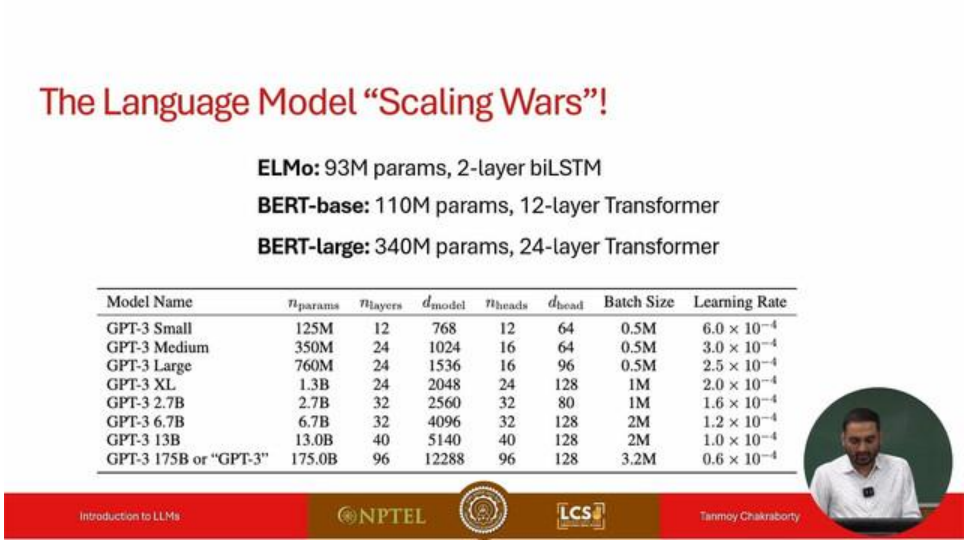


We have seen encoder-only models, right? We have seen models like BERT, which is a pure encoder only model, right, and these models are pre-trained with an objective called mask language modeling, right, MLM. We have seen models like GPT, right, which is a purely decoder only model, right, and these models are trained using autoregressive setup, right, or causal language modeling. This is called causal language modeling or autoregressive setup. There are other models like T5, BART and they are encoder decoder models. Okay, which are also trained with kind of autoregressive setup where your input will be fed to the encoder and decoder will predict the next word or decoder will predict if your input is part of, if your input has noise, the decoder model will be able to essentially denoise your input, right? And you know the board model came around 2018, first GPT paper was written in 2018, GPT-1 and then GPT-2 came in 2019, GPT-2 in 2019 and then GPT-3 in 2020.



Right and the GPT-3 paper showed that the model doesn't need any fine tuning, we just need to write prompts and the model will be able to understand your prompt, okay. We will discuss all these things in this lecture. Okay, so I strongly suggest you guys to read this wonderful survey paper. This is a survey paper written by Graham Neubig and his team from CMU and this is, you know, this rightly, nicely reflects the, you know, different prompting strategies and the kind of evolution of the overall, you know, prompt, you know, prompt as kind of aspect overall, how it evolves, right? How do we quantify different components of a prompt and so on and so forth. It's a very nice survey paper that I strongly recommend you to read.

Okay, so you know there have been kind of, we have seen, we have witnessed that there has been a kind of war, right, among all these giant industries, right, giant companies, Meta, OpenAI and Google, they have been building larger and larger models with more and more parameters and so on and so forth. So this is essentially a scaling war. You just increase the size of the models without changing the framework much, And you start seeing different emerging properties. So one such emerging property is this prompt. Now, this particular slide gives you an overview of how this parameter size increases over time.



Model Name	n_{params}	n_{layers}	d_{model}	n_{heads}	d_{head}	Batch Size	Learning Rate
GPT-3 Small	125M	12	768	12	64	0.5M	6.0×10^{-4}
GPT-3 Medium	350M	24	1024	16	64	0.5M	3.0×10^{-4}
GPT-3 Large	760M	24	1536	16	96	0.5M	2.5×10^{-4}
GPT-3 XL	1.3B	24	2048	24	128	1M	2.0×10^{-4}
GPT-3 2.7B	2.7B	32	2560	32	80	1M	1.6×10^{-4}
GPT-3 6.7B	6.7B	32	4096	32	128	2M	1.2×10^{-4}
GPT-3 13B	13.0B	40	5140	40	128	2M	1.0×10^{-4}
GPT-3 175B or "GPT-3"	175.0B	96	12288	96	128	3.2M	0.6×10^{-4}

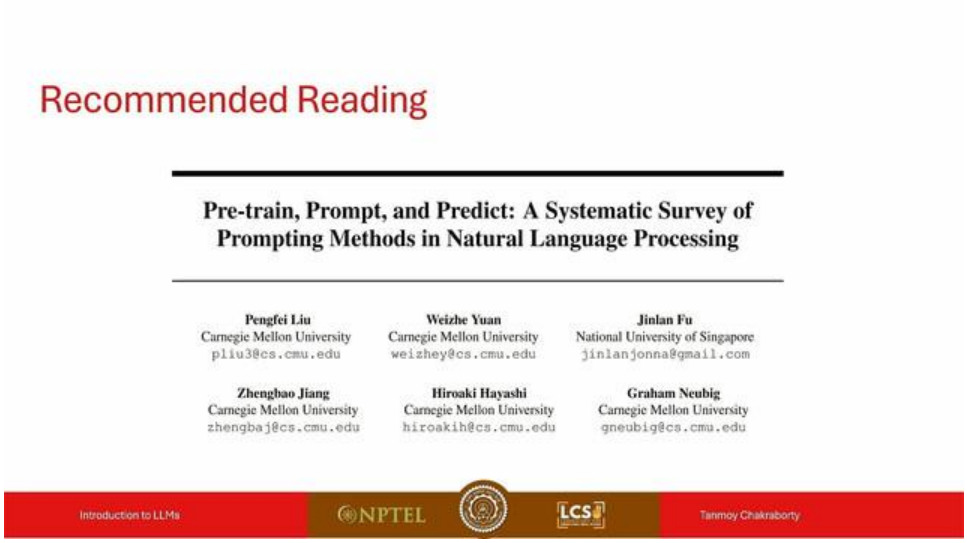
So earlier, we had the ELMO model, which had roughly 93 million parameters. Then BERT came with 110 million parameters and a 12-layer transformer. BERT base, BERT large, 24 layers, 340 million parameters, and so on. The GPT series started coming in; you know, if you look at GPT-3, Different variations of GPT-3: small, medium, large, XL, right? 125 million parameters, 12 layers, right? You have 12 attention heads, right? Dimensions of the attention head and the hidden state, right? And then bath sizes, different bath sizes. And as you see, as the model size increases, the batch size also increases, doesn't it? And then your learning rate decreases.

So essentially, you are training bigger and bigger models with more iterations. more times, right? Whereas, you know, we have been talking about models with millions of parameters, Then, you know, a couple of billion, 1.7 billion, 2.7 billion, and suddenly the GPT-3. which came with a gigantic size of 175 billion parameters, correct? This is kind of 100 times larger than GPT-3, right? And in terms of layers, now you see there are 96 layers, right? 96 decoder layers right and look at the you know dimension of your hidden state.

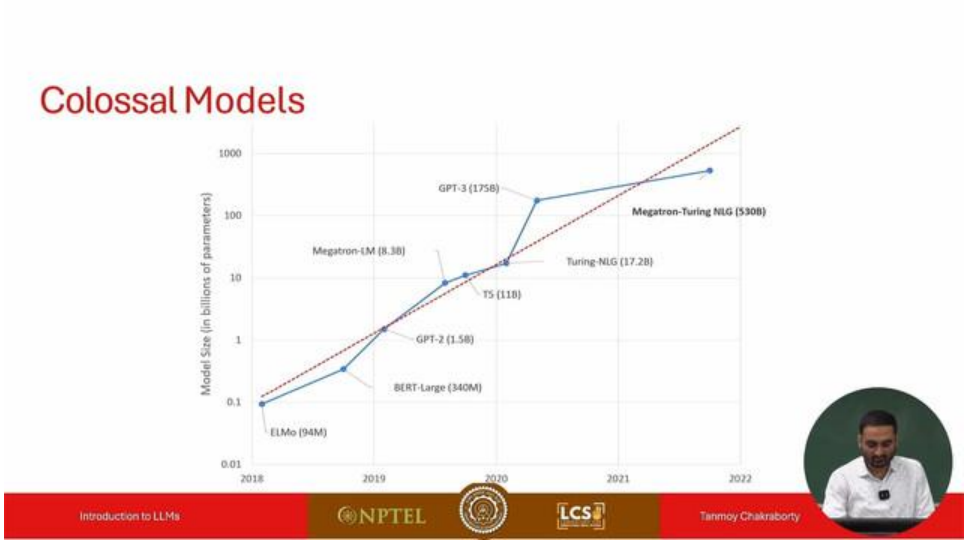
This is the number of heads: 96 heads. And right then, this is the dimension of every head. Bash size also increased extremely significantly. So earlier it was 1 m; now this is 3.2 m.

The learning rate also decreased significantly, right? So you are essentially training the model more and more times. For more and more iterations, the model will basically understand the data set. At the same time, if you look at the data. So, such a gigantic model automatically needs a large dataset for training. ELMO required approximately a 1 billion token size training set.

