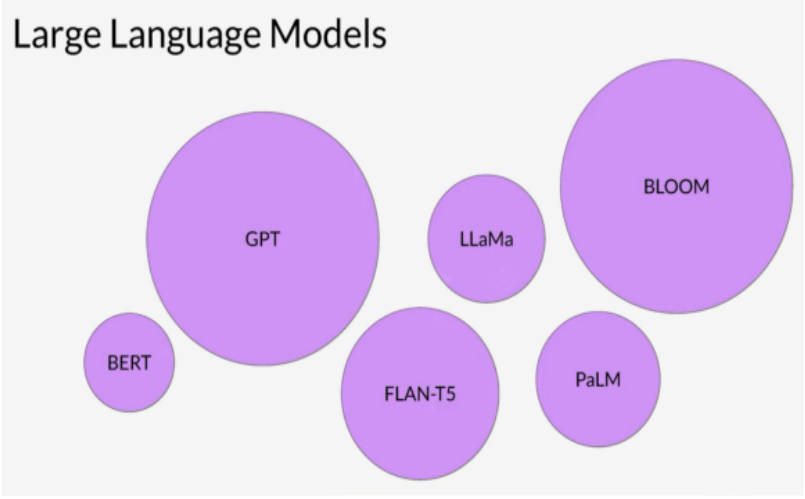
LARGE LANGUAGE MODEL

Notes by: Nikhil

Generative AI is a subset of traditional machine learning.



→ The models shown in the picture are the foundation models also called base models.

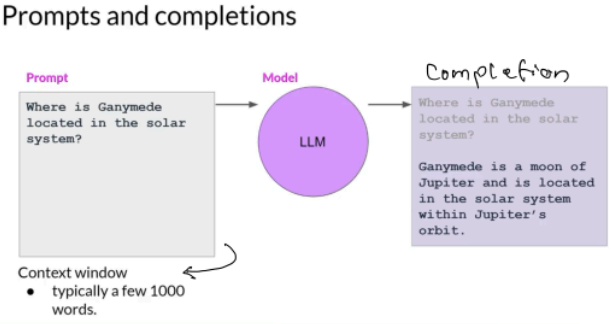
→ The size of the circle shows how big a model is

→ The size of the model is based on parameters.

→ The more parameter it has the more memory it has and can perform more sophisticated task.

→ The text or instruction that we pass to the LLM is basically prompt.

→ The text or instruction that we pass to the LLM is basically prompt.



→ The output of the model is called completion.

→ The act of using a model to generate output is called inference.

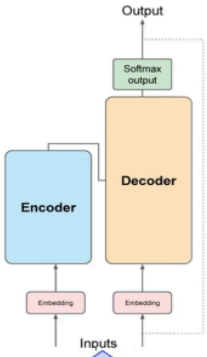
→ LLM is not a very new concept the earlier way to predict or generate next words or sentences was acieved through RNN Recurrent Neural Network.

→ But the limitation the RNN was with the compute power and only few previous words can't predict next word accurately.

→ This limitation was over come by Transformer introduced by google.

→ Now let's understand how transformer works :

Transformers



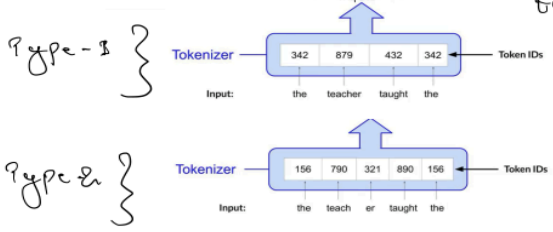
The transformer architecture is split into two two distinct part
↳ Encoder
↳ Decoder

→ Machine learning is basically a massive calculator and it understand numbers also called tokens but not words.

→ Now to work with this we need to first convert words into tokens

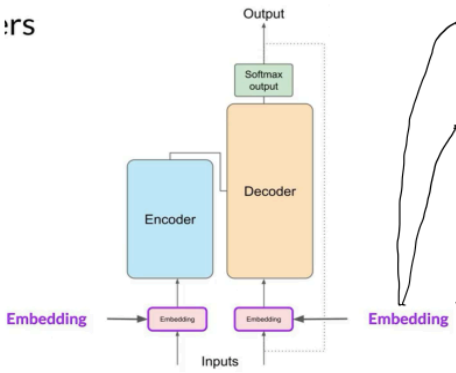
→ There are two ways to create tokens

i) You can either use complete words to create token IDs



ii) You can use parts of words to create token ID's

→ Once the input is selected with the anyone of the token id's then it is going to be used throughout

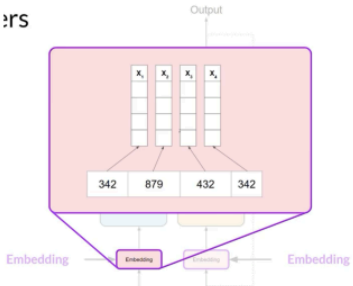


→ Once the input is converted into tokens then you can pass it into embedding layer.

→ This layer is a high trainable vector space and in this layer each token is represented as a vector and occupies a unique location in a space.

→ These vectors learn to understand the meaning and context of individual tokens in the sequence of input text.

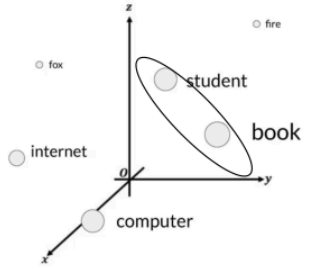
→ Traditional NLP uses this concept like word2vec

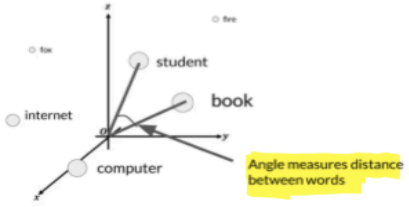


→ In this case you can see that each words is matched to a token ID and each token then mapped into vector.

→ In the original transformer paper the vector size is mentioned as 512.

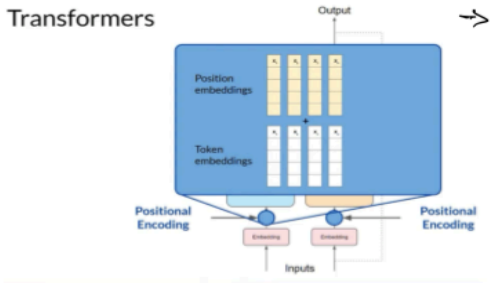
→ Now let's take a example if the vector size is 3 the we can plot the words into 3 dimensional space and understand the relationship based on how closely the words are placed into vector space.



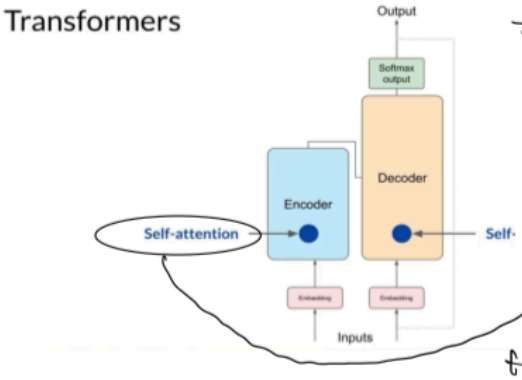


→ To calculate the distance between is by calculating the angle.

→ The model also adds positional encoding.



→ The model proces all the tokens parallely and the use of position encoding help the model to remember the position in the sentence.

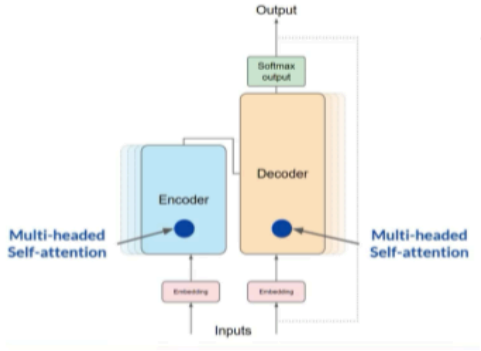


→ Once we combine the input token and the positional encoding then it pass to the self attention layer.

→ Here the model analyze the relationship of input token into your input sentence.

→ The attention weights are learned during the training and stored into this layers reflects the importance of words into input sequence with all the words into the sentence.

→ The attention weights are learned during the training and stored into this layers reflects the importance of words into input sequence with all the words into the sentence.

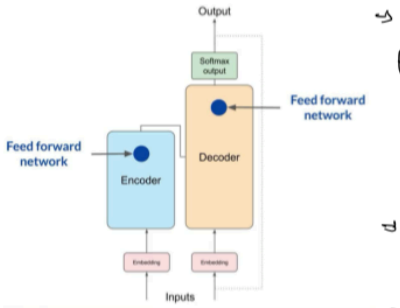


→ The learning of attention weights are not done only once but it create multi headed attention layer to learn the different weights parallely and independent of each other.

→ And multi-headed self attention defends our model to model but ranging from 12-100

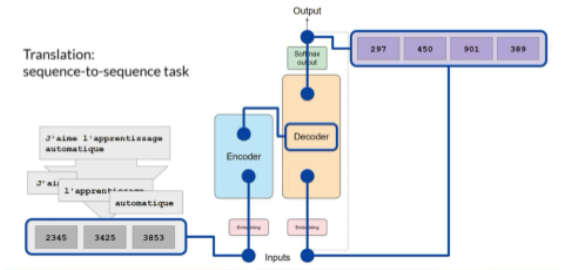
→ The idea of multi-headed is to learn every aspect of sentence like one layer learn about entity other layer learn about behaviour etc.

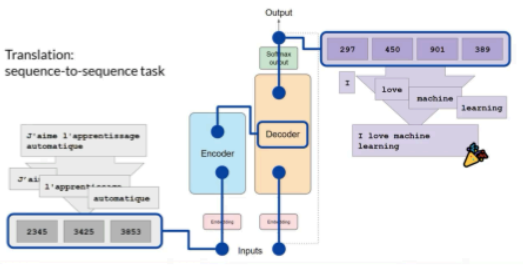
→ The weights of the layer assigned randomly and learn during training.

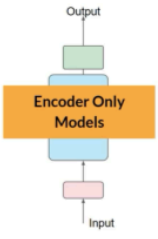


→ Now that we have weights learned during training then the input is processed through a fully connected feed forward network

→ The output of this layer is a vector of logit proportional to probability of each word and then this logit is passed to the softmax layer.







→ So far we have seen architecture of encoder and decoder model however we can also use encoder only model or decoder only model.

→ Encoder only model is an example of seq-to-sequence model.

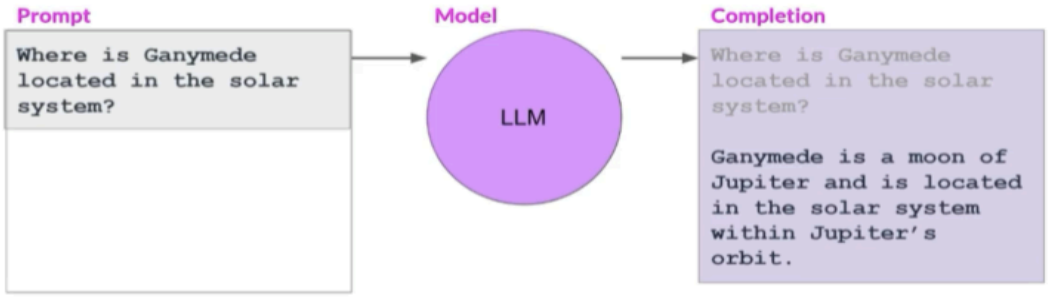
→ BERT is an example of encoder only model, also used for sentiment analysis.

→ The text that we feed into the model is called prompt

→ The act of generating text is called inference and the output of the model is called completion

→ The memory available to use the prompt is called context window.

Prompting and prompt engineering

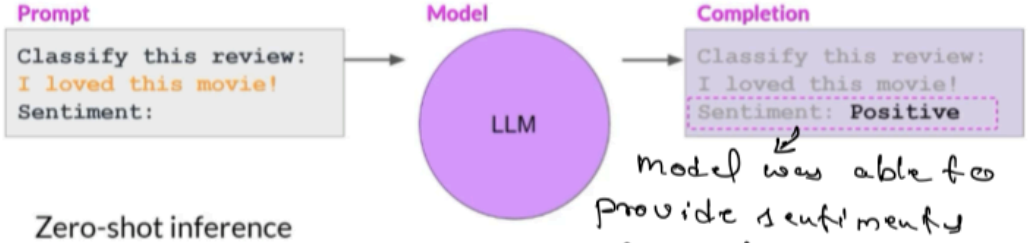


Context window: typically a few thousand words

→ It's not necessary that the model would behave as per you on the first try but if you want this to happen you must provide prompt multiple times to get the desired output. This work to improve and develop model is known as prompt engineering.

→ Providing example into the context window is called in-context learning.

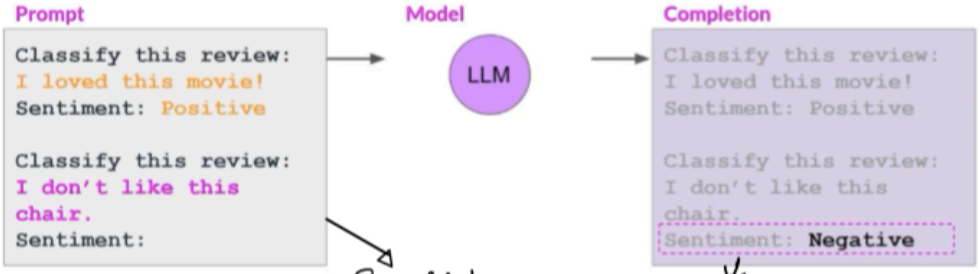
In-context learning (ICL) - zero shot inference



Zero-shot inference

model was able to provide sentiments correctly.

In-context learning (ICL) - one shot inference



One-shot inference

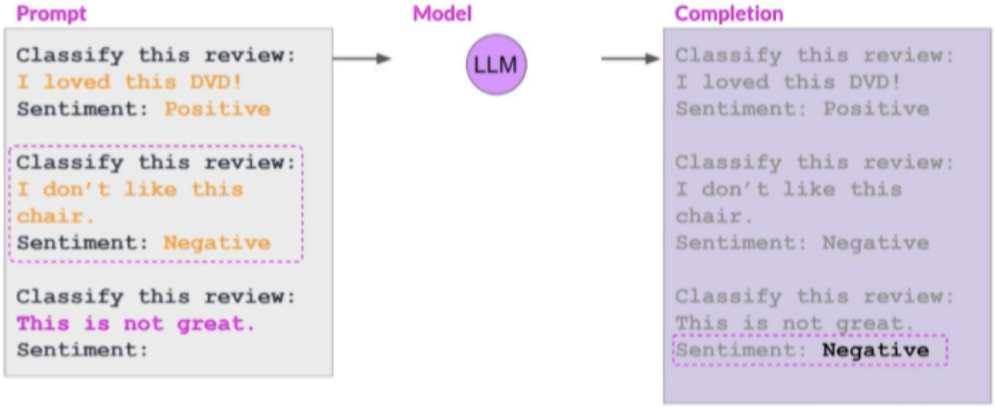
In this example we have provided one example for the model to understand it more accurately.

The inclusion of single example into context window is known as One-shot inference.

→ Sometimes single example won't be sufficient for the model to understand so in that case we can include multiple example and this is called few-shot inference.

→ larger model won't require one-shot or few-shot to produce better output however smaller model require one or few shot to produce completion accurately.

In-context learning (ICL) - few shot inference



Summary of in-context learning (ICL)

Prompt // Zero Shot

```
Classify this review:
I loved this movie!
Sentiment:
```

Prompt // One Shot

```
Classify this review:
I loved this movie!
Sentiment: Positive

Classify this review:
I don't like this
chair.
Sentiment:
```

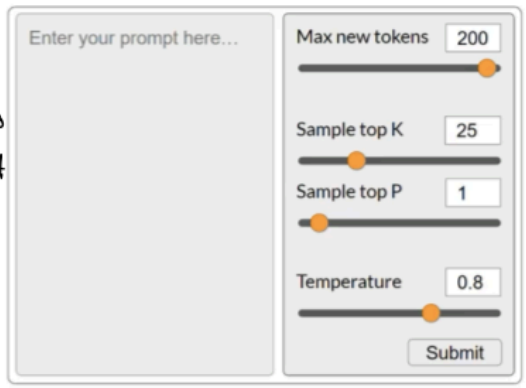
Prompt // Few Shot

```
Classify this review:
I loved this movie!
Sentiment: Positive

Classify this review:
I don't like this
chair.
Sentiment: Negative

Classify this review:
Who would use this
product?
Sentiment:
```

→ The limitation of in-context learning is the memory size of context window if you realise that even after providing lot of example if your model is not providing better output in that case you have to fine-tune your model.

→ BERT was trained on 110M and BLOOM was trained on 176B parameters.