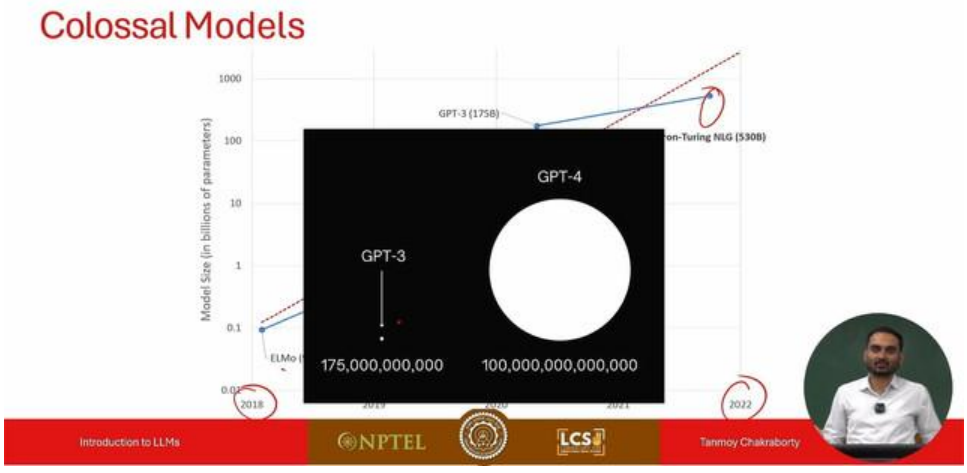
About 3.3 billion tokens are in the training data. Robata has approximately 30 billion tokens. And then the GPT model came with 300 billion tokens in the training set. And all these data cells contributed to these 300 billion tokens: Common Crawl.



WebTest, Book Corpus, and Wikipedia Corpus show the percentage of these data sets. So both in terms of the number of parameters and the number of tokens. Present in the training set to pre-train these gigantic models. Now this kind of gives you an idea of the model size, you know. across different years, starting from 2018 to 2022, You'll see models like ELMO, BERT, GPT, and the kind of linear scale.



Right? It increases quite linearly with time, right? So we have this Megatron Turing model and energy model. 530 billion parameter models, which are even, you know, even bigger than the GPT-3 model. Okay. So, and GPT-4, you know, God knows how many parameters there are, But, you know, it is huge, right? If GPT-3, then I got this figure from Twitter, right? If GPT-3 is this much, GPT-4 will be this much, right? So when we see a galaxy, right? We see our planet Earth, right? And then the sun, right? It looks like this. So if you look at, if you compare GPT-3 and GPT-4, You know, basically, you are comparing, you know, a planet.



Like arc right or whatever, and then you have the sun. Okay, but we don't know the exact number of parameters, by the way. This is, you know, a rough estimate somebody made on Twitter. I thought this figure was a nice depiction to show a comparison. between GPT-3 and GPT-4.

Okay, so what does the scaling law mean, right? What does the scaling mean? Okay, so in the GPT-3 paper, before GPT-3, If you look at the paradigm that we have discussed so far, we have pre-training. Right? So, pre-train, fine-tune, and predict, right? So you have a model that is pre-trained on raw documents and a raw corpus. This can be encoder-based pretraining or decoder-based pretraining. And then you fine-tune this model further for a task-specific dataset. So we discussed, for example, in the case of T5, you pre-trained T5.

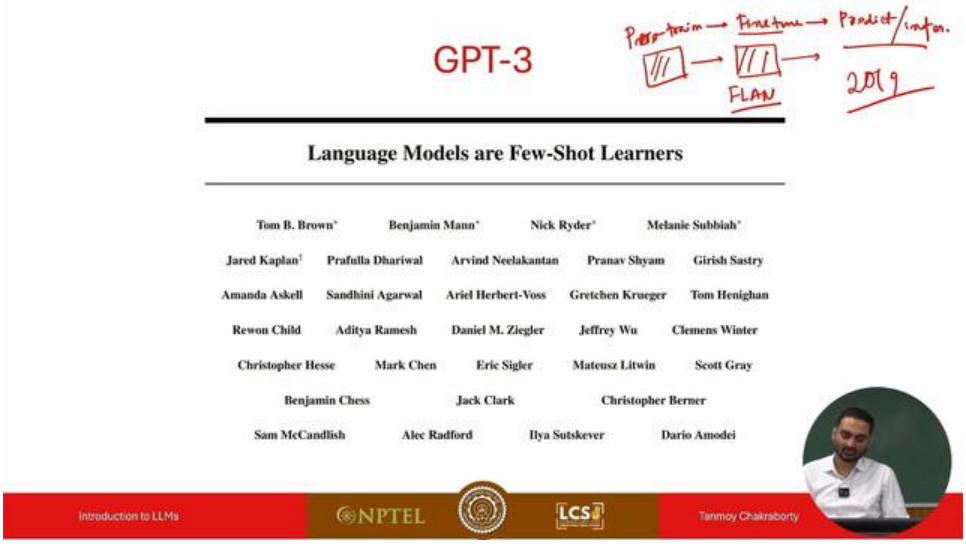
on this corrupted tokens, right? And so, basically, you pass a corrupted sequence and ask the model. To predict the actual sequence. And this is basically trying to learn the overall syntax and semantics of the language. And then you have many tasks. The task can be, let's say, sentiment or common sense reasoning.

Summarization, chat, and so on. There are a bunch of datasets that we use. For example, you can use this Flan dataset. Although Flan is specifically designed for instruction fine-

tuning, you can still use it. You know, the data sets from Flan to fine-tune the model, right? And then you make a prediction.

Okay. So, this is an advanced version of the old-school machine learning paradigm. where you had, where you did not have the concept of pretraining, You had only the concept of training, and then, you know, training and predicting, right? You train the model, and then you predict, right? But here, you now you are, you are incorporating a new step Which is called pre-training. Then you fine-tune. The fine-tuning is similar to the old-school training, right? You just updated some of the parameters for your task.

And then you predict, okay, prediction, or infer. So this has been the paradigm until 2019. Now this GPT-3 paper came out in 2020, and they said, you know, This model is so big that you do not need it to fine-tune. for your downstream task, right? Your model has been pre-trained, hasn't it? And your model knows that the model has been pre-trained on a gigantic data set. So your model has come to know how to solve different tasks.



Right now, you have a downstream task that you want to. Which you want to perform and which you want to solve, right? You do not need the model to fine-tune to get fine-tuned. For this specific task, right? What you need to do is just write a suitable prompt. Okay. The prompt is nothing but an instruction.

You write simple English instructions, right? Assuming that this is a monolingual model, correct? And the model will be able to respond to the instruction. Okay. Remember this

specific instruction or this specific task, which is there in your hand has the, this task may not have, May not have been exposed to the model during the pre-training period. Right. And they observed this, and as the name suggests, language models are few-shot learners.

And what is few-shot? I will discuss it later. Right? So, they observed this in their paper. And they observed this wonderful phenomenon. And they call this phenomenon an emerging property of the model. And they were also very confused.

They were surprised. They were confused. Why did this suddenly happen, right? What is it about the model that made it so powerful? Any kind of task that you want to solve, the model is solving it. Right? Now, of course, the accuracy of these models may not be very high. As compared to a fine-tuned model, right? You have a fine-tuned model for your task, whereas you have a pre-trained gigantic model which has not been fine-tuned, but you write prompts. and this model essentially follows your prompt properly and produces the output versus this output, the fine-tuned model's output. Of course, you might not be able to compare them. And this is not a fair way to compare these two things. Because they are trained in a very different way.

But still, without fine-tuning, you may be able to achieve. Let's say 70% or 80% of the accuracy of the fine-tuned model, okay? And that is huge because, let's say, when it comes to a task. for which you do not have enough training data, How do you fine-tune the model? You don't have enough training data to fine-tune the model. but you now write prompts and your model will be able to solve this very efficiently And that may serve your purpose. But the question is, do you know how good the accuracy of these models is, right? with respect to different prompts, the first question.

The second question is whether the size of the model increases, right? Is this the case that you see this emerging phenomenon? This emerging property occurs only after a certain threshold of the model size is reached. Or, this is a property of a smaller-sized model, right? Can we get a similar prompt? Can we get a similar output with the same prompt? For, let's say, a 7 billion model versus a 175 billion model, right? So these are the questions that we are going to answer in today's lecture, right? Okay, so now, as I mentioned, traditional framework, right? In a traditional machine learning framework, you fine-tune your model

with different examples. So this is example one, example two, example three. Now you give these examples one by one. And your gradient, your parameters of the model, get updated one by one.

So you feed one example; of course, you don't feed one example at a time. You feed a batch of examples at a time. And you update the gradient; then again, you pass a batch of examples. The gradient will be updated, and so on and so forth. So in a normal fine-tuning setup, right, your gradient.

The parameters of some of the model may be. Maybe not all the parameters; maybe some of the parameters will be updated. during fine tuning right. But in case of prompt, your model, your parameters are not updated at all. What do you need to do? You need to write prompts suitably.

And there is no way to understand what the optimal prompt is. for a specific model. You just have to try out multiple prompts and see which prompt works best. You have to quantify what you mean by better. Of course, there are some ways to automatically figure out suboptimal prompts.

But still, it is human-driven, right? Human intervention is needed. Therefore, you know, these days prompt engineering has become One of the important fields for recruitment, for jobs, right? Okay, so now we are talking about introducing a new term. shot learning, zero-shot learning, one-shot learning, Zero-shot prompts, one-shot prompts, few-shot prompts, and so on. What is a zero-shot prompt? In zero shot prompt, let's say you want to Your task is to translate an English sentence into a French sentence, right? So in your prompt, you just write "translate English to French," right? And the word that you want to translate is cheese, for example, right? And then you use some symbols. So in this case, let's say you use this symbol.

Nobody knows whether this symbol is suitable. Either this symbol is suitable or a dash is suitable. Nobody knows, do they? You write this task. So, this is called a task description. You write the task description and then the task that you want to perform.

and you ask the model to perform it. So there is no fine tuning. This is just a prompt. In the case of fine-tuning, in the case of normal fine-tuning, You fine-tune the model, but here you are not fine-tuning it. Here, there are multiple questions that can arise.