Generative configuration - inference parameters



Inference configuration parameters

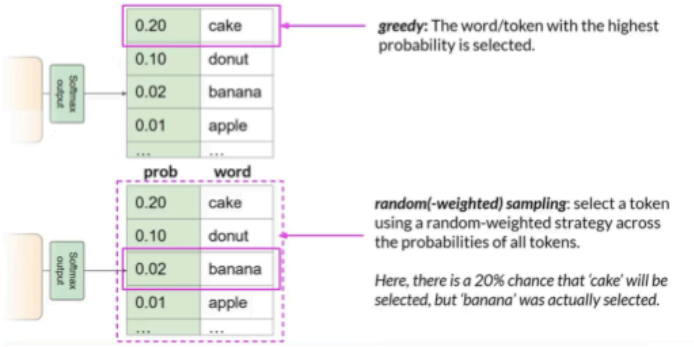
But these inference parameters are different than the training parameters which was learnt during training.

Inference parameters are influenced during inference to provide better output.

Generative configuration - inference parameters

* max new tokens basically gives you maximum words / tokens in the completion.
* This can be used to limit the tokens in output or you can think this as putting a cap on it.

Generative config - greedy vs. random sampling



→ The output of softmax layer is the probability distribution across the entire dictionary of words that the model choose.

→ Most LLM operates by default on greedy sampling in order to predict new words this is the simplest technique where the model chooses the word with highest probability.

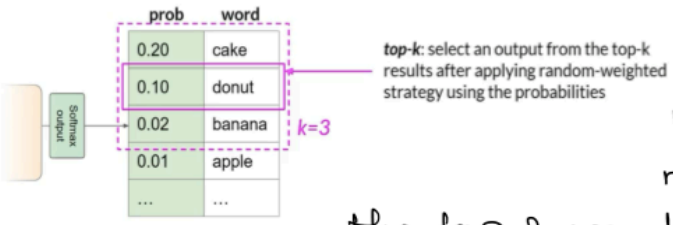
→ The second approach to introduce the variability is random sampling to select more probable words with no repetation of words.

→ Based on the requirement either we need to disable greedy and enable sampling.

Generative configuration - inference parameters

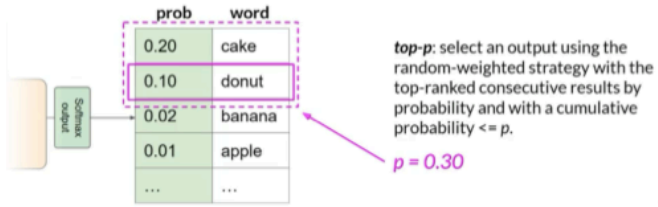
Top k and top p are two sampling techniqe that help limit the random sampling and produce more sensible.

Generative config - top-k sampling



→ In this example you are setting k=3 which means you are restricting model to select between the top 3 words and the probable selection is "donut" here. By this method model has some randomness while avoiding to choose from more improper selection.

Generative config - top-p sampling

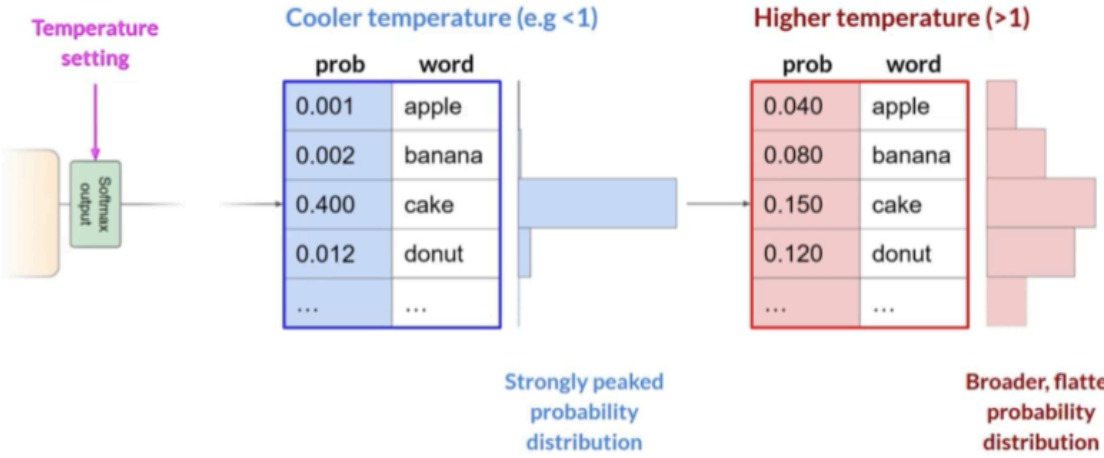


→ The other method is to set the value of p and based on that the words are randomly selected.

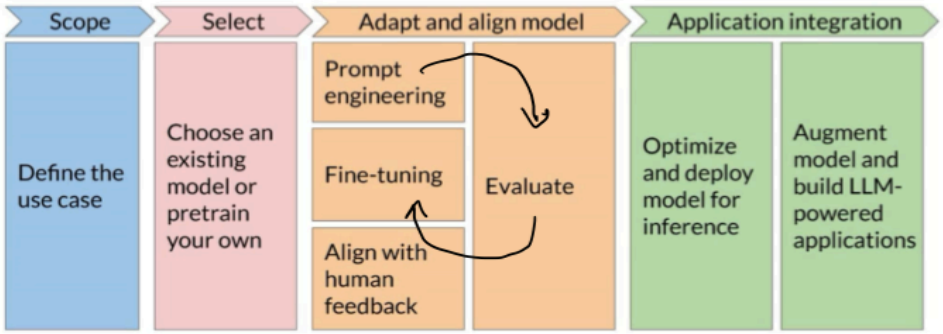
→ In this case the probability is set to 0.30 so the model has selected two words that fall within the probability and then it perform random selection.

→ One more parameter is temperature which control the randomness means higher the temperature higher the randomness and lower the temperature lower the randomness.

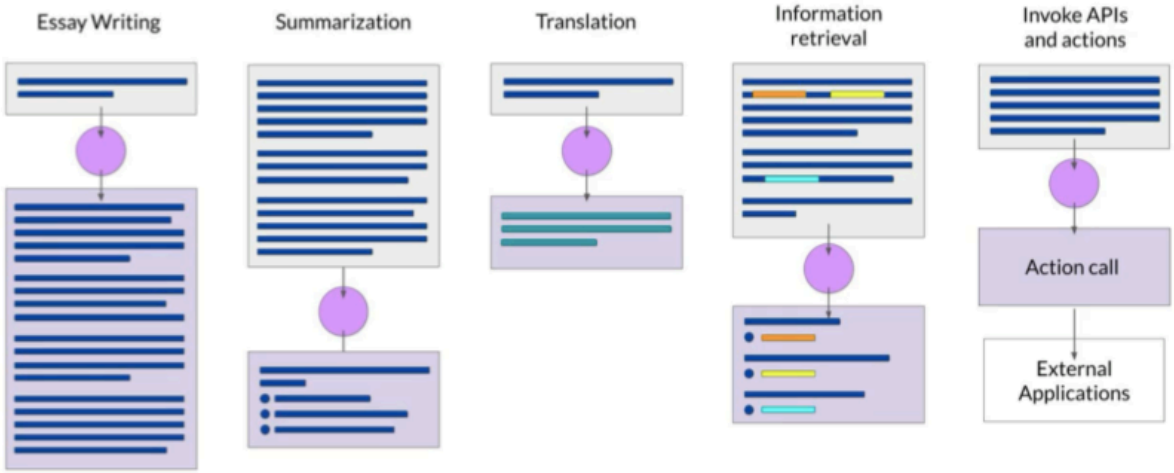
Generative config - temperature



Generative AI project lifecycle



→ The first part is to define or decide on the use case means what is the problem you want to solve. Below are the examples of multiple use cases :-



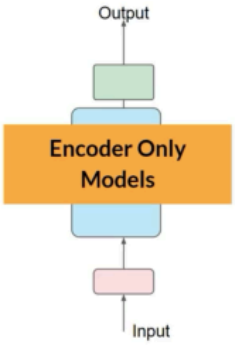
→ The second part is to check if we want to train our own model from strach or we can use the existing model. In general we use existing model.

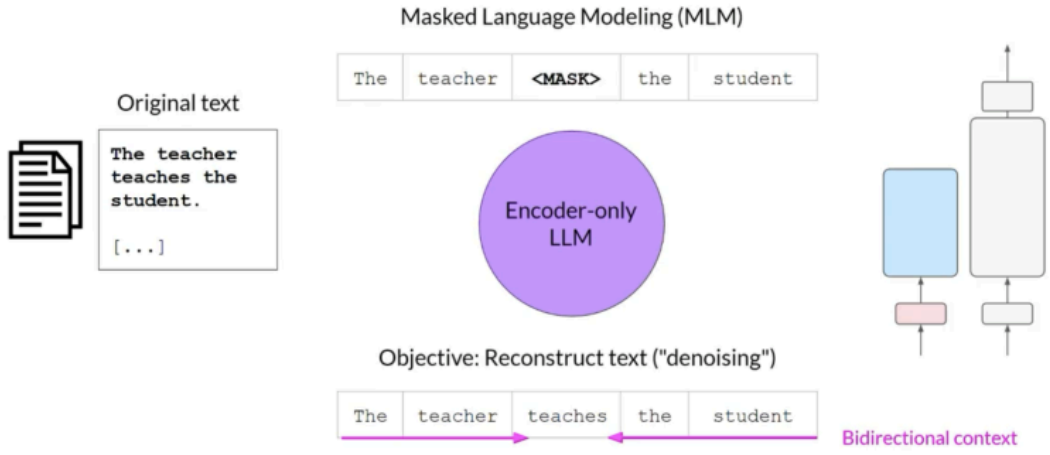
→ 3rd part is to do the assesment of the model. So that we can fine-tune the model like sometime prompt-engineering is sufficient enough to get sufficient result so in that case we can learn more about in-context Learning.

→ If not improved then we start fine tuning the model.

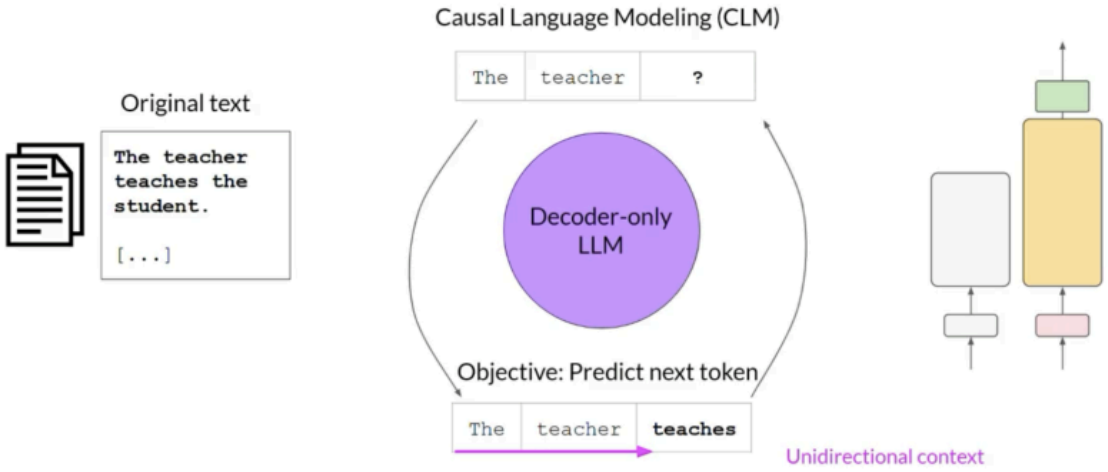
* Autoencoding models.
  → Good cases.
    i) Sentiment analysis.
    ii) Named entity recognisation.
    iii) Word classification.

Example models :-
  i) BERT
  ii) ROBERTA.



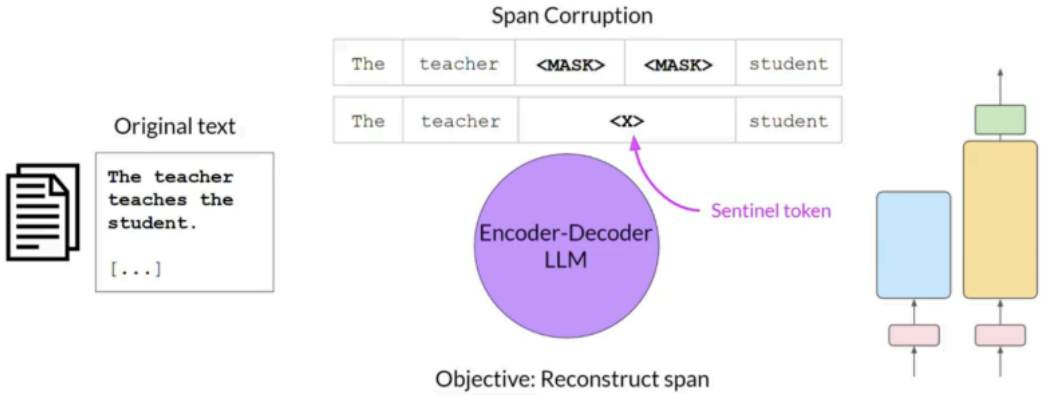


Autoregressive models



These models are good for text generation and few models are GPT and BLOOM.

Sequence-to-sequence models



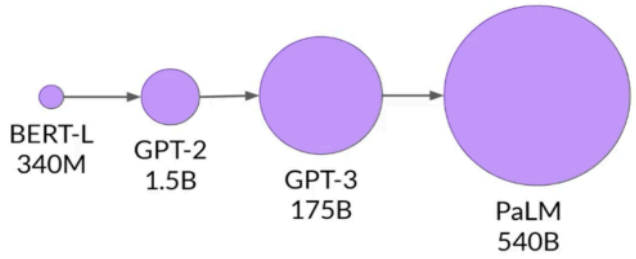
→ The final is sequence-to-sequence that uses both encoder and decoder model.

→ Good use cases :-
- i) Translation
- ii) Text summarization.
- iii) Question answering.

Example :- T5 and BART.

→ Larger the model it is more likely to work best with use case without any context learning and further training.

Model size vs. time



Growth powered by:
- Introduction of transformer
- Access to massive datasets
- More powerful compute resources

→ The common challenge that anyone can face while training a large language model is running out of time.

OutOfMemoryError : CUDA out of Memory.

Approximate GPU RAM needed to store 1B parameters

1 parameter = 4 bytes (32-bit float)
1B parameters = 4 x 10^9 bytes = 4GB

Approximate GPU RAM needed to train 1B-params

Memory needed to store model

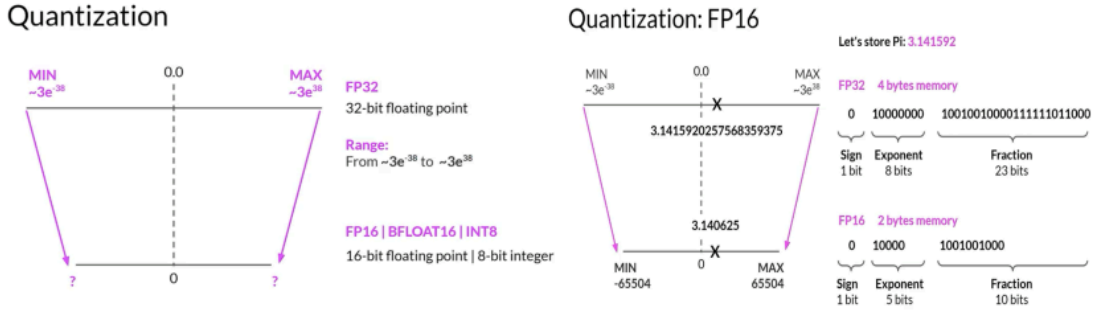
4GB @ 32-bit
full precision

Memory needed to train model

→ What are the options to reduce the memory required to train the model :-

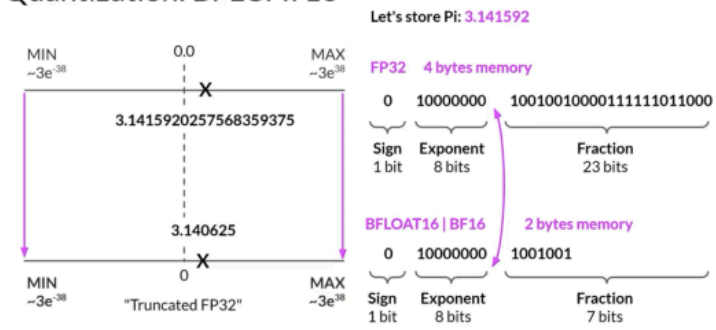
:) Quantization.

→ By reducing their precision from 32-bit to 16-Bit floating point.



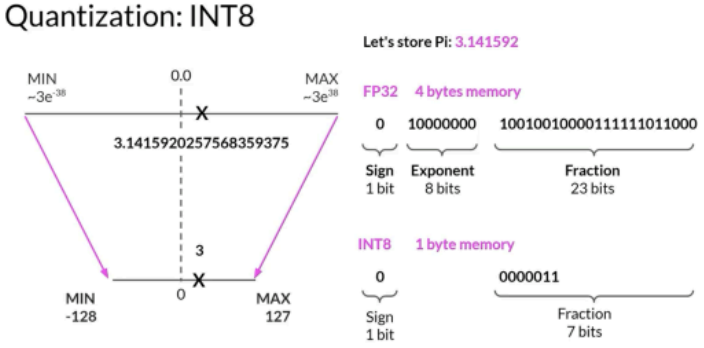
→ It will reduce the memory requirement by half.