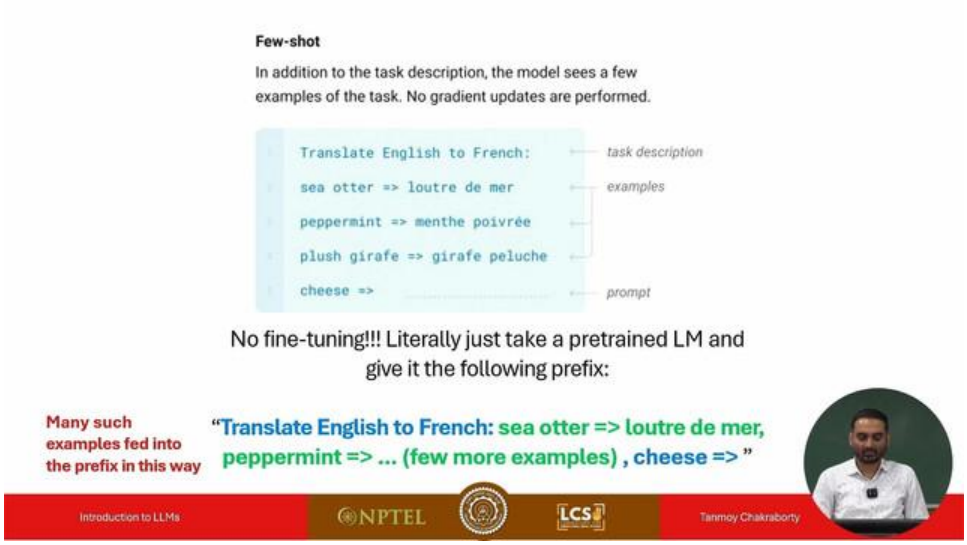
The first question is how to write this task description. So if somebody writes, let's say, "please translate the following English word to French," That can also be a task description. So nobody knows how to write an appropriate task description. and then how to also represent your task. Whether I should use this, you know, equal symbol and then followed by, you know, this greater-than symbol Or some other symbol that we should try out. Okay,

nobody knows. Okay, so later on we will see that, you know, can we. So later on, we will see how to quantify the sensitivity of a prompt. With respect to a model, right? Let's say I change this colon, right, to an arrow, for example.

For this symbol by a colon, right, what would be the impact of this? To the accuracy? And ideally, your model should be robust enough to handle small variations here. And there in the syntactic structure of the. of the prompt, your model should not produce diverse accuracy. It should be consistent, but nobody knows how to quantify this sensitivity. So we will discuss how to quantify this later.

So this is zero shot. Zero-shot means you are not giving any examples to the model. Now, in one shot, you give an example. So your task description is to translate English to French, right? And this is an example. This is your English word, and this is your French word.

And then you ask your model to translate cheese, right? So when this prompt is processed, right? And remember, all these things are happening, you know, in a context set up, right? In the context setup, you were writing the task description and some delimiter. You know the example one delimiter, and you know your task, right? This is one shot because you are now giving one example to the model. Right? You can give more examples: two-shot, three-shot, few-shot. Let's say, in this case, the same task description. Now you are giving three examples to the model.



Right? The more examples you give to the model, the better the model should be able to. Understand things properly. But again, there is no guarantee of what is going to be. The optimal number of examples that one should provide. It should not be, let's say, 1,000 examples.

Right, in your prompt, right? And you also won't be able to write 1,000 examples in your prompts. Because the prompt length is also limited. Because let's say you know an LLM can only access, you know, A prompt length of, let's say, 4K or 10K, right? 10k tokens. So

you are not allowed to give so many examples in your prompt. Now remember, if you have so many examples, You should not use a direct prompting approach.

You can fine-tune the model, and because now, let's say. You have 1,000 examples or 2,000 examples; you can think of fine-tuning. A few layers, right? Later, we'll talk about parameter-efficient fine-tuning. There we will see that we can fine-tune something called the adapter layer. A new layer that you can add to the model.

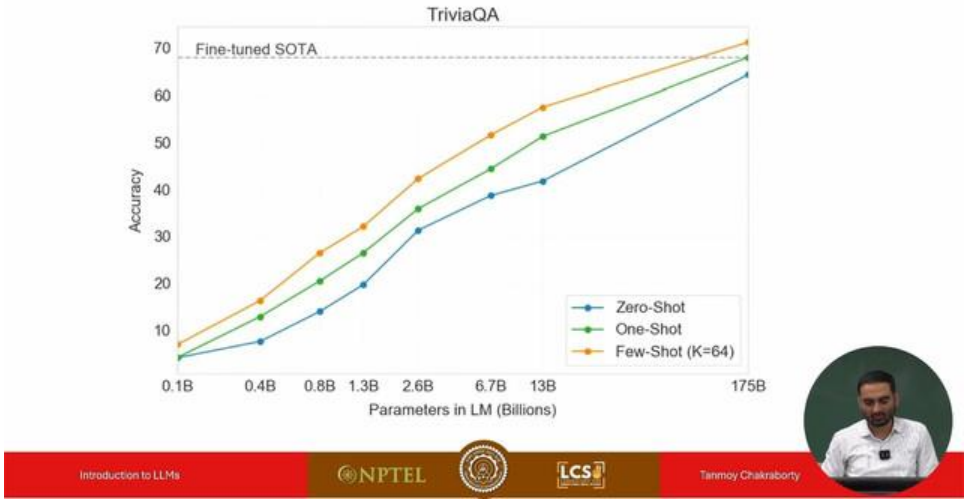
And you can only fine-tune those adapter layers. If you have many examples, If you don't have many examples, you can use zero-shot or few-shot prompting. So, as I mentioned, many such examples will be fed to the model. So, this is your entire prompt, your task description, and some delimiter, right? Your first example, some delimiter; second example, some delimiter; and so on and so forth. And then the task that you want to perform is.

And you ask the model to essentially keep generating tokens. Okay. It may look, if you guys are not aware of this. It may look like a sci-fi movie because, you know, God knows. how this actually happens because you know understanding how prompts work, Right, it is still not fully understood, right.

People have been studying things in our lab; we have also been studying. We have been trying to understand how you know that different prompting strategies work. In-context learning works, but it's not very clear. The science behind this prompting technique is not very clear. So now let's look at this new paradigm: pre-training and fine-tuning, which we have seen, right? And how does the new paradigm, which involves pre-training and prompting, So a new paradigm has emerged called pre-training, prompting, and prediction, okay? As opposed to so fine-tuning, is not needed anymore.

It is repeated by prompting. Okay. Okay. So, how do we check this? You know, zero-shot prompt, one-shot prompt, two-shot prompts out, right? What is the accuracy of different types of prompting techniques? Let's look at this. And for this, let us use this dataset called the Trivia QA dataset.

It is, you know, an open book question answering data. where your model is asked about some of these questions, like Who was the most famous pop deck poppies, and so on and so forth? So these are all standard questions that you ask. And trivia QA is a standard data set. And on this dataset, now you have a model. You have two models and one pre-trained model.



One pre-trained model has been fine-tuned on this specific data. And then you basically use this for inference. So this trivia QA data has been split into two parts. One part is used

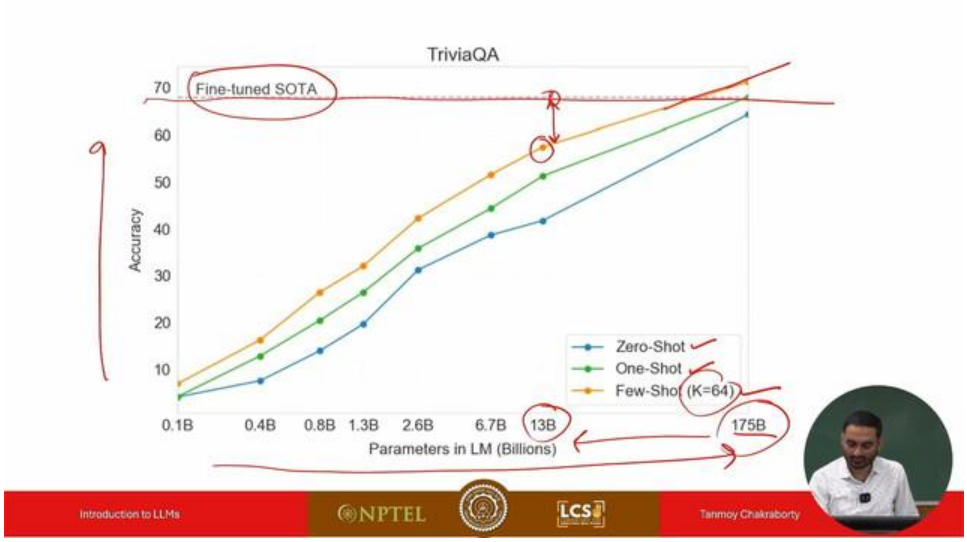
for fine-tuning, and the other part is used for prediction, right? So this is your fine-tuned SOTA model, a state-of-the-art model. Your other model, let's say GPT, is not fine-tuned.