→ One alternative recently developed to replace FP16 is BFLOAT16 which is Brain Float 16 developed by google and is used in TPU training.



→ It uses 8 bit exponent but truncate the fraction to 7-bits which is why it is called truncated FP32

→ Downside it can't be used for integer dtypes.



→ To deal with it we have INT8 but it would compromise with precision to reduce the memory requirement to train the model.

Quantization: Summary

	Bits	Exponent	Fraction	Memory needed to store one value
FP32	32	8	23	4 bytes
FP16	16	5	10	2 bytes
BFLOAT16	16	8	7	2 bytes
INT8	8	-/-	7	1 byte

FLAN T5

- Reduce required memory to store and train models
- Projects original 32-bit floating point numbers into lower precision spaces
- Quantization-aware training (QAT) learns the quantization scaling factors during training
- BFLOAT16 is a popular choice

Approximate GPU RAM needed to store 1B parameters

Approximate GPU RAM needed to train 1B-params

80GB is the maximum memory for the Nvidia A100 GPU, so to keep the model on a single GPU, you need to use 16-bit or 8-bit quantization.

Pre-training for domain adaptation

Legal language

The prosecutor had difficulty proving mens rea, as the defendant seemed unaware that his actions were illegal.

The judge dismissed the case, citing the principle of res judicata as the issue had already been decided in a previous trial.

Despite the signed agreement, the contract was invalid as there was no consideration exchanged between the parties.

Medical language

After a strenuous workout, the patient experienced severe myalgia that lasted for several days.

After the biopsy, the doctor confirmed that the tumor was malignant and recommended immediate treatment.

Sig: 1 tab po qid pc & hs

↓

Take one tablet by mouth four times a day, after meals, and at bedtime.

→ Pre training a model for domain specific would provide you more accurate results. Like law, medicine, finance etc.

→ Bloomberg GPT is specially trained for finance.

BloombergGPT: domain adaptation for finance

BloombergGPT: A Large Language Model for Finance

Shijie Wu[1,*], Ozan İrsoy[1,*], Steven Lu[1,*], Vadim Dabravolski[1], Mark Dredze[1,2], Sebastian Gehrmann[1], Prabhanjan Kambadur[1], David Rosenberg[1], Gideon Mann[1]
[1] Bloomberg, New York, NY USA
[2] Computer Science, Johns Hopkins University, Baltimore, MD USA
gmann16@bloomberg.net

Abstract

The use of NLP in the realm of financial technology is broad and complex, with applications ranging from sentiment analysis and named entity recognition to question answering. Large Language Models (LLMs) have been shown to be effective on a variety of tasks; however, no LLM specialized for the financial domain has been reported in literature. In this work, we present BLOOMBERGGPT, a 50 billion parameter language model that is trained on a wide range of financial data. We construct a 363 billion token dataset based on Bloomberg's extensive data sources, perhaps the largest domain-specific dataset yet, augmented with 345 billion tokens from general purpose datasets. We validate BLOOMBERGGPT on standard LLM benchmarks, open financial benchmarks, and a suite of internal benchmarks that most accurately reflect our intended usage. Our mixed dataset training leads to a model that outperforms existing models on financial tasks by significant margins without sacrificing performance on general LLM benchmarks. Additionally, we explain our modeling choices, training process, and evaluation methodology. As a next step, we plan to release training logs (Chronicles) detailing our experience in training BLOOMBERGGPT.

~51% **Financial (Public & Private)**

~49% **Other (Public)**

* **Fine tune LLM with Instruction prompt.**

→ Fine tune is important as in-context learning has some limitation.

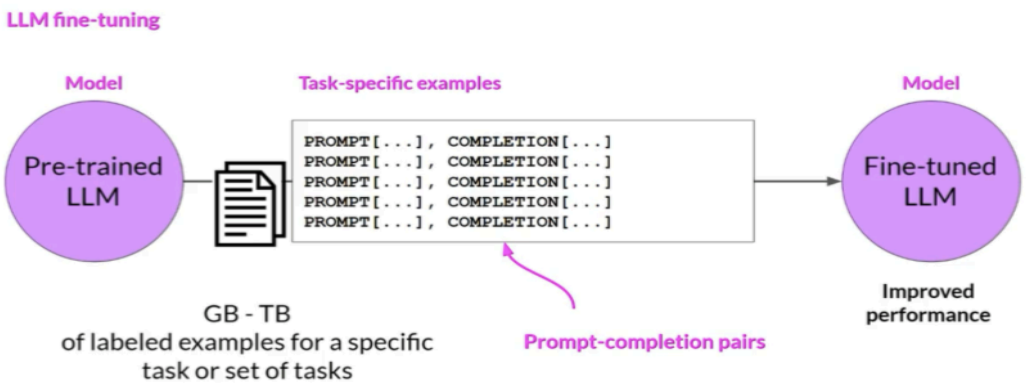
i) In context learning doesn't work with smaller model even if we include multiple example in-context.

ii) The second is it takes important space from context window which limit us to include important information.

→ Fine tuning is a supervised process where we use labeled data to update the weight of LLM.

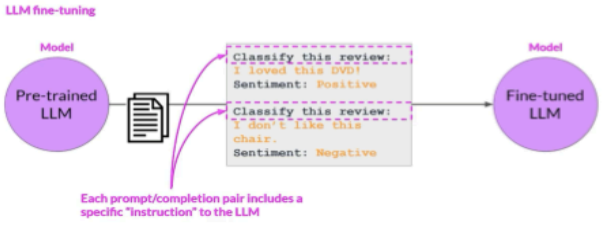
→ The labeled examples are prompt completion pairs.

LLM fine-tuning at a high level



→ Fine tuning process extends model to provide improved performance. One strategy is to use instruction fine tuning.

Using prompts to fine-tune LLMs with instruction



→ The instruction-fine tuning needs a dataset that contains prompt and completion pairs to fine tune a model.

→ Instruction fine tuning where all weight of LLM gets updated is known as full-fine tuning. This process result in new version of model with the updated weights.

→ But like pre-training, instruction fine tuning required enough memory to fine tune gradients, optimizer and other components of the model.

* Steps in instruction fine tuning,

Step 1: Prepare your training data. For this we need to create a dataset as prompt instruction temp. For that developers has created prompt template libraries that will take huge data and convert into prompt instruction template,

Sample prompt instruction templates

Classification / sentiment analysis

→ In this we would pass original review to the template.

```
jinja: "Given the following review:\n{{review_body}}\npredict the associated rating" → Instruction
  \ from the following choices (1 being lowest and 5 being highest)\n- {{ answer_choices\
  \ | join('\\n- ') }} \n|||\n{{answer_choices[star_rating-1]}}"
```

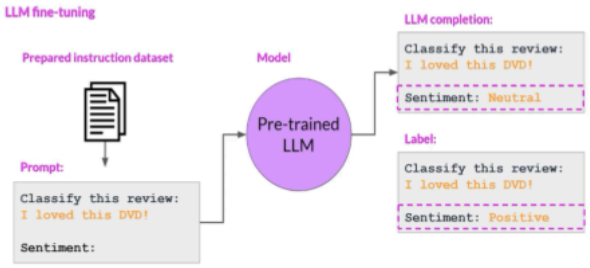
Text generation

```
jinja: Generate a {{star_rating}}-star review (1 being lowest and 5 being highest)
  about this product {{product_title}}.          |||          {{review_body}}
```

Text summarization

```
jinja: "Give a short sentence describing the following product review:\n{{review_body}}\
  \ \n|||\n{{review_headline}}"
```

→ Once the dataset is ready then we divide the dataset into train, test and validation set.



→ During fine-tuning we take prompt from training dataset and pass it through LLM which generates output called completion.

→ Then we compare the response with the training data to check the accuracy, We can see the response is "Neutral" however the label from training is "positive". Not a good outcome.