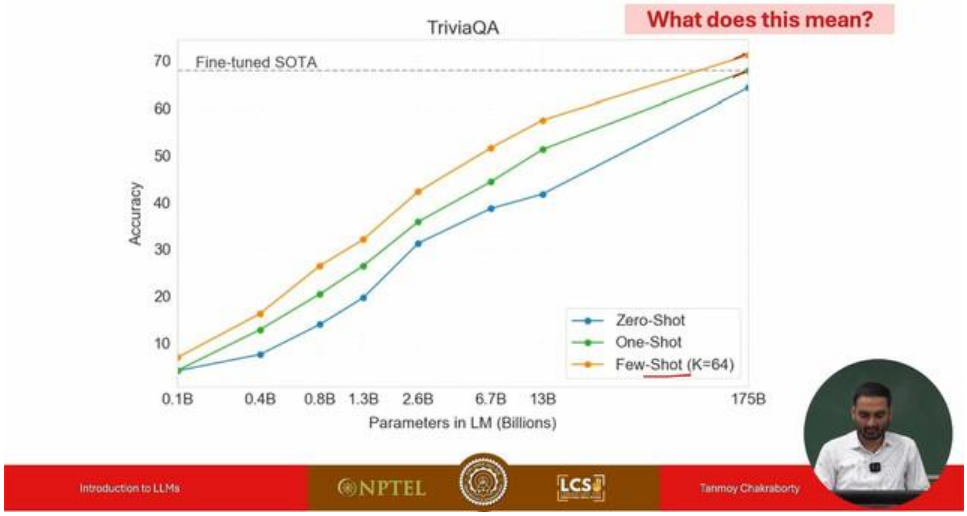
But in one case, you use zero-shot. In one case, you use one shot, and in the other case, you have a few shots. 64 examples are given correctly. So, on the x-axis, you are increasing the number of parameters of the model. from 0.1 billion to 175 billion.

On the y-axis, you see the accuracy, right? So this is the accuracy of your fine-tuned model, and you are trying to compare this. With your non-fine-tuned, prom-based models. As you see, at the higher range, the 175 billion range, The few-shot model surpasses the fine-tuning model. Fine-tuning needs a lot of data, remember. whereas for few-shot you need only 64 In this case, you need only 64 examples to write the prompt.

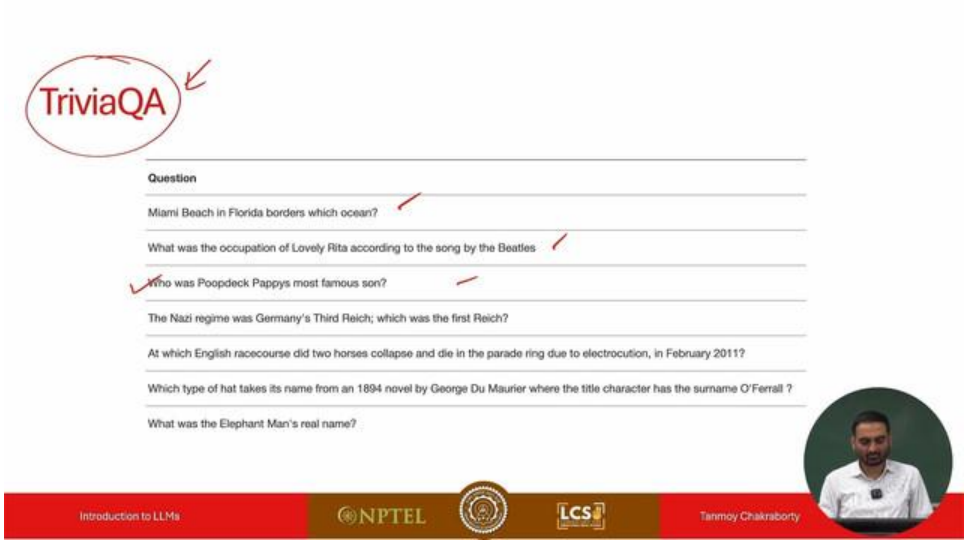
But that model surpasses and outperforms a fine-tuned SOTA, and that is quite amazing. In fact, if you look at a $13 billion model, which is much, much smaller than a 75 billion model and a 13 billion model. A few-shot performance is also significantly comparable to the fine-tuned model. Approximately 60% and approximately 68% to 69% accuracy. In fact, if you look at the...

If you compare the green line and the yellow line, right? Whatever. Yellow, orange, okay. You see, the difference is not that much, right? Not that much, okay? So now let us understand what this means. What do you know about the accuracy of few-shot or one-shot models? you know surpassing a fine tune model mean right. Now, if you think of it carefully, you know this particular trivia QA dataset.





Right? Now the model has been trained on gigantic web data. So all the data that are on the internet have been used. To pre-train these models, right? And these questions are not very unfamiliar questions, right? These questions may have appeared in different locations in this data set, right? Now, when the model was trained on the data, The model must have been exposed to these sentences, not necessarily in this question format. But must have been exposed to these sentences during pre-training. Right now, this is essentially when this question comes.



During prompting, the model needs to locate the right specific parameter. and trigger that parameter which corresponds to this question Although you know I can say this so easily, right? But this is not that easy; remember this ultimately at the end of the day. You have 175 billion parameter models, right? But what I'm trying to hint at is that this data has already been exposed. The questions that you were asking are actually from this data. So it is not extremely surprising if the model produces the right answers, right? What might have been surprising to us was a question.

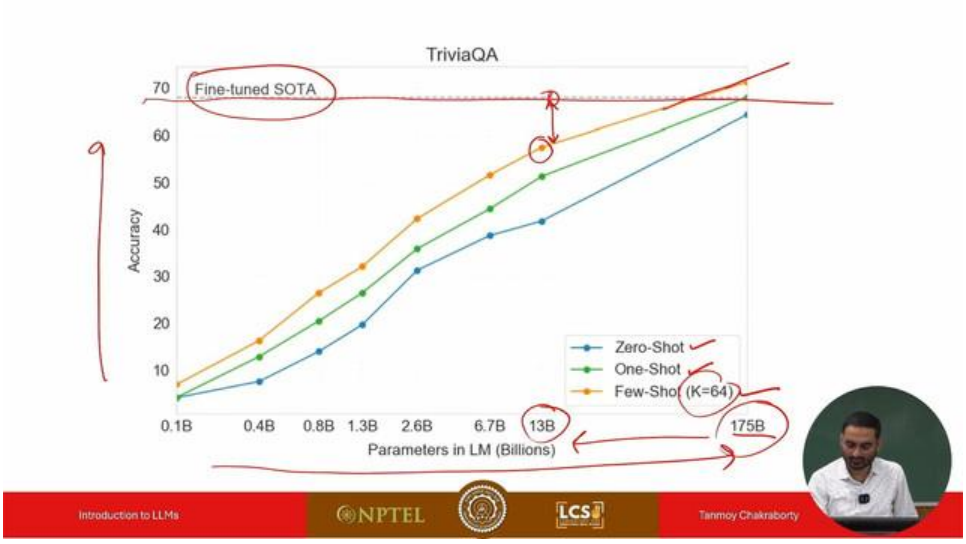
Which is completely out of the book, completely out of the data set, has been asked. And the model is able to accurately answer that question, right? But forming such a question, as you understand, is extremely difficult—enormously difficult, right? It's not possible at all. I mean, given the fact that you do not know. Which data set has been used to train this model? Because all the recent models are black boxes. We do not know which data set has been used, right? Which model parameters, how many model parameters, and so on and so forth.

Okay. Now, if you look at the training data, 7% of the training data is in. Languages other than English. Only 7% of the data is in other languages, and the remaining 93% of the data is in English, right? When you want to do a non-English kind of task, let's say machine

translation, English to French, right? French to English, English to German. German to English, English to Romanian, Romanian to English, right? Let's see what happens.

So this is my SOTA model, which is a supervised model, right? and these models are, now this is supervised machine translation model. This model has been trained, not pre-trained, on this task separately. Whereas models like Mbert and XLM, these models are pre-trained. And fine-tuned on these tasks, right? So this is supervised, and this is pre-trained plus fine-tuned. And these models are only pre-trained.

0 shot, 1 shot, few shot. So if you look at the result, of course you don't expect it. that a pre-trained model with few shots will beat a supervised model. This is not possible. You also should not expect that a few-shot model will outperform.



a fine tuned model. This is not happening, either. If you look at the results very carefully, from English to French, 37% accuracy, 32% accuracy here, 34% accuracy, 39% accuracy here, right? And by the way, the scores reported here are not accurate. These scores are essentially the BLEU score, right? There's something called BLEU score in machine translation. that was used to report the accuracy.

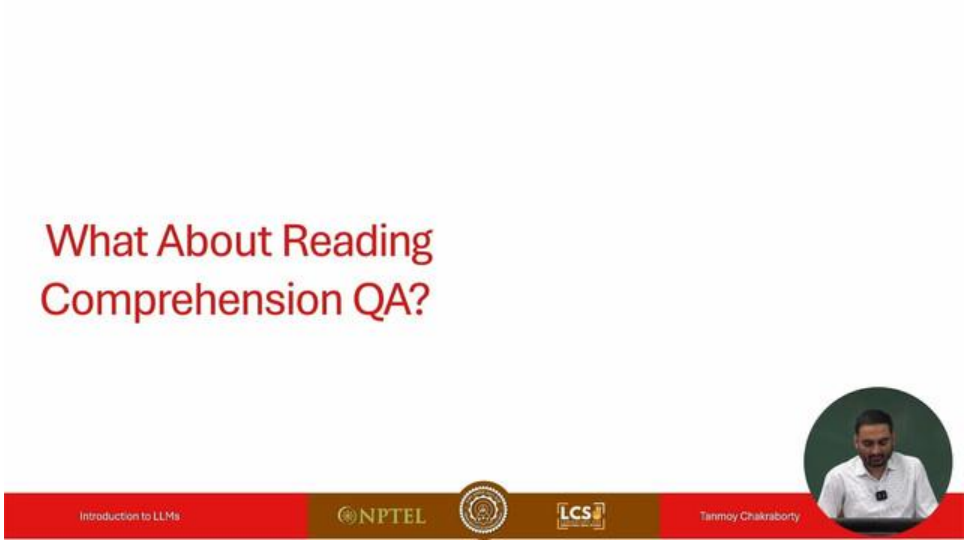
Okay. What numbers are important here? 29.8, 29.7. Here, a few-shot model essentially beats a fine-tuned model, right? 40.2, 40.6, 38.5; this is not doing well, right?

39.9, 39.5, 38.6; very competitive, okay. What I'm trying to say here is that although only 7% of the data is not in English, And there are so many languages; Indian languages are also among them.

Not in a great percentage, but still, they are there in the data. And there are so many languages around the world. But still, the model performs, you know, the GPT model without fine-tuning. performs quite competitively, right, with a SOTA model. So SOTA supervised model and a fine-tuned model.

Okay. Now look at the accuracy, BLEU score versus the model parameters. right, on different machine translation task. As you increase the size of the models from 0.1 billion to 175 billion, correct? Let's say tasks like French to English, this blue line, right? You see the increase, right? So all the dotted lines correspond to translations from English to other languages.

Whereas solid lines correspond to translation from another language to English, okay? So the interesting part to note here is that there is no plateau. The improvement is kind of monotonic, right? You still see, you know, a kind of increasing trend in terms of accuracy. Okay. So this is interesting. Let's see some other tasks.



We have seen the question-answering trivia QA task. We have seen machine translation. Now, let's see, you know, the comprehension QA task. And this is a harder task. This is harder than the previous QA tasks.

And here, you have data sets like COQA, right? Squad data, you know, is very popular, right? And you compare a fine-tuned SOTA model with all the GPT models. Right? Here you see that the accuracy is not very comparable, right? 90.7 and 85 are okay, but 89.1, 36, 74, 44, 93, and 69 are not very comparable, right? So it seems that if the task is harder, All these models, GPT models with prompting techniques, essentially struggle. Therefore, fine-tuning is still a better strategy.

If you have a significant amount of data, set. So this is another plot where they reported the glue score. This is, you know, the GLUE benchmark we talked about, right? Glue super glue benchmark and number of parameters on this axis. And this is the glue score, and the y and x axes indicate the time, right? So you see, you have different models like ELMO, GPT-3, and BERT. GPT-1, GPT-2, ExcelNet, and Megatron LM of different sizes.

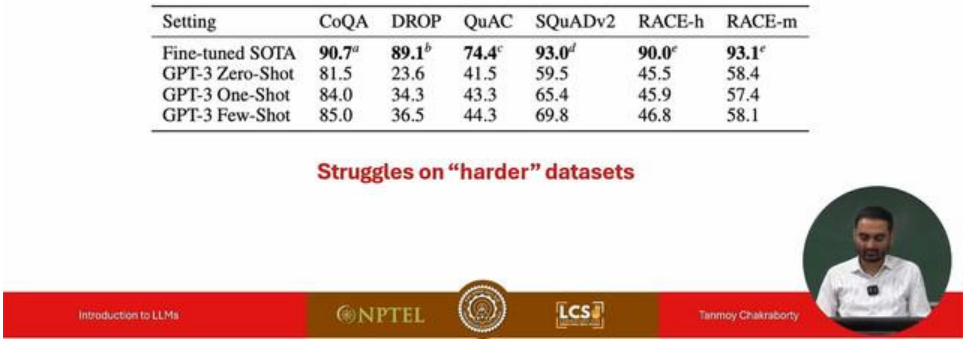
So, the size increases quite exponentially. And with the increasing size, you also see that. So this blue line corresponds to the parameters; the number of parameters shows exponential growth. Whereas accuracy is, if you see, quite linear, right? Accuracy is very linear. Glue scores are linear.

It's ranging from 65 to 95 degrees. So from here, you may say that there is no guarantee. that with the exponential increase in the number of parameters The accuracy will also increase exponentially. If you compare, let's say, Megatron LM versus GPT-2, the accuracy is quite the same. This is approximately 85%, whereas if you look at the model size, This is GPT-2, and Megatron is here.

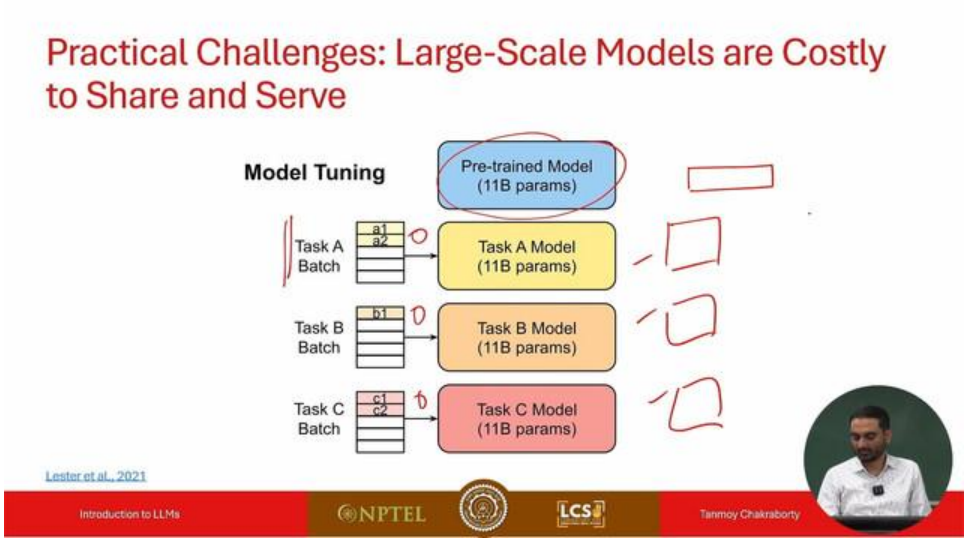
So this is not Megatron; I think this is ExcelNet. Megatron is here, I guess. But it's still right if you look at this point and this point. Accuracy-wise, not a great jump from 85 to 90. Whereas the parameter size-wise is a huge jump, okay.

So it is not that easy, right? I mean, if you just increase the number of parameters. Assuming that you know it will automatically produce good results. This may not be the case. Okay, so what are the practical challenges? Let's see. Okay.

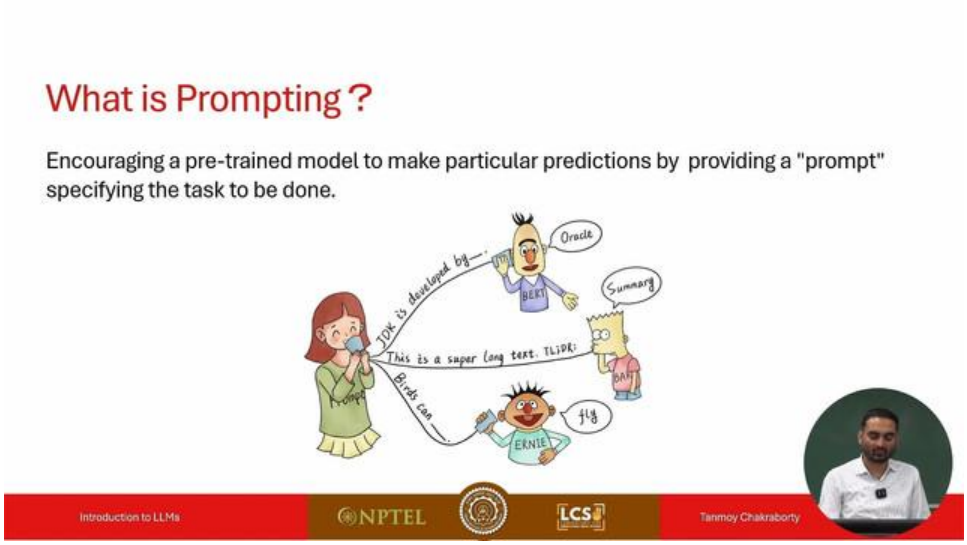
So, such big models cannot be moved very easily, right? It's not like I will upload it to the cloud. And you will download it easily, right? And you will run it on your machine. It's not possible. So, is it possible that we have a pre-trained model, right? And for different tasks, I have specific prompts designed beforehand, correct? And these prompts are essentially indicative of the tasks, Not the entire prompt, but can we think of a prefix, a prompt prefix, right? Which will essentially act as a descriptor of the model for the task, right? And can you basically, you know, train this separately for the task, right? So for task one, this is what task A is: you have a prompt. You have some shot prompts, right? Task B, you have some prompts.



Task C, you have some prompts, and so on and so forth, right? Let's see. Or you can think of what else you can. You can think of three or four different models with three different APIs. People now just call these APIs and get the output. Language model prompting is the solution, as I mentioned earlier.



You can use zero-shot and few-shot. Now let's try to formally discuss what a prompt means. A prompt is essentially an instruction to the model for performing a specific task. by the model. Now, the task can be as simple as what a bird can do? The bird can fly, right? You just wrote, "Bird can dash."



The model will say that the bird can fly. Or let's say some sort of QA-type setup where you ask a JDK is developed by Dash, and the model returns Oracle. Or it can be a summary. So you wrote this is a super long text. TL;DR.

So you ask the model to summarize this one line. and the model will read this and summarizes the text right. So on and so forth, and you see the prompts are very diverse, okay. So let's look at the terminology that we generally use. And these are very formal terminologies.

I don't think these are very important to remember. But just for the sake of completion, let us see. Your input is X, denoted by X. Your output is Y. Let's say your input is a sentence that you want.

The model to find the translation for, sorry, the sentiment for, right? Your input is a sentence; let's say, "I love the movie." And your output is positive sentiment or negative sentiment, right? You have a prompt function. A prompt function is also called a prompt template. Okay, and it looks like this. So, this x is the input, right? Overall, it was a Z movie.