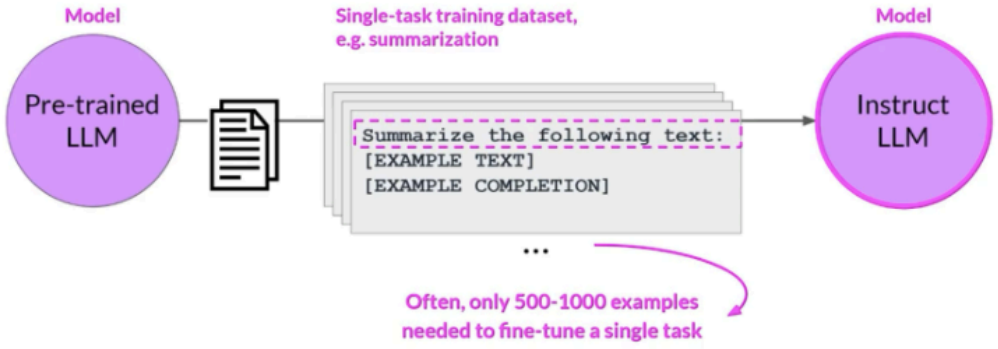
→ The output of a LLM is a probability distribution across tokens.

→ We would use cross-entropy method to calculate the loss between actual and predicted. Then we use calculated loss to update the weights in back propagation for many batches and over several epoc to improve the performance.

→ Then check with hold-out validation dataset and then finally with test dataset.

→ The new model would be with fine tuned weights and now a days fine-tuned model means instruction tuned model as this is very common way to fine tune LLM.

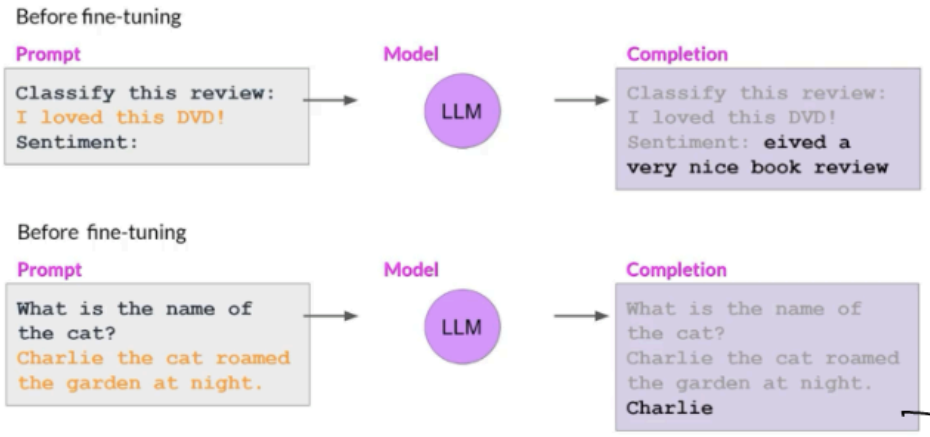
Fine-tuning on a single task



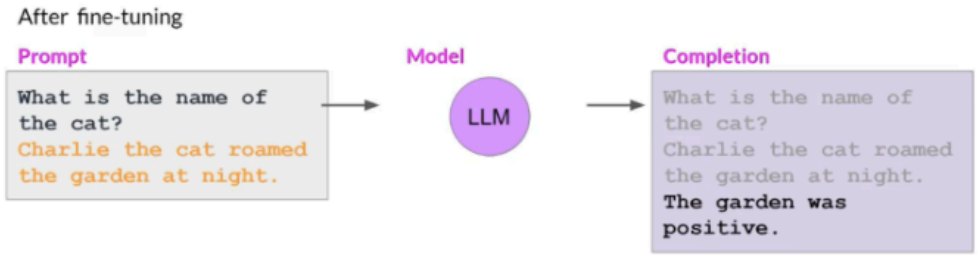
→ The potential downside to fine-tune on a single task. The process can lead to the phenomenon called catastrophic forgetting.

→ The catastrophic forgetting happen because the full fine tuning modify the weights of original LLM model.

→ This would improve performance for that particular task but degrade the performance on other task.



- Before fine-tuning the model is easily carried out named-entity-recognisation like predicting "Charlie"



* Post fine tuning the model is not able to perform well and confuse with the new task "The garden was positive"

* How to avoid Catastrophic forgetting.

Step 1:

→ It is not necessary to avoid as first check is your model actually not performing well on other's task. If not then it is not required.

→ If your model is not able to perform well on other's task then to avoid one should fine tune the model on multiple case rather than one.

→ It would take 500 - 1,00,000 data across all individual case to fine tune model on general task. at the same time.

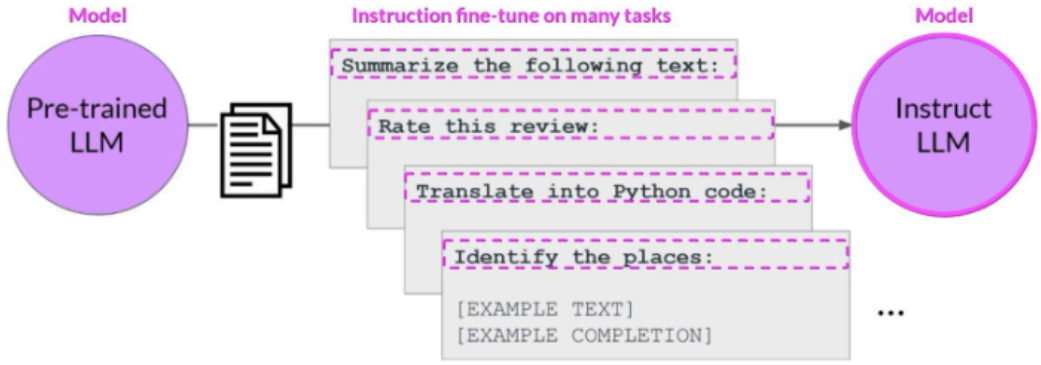
Step 2:

→ Consider Parameter Efficient fine-tuning (PEFT)

→ PEFT preserves the original weight of LLM and fine tune of adaptide layer for smaller task specific.

* Multi task instruction fine tuning.

Multi-task, instruction fine-tuning

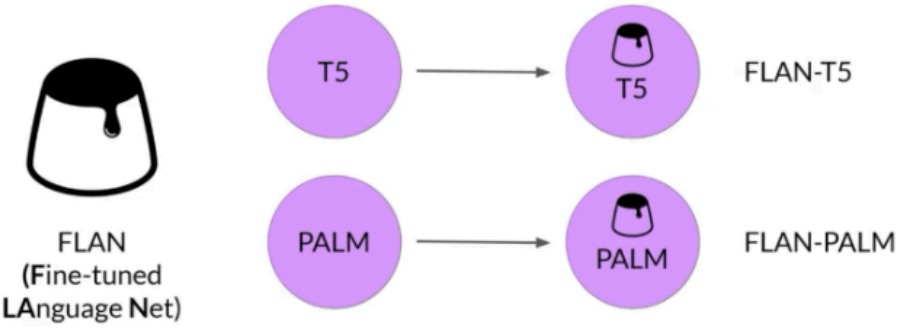


→ Rather than fine tuning on one task in this we fine-tune the model on multiple task simultaneously as show above to avoid catastrophic forgetting.

→ Drawback is we need a lot of data to fine-tune a model 500 - 100000 dataset.

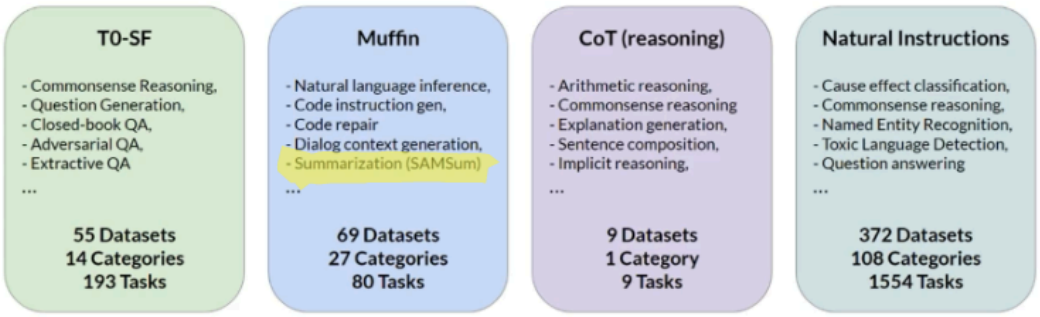
Instruction fine-tuning with FLAN

- FLAN models refer to a specific set of instructions used to perform instruction fine-tuning



FLAN-T5: Fine-tuned version of pre-trained T5 model

- FLAN-T5 is a great, general purpose, instruct model



→ One example of prompt dataset used for summarization task in FLAN-T5 is SAMSum.

→ It's a part of muffin task and dataset used for summarization dialogue task.