So, Z is another placeholder; x is a placeholder. Z is a placeholder, x is for the input, and Z is for the output, right? So let's say you have many examples, right? And you write this way. So I hate this movie. This is the input. I hate this movie.

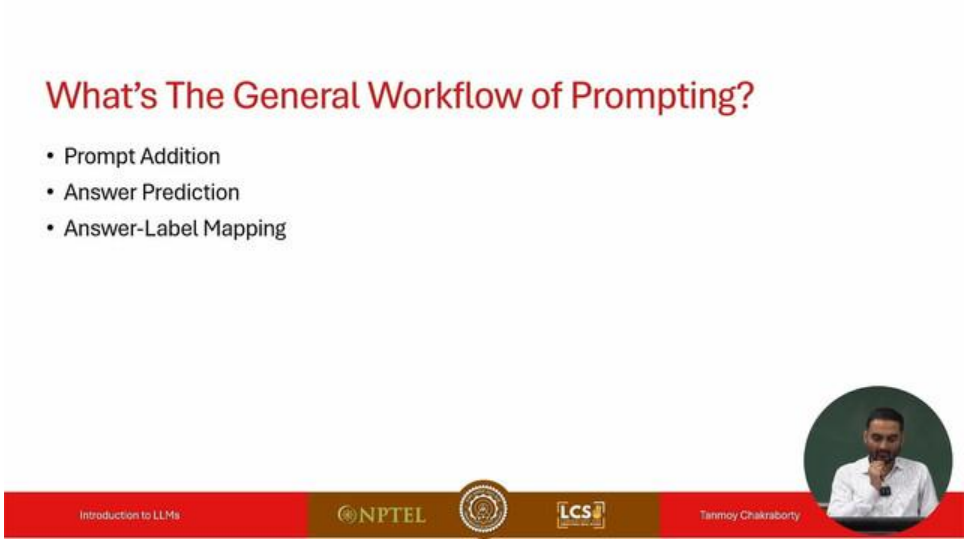
This was a bad movie. I love this movie overall. It was a good movie, wasn't it? And so on and so forth. So you have the template, the inputs, and the outputs. You just, you know, replace the placeholder with the input.

And the placeholder Z by the output, right? So your prompt looks like this. Okay. This is a field prompt. I love the movie. Overall, it is a bad movie right now, isn't it? This here Z will be filled by the word bad.

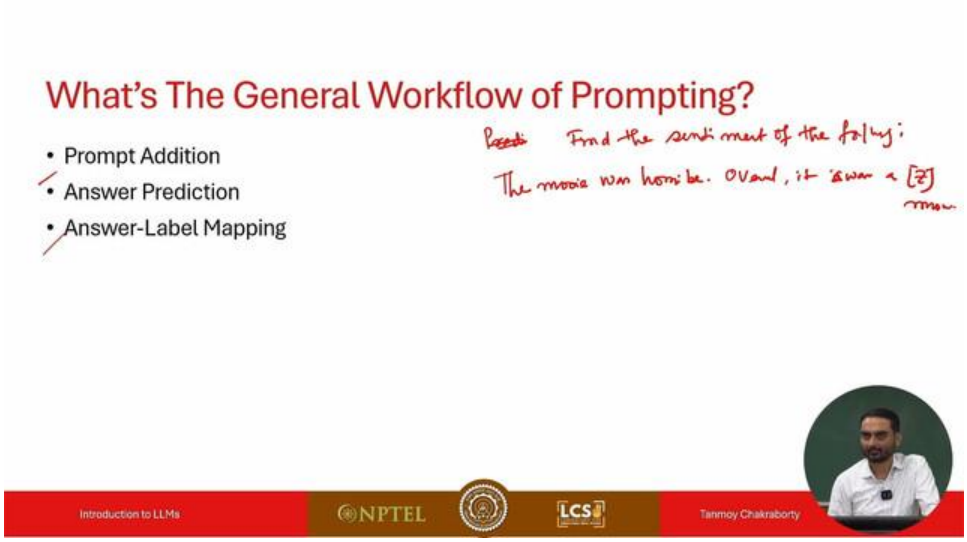
Okay. It can be good. It can be bad. And you, right? So you feed this as your prompt to the model, right? And you ask the model to essentially fill this placeholder. and your model

will fill this by the word good, Which is the answer prompt, okay? And what are the answers? Answers can be good, fantastic, or boring, right? You can also define these answers in your prompt.

Now, let's look at some of the flows, shall we? Prompt addition, answer prediction, and answer label mapping. What does it mean? So, let's say your task is to predict the sentiment. So you say that or find the sentiment of the following, right? Then, colon, and you have the template right. Overall, it was—dash—right. And you write in this way: the movie was horrible; overall, it was a Z movie.

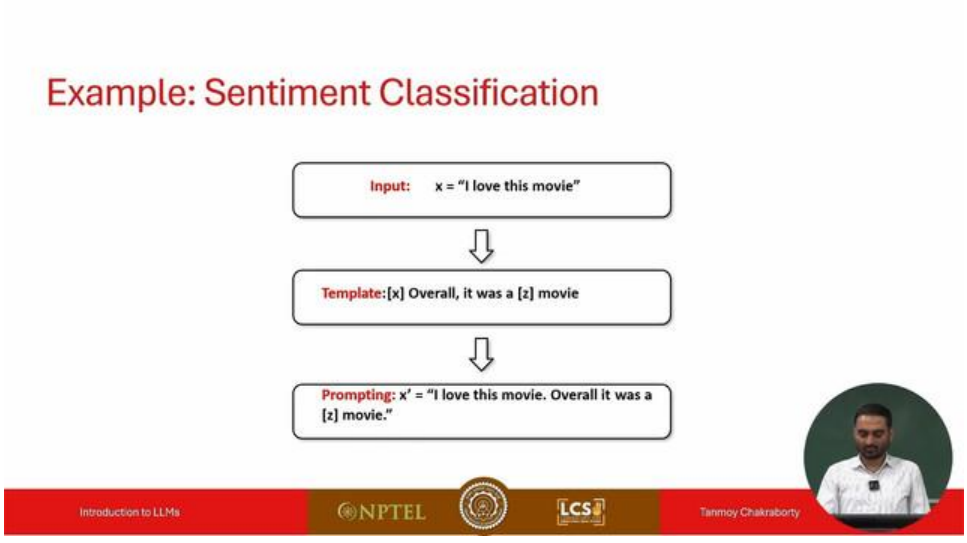


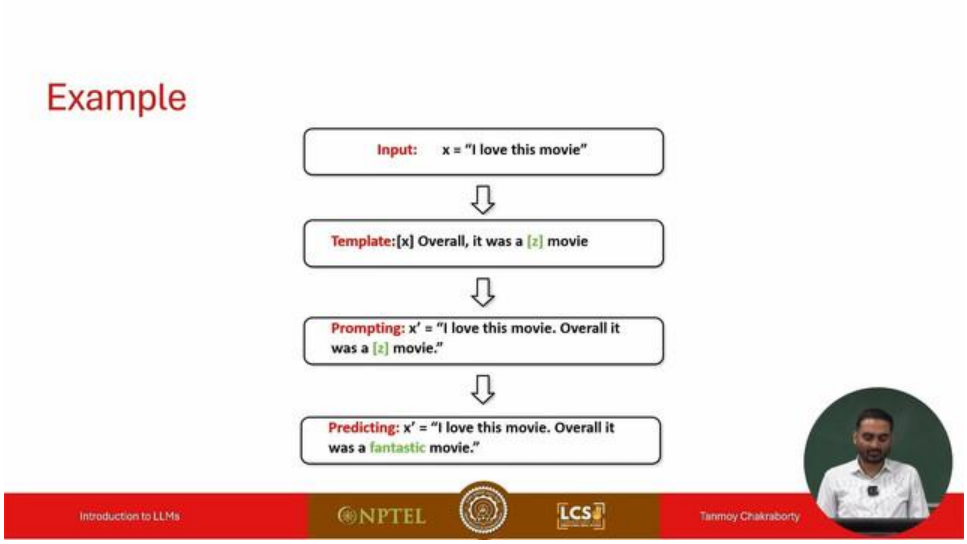
Okay, and you give it to the model; God knows. What the model predicts, right? What the model produces? Let us say the model produces that yes, it was indeed a horrible movie, okay. Okay, now this is your prompt addition. This is your answer, isn't it? But remember, you have only three levels: good, fantastic, and boring. So your task is to map this answer to one of these labels, correct? Now, in this case, it will be boring.



This will be boring, right? So how do we map this? This is also something that we need to figure out. And this is called answer-label mapping. Okay. By the way, today's models are very, very sophisticated. You don't need all three of these components.