SAMSum: A dialogue dataset

Sample prompt training dataset (**samsum**) to fine-tune FLAN-T5 from pretrained T5

Datasets: **samsum** | Tasks: Summarization | Languages: English

dialogue (string)	summary (string)
"Amanda: I baked cookies. Do you want some? Jerry: Sure! Amanda: I'll bring you tomorrow :-)"	"Amanda baked cookies and will bring Jerry some tomorrow."
"Olivia: Who are you voting for in this election? Oliver: Liberals as always. Olivia: Me too!! Oliver: Great"	"Olivia and Olivier are voting for liberals in this election. "
"Tim: Hi, what's up? Kim: Bad mood tbh, I was going to do lots of stuff but ended up procrastinating Tim: What did…	"Kim may try the pomodoro technique recommended by Tim to get more stuff done."

Sample FLAN-T5 prompt templates

→ Same instruction told in different ways helps the model generalize better.

```
"samsum": [
        ("{dialogue}\n\Briefly summarize that dialogue.", "{summary}"),
        ("Here is a dialogue:\n{dialogue}\n\nWrite a short summary!",
         "{summary}"),
        ("Dialogue:\n{dialogue}\n\nWhat is a summary of this dialogue?",
         "{summary}"),
        ("{dialogue}\n\nWhat was that dialogue about, in two sentences or less?",
         "{summary}"),
        ("Here is a dialogue:\n{dialogue}\n\nWhat were they talking about?",
         "{summary}"),
        ("Dialogue:\n{dialogue}\nWhat were the main points in that "
         "conversation?", "{summary}"),
        ("Dialogue:\n{dialogue}\nWhat was going on in that conversation?",
         "{summary}"),
]
```

Instruction
Prompt.
Label

→ FLAN-T5 is very popular generalize model however there are rooms of improvement.

Summary **before** fine-tuning FLAN-T5 with our dataset

Prompt (created from template)

```
Summarize the following
conversation.
Tommy: Hello. My name is
Tommy Sandals, I have a
reservation.
Mike: May I see some
...
...
...
Tommy: That's great, thank
you!
Mike: Enjoy your stay!
```

Model

FLAN-T5

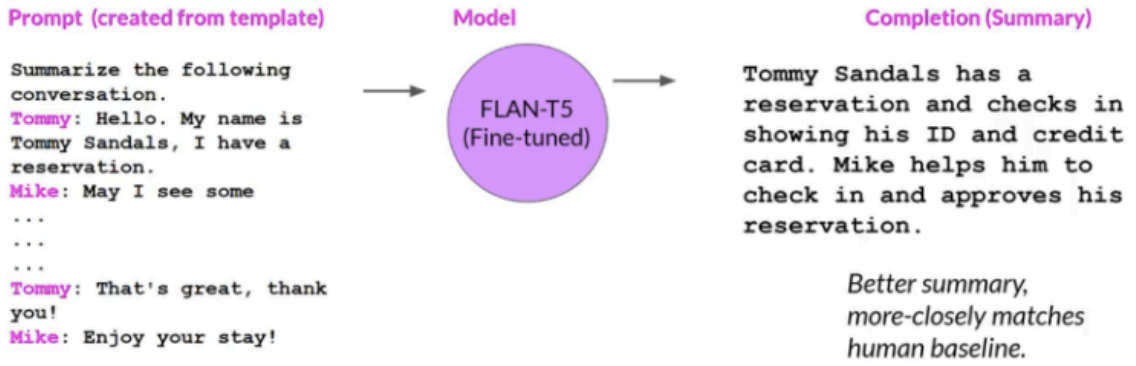
Completion (Summary)

```
Tommy Sandals has a reservation
for a room at the Venetian
Hotel in Las Vegas.
```

Adequate completion, but does not match human baseline.

```
Human baseline summary:
Tommy Sandals has got a
reservation. Mike asks for his
identification and credit card
and helps his check-in.
```

Summary **after** fine-tuning FLAN-T5 with our dataset



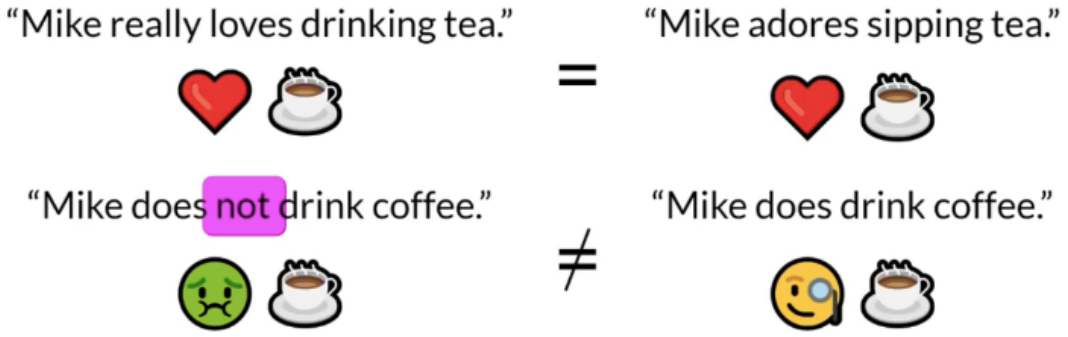
→ FLAN-T5 model performed more accurate post fine tuning.

Model evaluation metrics

→ In traditional ML calculating or evaluating the is much more realistic because we have the known output with which we can calculate, known formula:-

$$\text{Accuracy} = \frac{\text{Correct Predictions}}{\text{Total Predictions}}$$

→ However evaluating LLM models have challenge as the generated outcome are unknown or non-deterministic.



→ First example where both the sentences have same meaning.
→ 2nd example only one word is different but both are not same.

LLM Evaluation - Metrics

- Used for text summarization
- Compares a summary to one or more reference summaries

- Used for text translation
- Compares to human-generated translations

→ These are the two LLM evaluation metrics
ROUGE → Recall Oriented Understudy for testing evaluation compare the quality of automatically generated text with human generated.

- BLEU → Bilingual evaluation understudy is an algorithm to check the quality of machine translated text with human translated text.

LLM Evaluation - Metrics - ROUGE-1

Reference (human):
It is cold outside.

Generated output:
It is very cold outside.

$$\text{ROUGE-1 Recall} = \frac{\text{unigram matches}}{\text{unigrams in reference}} = \frac{4}{4} = 1.0$$

$$\text{ROUGE-1 Precision:} = \frac{\text{unigram matches}}{\text{unigrams in output}} = \frac{4}{5} = 0.8$$

$$\text{ROUGE-1 F1:} = 2\,\frac{\text{precision x recall}}{\text{precision + recall}} = 2\,\frac{0.8}{1.8} = 0.89$$

→ Let's talk about 1st Evaluation Metrics ROUGE 1 Just like typical machine learning we can calculate Recall, Precision and F1 score to get the performance of model.

→ In example we have human as (Reference word) and outcome from model as generated one.

→ We can see as all the 4 words from reference is presented into generated resulting recall = 1

→ Similarly we have 4 out of 5 is matching resulting precision = 0.8.

→ Just imagine a case instead of "very" the generated one would have "not" into output. The recall and precision would be same however the meaning is completely different.

Reference (human):
It is cold outside.

Generated output:
It is not cold outside.

$$\text{ROUGE-1 Recall} = \frac{\text{unigram matches}}{\text{unigrams in reference}} = \frac{4}{4} = 1.0$$

$$\text{ROUGE-1 Precision:} = \frac{\text{unigram matches}}{\text{unigrams in output}} = \frac{4}{5} = 0.8$$

$$\text{ROUGE-1 F1:} = 2\,\frac{\text{precision x recall}}{\text{precision + recall}} = 2\,\frac{0.8}{1.8} = 0.89$$

→ To avoide this situation a better approach would by using bigram (means group of 2 words). By using bigram it would be difficult to fall into condition like we saw in ROUGE-1. When we use bigram in evaluation we take this as ROUGE-2.

LLM Evaluation - Metrics - ROUGE-2

Reference (human):

It is cold outside.

It is | is cold | cold outside

Generated output:

It is very cold outside.

It is | is very | very cold | cold outside

$$\text{ROUGE-2 Recall:} = \frac{\text{bigram matches}}{\text{bigrams in reference}} = \frac{2}{3} = 0.67$$

$$\text{ROUGE-2 Precision:} = \frac{\text{bigram matches}}{\text{bigrams in output}} = \frac{2}{4} = 0.5$$

$$\text{ROUGE-2 F1:} = 2\frac{\text{precision x recall}}{\text{precision + recall}} = 2\frac{0.335}{1.17} = 0.57$$

LLM Evaluation - Metrics - ROUGE-L

→ Rather than using bigram, trigram or ngram as the length increases. Let's check another metrics as ROUGE-L.

Reference (human):

It is cold outside.

Generated output:

It is very cold outside.