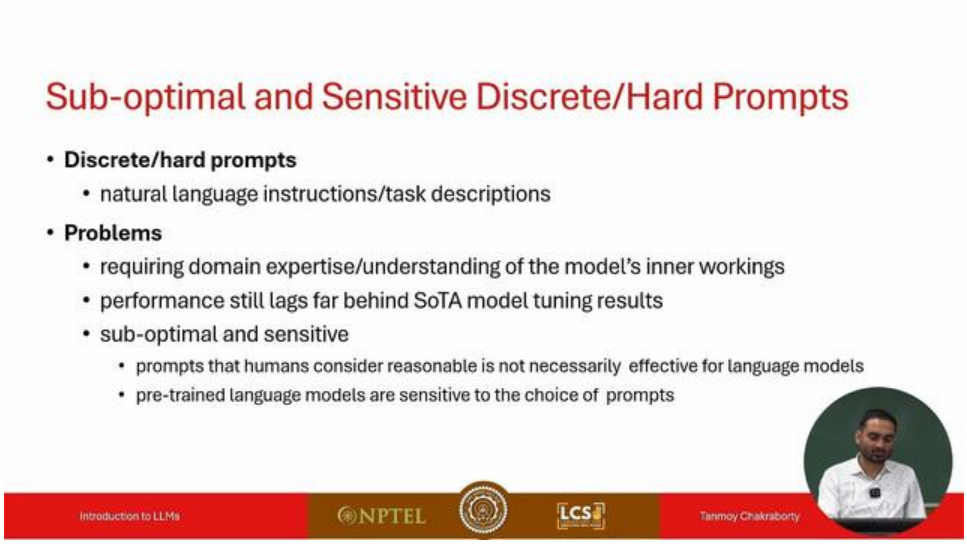
Now I'm talking about models that were there in 2020 and 2021. They were not that great in terms of different prompts. But you know, that time you had to have three different components, right? Okay, so for prompt addition, you can figure out, I mean, You can think of templates. Right? The one that I mentioned, this is input; this is template. And these are your prompts, right? For answer prediction, you know, this is your answer.





That you obtained from the model, right? And how you map fantastic to positive is something that you want to decide. So you can have another head here, which will, you know, Take this as your input and predict, let's say, two outputs. One of the two outputs, positive and negative, and so on, right? There are two types of prompts. One is called a field prompt. So one is called a prefix prompt.

Another is called a cloze prompt. In the prefix prompt, you add this output placeholder at the end of your sentence. Assuming that your model is an autoregressive model, right? Your model will keep on generating berts, Whereas close form your placeholder can come anywhere in the sentence, right? So the prefix prompt will be useful for GPT models. And this kind of model prompt will be useful for BERT, right? Most of the prompts are hard prompts, natural language instructions, and human-readable instructions. The problem with the hard prompt is that we do not know.

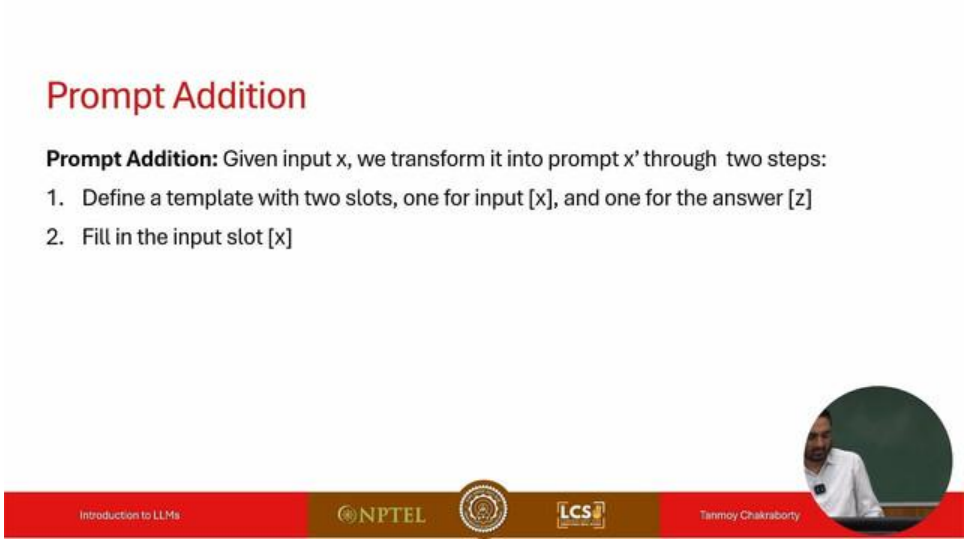


What's going to be the optimal prompt, right? So, people move from a hard prompt to more of a soft prompt, right? And why is a hard prompt even problematic? Let's look at this example. So this is an accuracy table where you write a prompt in four different ways. Right? So X is located in Y. Again, X and Y are placeholders. If you write in this using this template, you get 31% accuracy.

Accuracy is precision, isn't it? If you write the same thing in a different way, X is located in which city, country, or state? Yes, you get 19% accuracy, right? If you write it this way, X is located in which country? Question mark in Y, right? You achieve 51% accuracy. Nobody knows why this is happening. So the models are sensitive to different types of hard prompts. That's why people thought that we should try to shift towards a soft prompt. where you have prompts that a human won't be able to read Because these are vectors.

Prompts are going to be vectors, and humans won't be able to read them. But after all, what matters is how the model performs. It doesn't matter what humans need or not, But what matters is whether the model produces the right answers or not. Okay. Okay.

So we are now moving from discrete prompt to the continuous prompt. And during 2020 and 2021, there have been enormous efforts in designing manual prompts. mining prompts, paraphrasing prompts, gradient-based search for hard prompts. People have tried out if you freeze most of the models. Only keep some of the model's parameters for your downstream tasks.

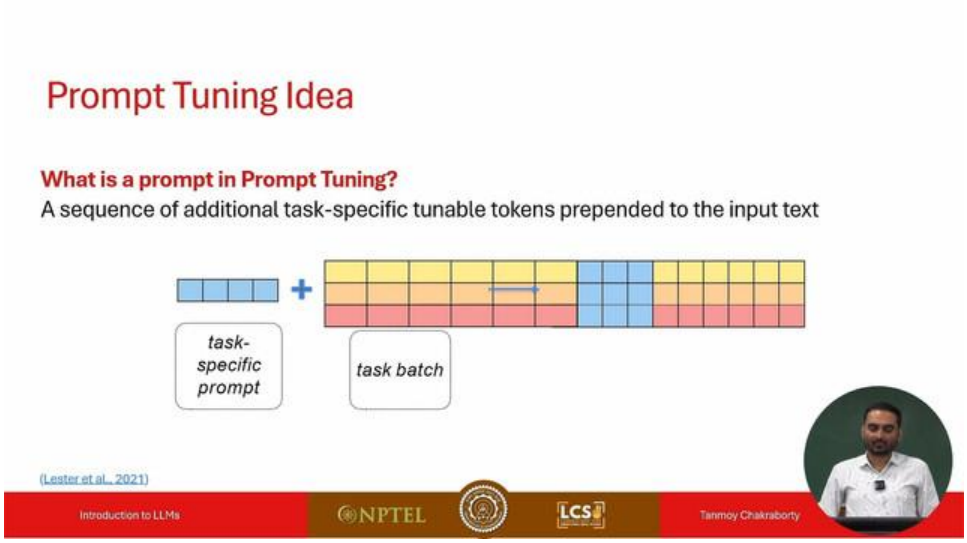


Given a set of words, you ask the model to choose. One of these words one by one and see how the gradient flow happens. and how the accuracy is affected and so on. And then you decide which word is suitable for your prompt. You know, auto-prompting and so on and so forth.

But you know, these things are not very suitable or useful these days. So we are now moving towards a continuous or soft prompt. right, where we will essentially produce embeddings as prompts. Okay. So we will discuss two ideas.

One is called prompt tuning, and the other is called prefix tuning. Okay. In prompt tuning, what you do? When you write the task description, write in English, right? You prepend a

set of parameters, a parameter vector, for example. Right, what's going to be the size of the vector is again a hyperparameter. But you prepend a new parameter near the new set of parameters.



Before the task description. So translate from English to Hindi. But before that, you prepend it, right? Translate English to Hindi, and then blah blah blah, okay. But you have some parameters here. Now these parameters are randomly initialized right.

You can think of better installation techniques, but I will show you later. that random installation is good enough right. And let's say you have a T5 model, right? Okay. This is encoder, this is decoder. and your output is a translation, Hindi translation, right.

You give it to the model, okay? All the parameters of the models are frozen. Only this set of parameters are unfrozen and when you, So let's say you have a tiny data set for finding the optimal parameters right here. And this tiny data set is being used for fine-tuning, right? And here, when I said fine-tuning, don't mix it up with actual fine-tuning. Here, fine-tuning means that you are only fine-tuning the prepended embedding, right? So you compute a loss here; this loss gets backpropagated. End to end till the input.

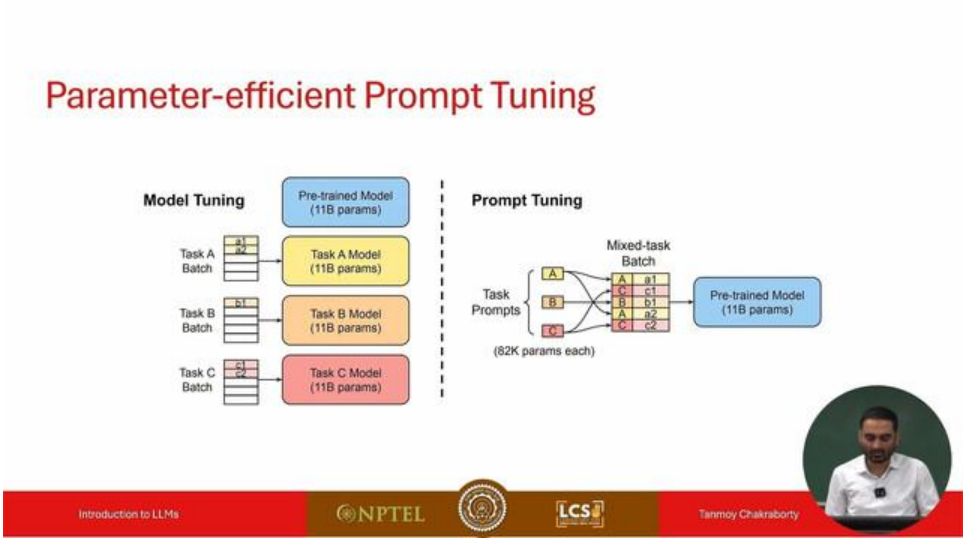
None of the parameters will be updated. Only these parameters will be updated. Okay. So this is called prompt tuning. And why this is important is that it is important because now let's say for translating English to Hindi, you have one specific prompts, shot prompts.

These are called shot prompts, right? Let's say, for summarization, you have another set of shot prompts. which you basically fine-tune. For QA, you have another set of shot prompts, and so on and so forth. Now, you can actually share these shot prompts, can't you? From one computer to another very easily because this is a small vector. Right? And in this way, you can essentially see the entire model.

you just replicate the model, you share the prompt parameters, and this prompt parameters act as task specific parameters. And then, you know, I mean, as if these parameters correspond to this parameter, The set of parameters corresponds to the task that you want to perform, right? Now, so this is prompt tuning 2021. Again, around the same time, in 2021, something called prefix tuning was proposed. The same idea; the only difference is that in the case of prefix tuning, You add or prepend these parameters not only with the input, But also with all the other layers. So every layer has an additional parameter that is prepended, right? So you are increasing the number of parameters, but that's okay.

Because this parameter size is not huge, but when you fine-tune, You only fine-tune these parameters. These prompt tuning and prefix tunings also come under the Overall, the big umbrella of parameter-efficient fine-tuning. We will discuss it later, right? Parameter-efficient fine-tuning, PFT, okay. And we will see later models like adapter; know adapters are of different types. Let us say LoRa; some of you have heard of these terms, right? And LoRa, AdaloRa, and so on and so forth, we will discuss later.

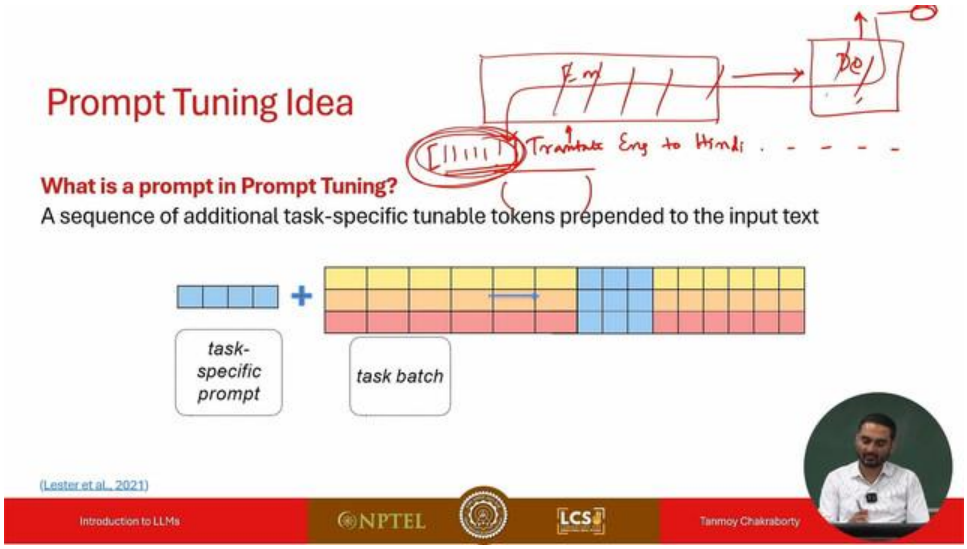
What we have learned is that we wanted to share our model with others, right? What can we do? You can share the prompts that we learned during the fine-tuning process. and these prompts are going to essentially describe the task that you want to perform. Of course, you need to write suitable descriptions. But these prompts will be suitable for the models to essentially understand the task, okay? Let's look at some of the results. X-axis: number of parameters, Y-axis: accuracy score.

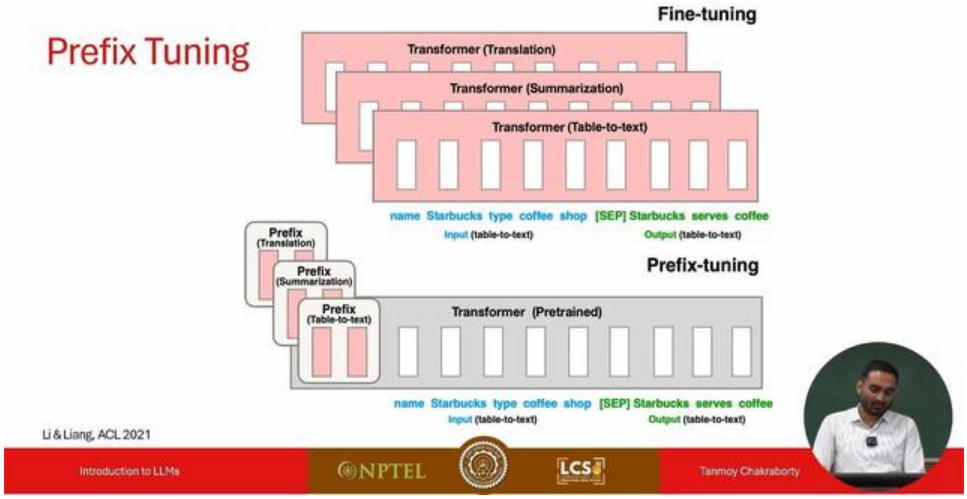


And here they use this superglue benchmark, right, we discussed earlier. The blue line is a prompt for manual design; this is manual. The orange line is model tuning. You fine-tune the model, right? Remember, when I say fine-tuning, this is full fine-tuning. The green line is prompt tuning.

It can be a prefix, right? Prompt tuning—I mean, prefix tuning—prompt tuning we discussed, right? This is the kind of model. So, manual prompt and manual prompting are not great ideas in terms of accuracy. As you see here. Of course, full fine-tuning is the best, but the green line, which is prompt tuning, It comes in between full fine-tuning and manual tuning, right? With the increasing size of the model's parameters, Prompt tuning matches the output of full fine-tuning. This is very important, right? Remember, for full fine-tuning, you need to fine-tune a lot of parameters.

Where in prompt tuning do you only need to fine-tune a few parameters, right? So, with the increasing size of the models, you may be able to achieve the same accuracy. as that of a fully fine-tuned model, okay? Okay, and you see here kind of a variance, right? It basically indicates that this accuracy is not stable, right? So in the middle zone, right in this zone. Accuracy can vary if you change the prompt. I mean, if you change the, you know, the data, right? The task's accuracy may vary. Okay, let us look at the length.



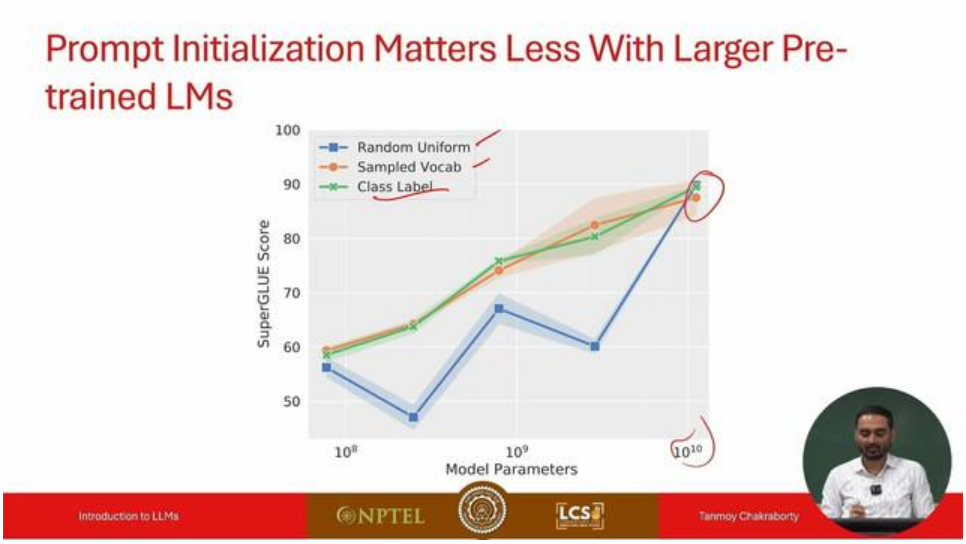


So I said that you needed to prepend something, right? Let's say you prepend only one embedding versus five embeddings. 20 embeddings, 100 embeddings, and 150 embeddings, okay? With the increasing size of the models, let's see what happens, okay? One embedding is not a very good idea, is it? Five embeddings are better than one embedding, right? But between choosing 20, 100, and 150. You see, these lines are almost the same. So there is no point in choosing; there is no point in adding 150 parameters. 150 tokens, right? As a prompt, you can simply add 20 or 30 additional tokens.

That is good enough for your model. So, I talked about initializing these parameters, right? So this experiment said that if you randomly initialize these prompt parameters. Versus

using some sample vocabulary, let's say you choose some words. which you feel are suitable for this prompt and you take the corresponding embeddings From, let's say, "word" to "vehicle" glove, and you initialize this. What happens if you are even, you know? More advanced, you want to take more class labels if you want to take.

Right, the labels that you want to predict as your prompts, right? What will happen? So small models, of course, sampled vocabulary or class labels are much better than Random initialization. But for bigger models, it doesn't matter. You can start with random initializations. It will produce the same output as that of the sample vocabulary. and class labels that you can use.



The problem with soft prompting is that it's not at all user-friendly. You will not get to know what has happened within these parameters. These are not interpretable. Therefore, hard prompts, although they are not a very good solution, are still used in certain contexts. But hard prompts are still chosen as the default for writing prompts.

Okay, in the next module, submodule I would say, I will discuss a few advanced prompting techniques. and then we will discuss the way to you know measure the sensitivity of a prompt. Okay, thank you.