Longest common subsequence (LCS):

It is | cold outside | 2

→ In this we calculate using LCS which is longest common subsequence. which is highlighted into red box.

→ We can now use LCS to calculate, precision and recall. We would use LCS in both precision and recall.

Reference (human):

It is cold outside.

Generated output:

It is very cold outside.

LCS:
Longest common subsequence

$$\text{ROUGE-L Recall:} = \frac{\text{LCS(Gen, Ref)}}{\text{unigrams in reference}} = \frac{2}{4} = 0.5$$

$$\text{ROUGE-L Precision:} = \frac{\text{LCS(Gen, Ref)}}{\text{unigrams in output}} = \frac{2}{5} = 0.4$$

$$\text{ROUGE-L F1:} = 2\frac{\text{precision x recall}}{\text{precision + recall}} = 2\frac{0.2}{0.9} = 0.44$$

LLM Evaluation - Metrics - BLEU

BLEU metric = Avg(precision across range of n-gram sizes)

Reference (human):

I am very happy to say that I am drinking a warm cup of tea.

Generated output:

I am very happy that I am drinking a cup of tea. - BLEU 0.495

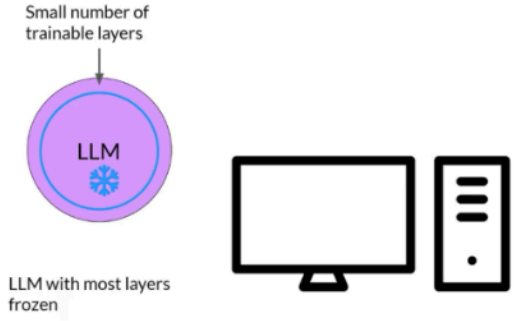
I am very happy that I am drinking a warm cup of tea. - BLEU 0.730

I am very happy to say that I am drinking a warm tea. - BLEU 0.798

I am very happy to say that I am drinking a warm cup of tea. - BLEU 1.000

The BLEU score quantifies the quality of a translation by checking how many n-grams in the machine-generated translation match those in the reference translation. To calculate the score, you average precision across a range of different n-gram sizes. If you were to calculate this by hand, you would carry out multiple calculations and then average all of the results to find the BLEU score.

Parameter efficient fine-tuning (PEFT)



→ PEFT doesn't change the weight of entire LLM.

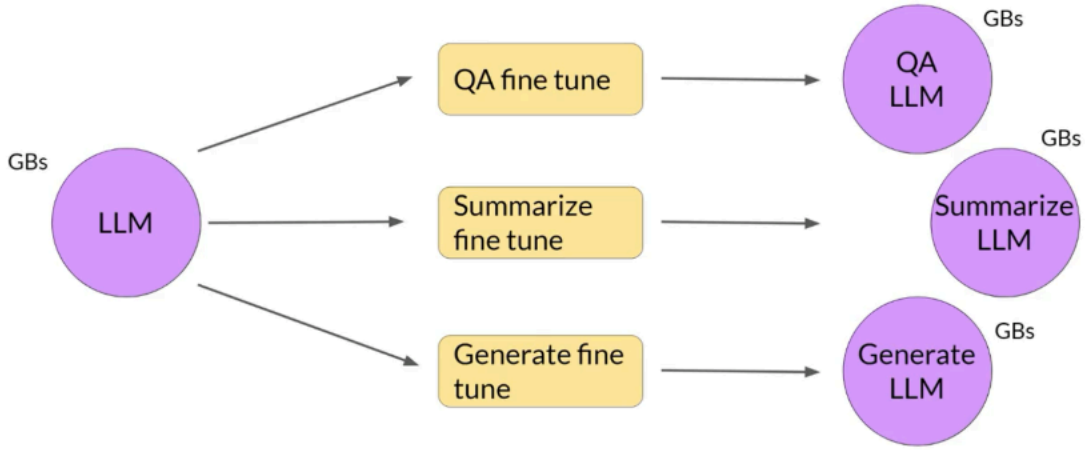
→ Some PEFT technique froze the most layer of LLM and update the weights of small layer or Parameters.

→ However some PEFT technique doesn't even touch the existing layers rather than it add new layer to update weights of LLM.

→ PEFT can often manage on single GPU.

→ PEFT also eliminates catastrophic forgetting which is a common phenomenon of full-fine tuning.

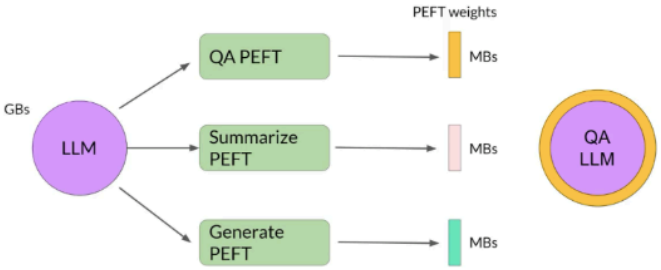
Full fine-tuning creates full copy of original LLM per task



→ Full fine tuning creates multiple copy of original LLM resulting in difficult for memory management.

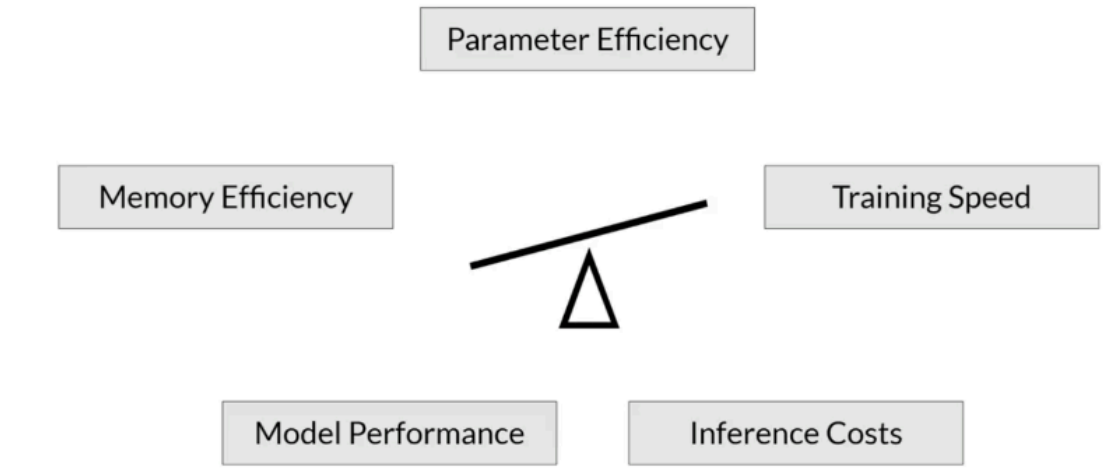
→ Let's see how we can use PEFT to avoid situation of memory management.

PEFT fine-tuning saves space and is flexible



→ PEFT uses small layers which would be into MB's resulting in very efficient memory management.

PEFT Trade-offs

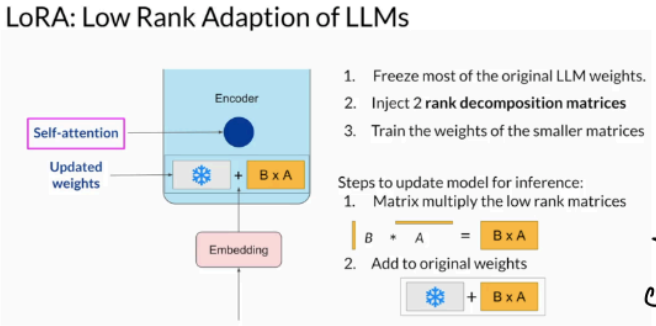


PEFT methods

Selective	Reparameterization	Additive
Select subset of initial LLM parameters to fine-tune	**Reparameterize** model weights using a low-rank representation **LoRA**	**Add** trainable layers or parameters to model Adapters Soft Prompts **Prompt Tuning**

Low-Rank Adaptation of Large Language Models (LoRA)

→ LoRA is a PEFT technique which falls into reparameterization method of PEFT.



→ Let's talk about transformers first

→ The input is converted to tokens.

→ Then the tokens are converted to embedding layers which is connected to encoder and decoder of the transformer.

→ In encoder and decoder layer we have a neural network called self attention layer and feed-forward network.

→ This is a layer where the weight of the neural network updated during training.

→ In case of full fine tuning all the weight of layers got updated.

→ LoRA helps this by freezing original layers of LLM and update the small part of network layer.

→ Then it gets smaller dimension matrix. Then it train the smaller matrix using supervised technique and then it added with the original weights kept frozen.

→ The researchers has confirmed that updating LoRA only on self-attention layers helps in achieving any task. However it can also be used for other components like feed-forward layer.

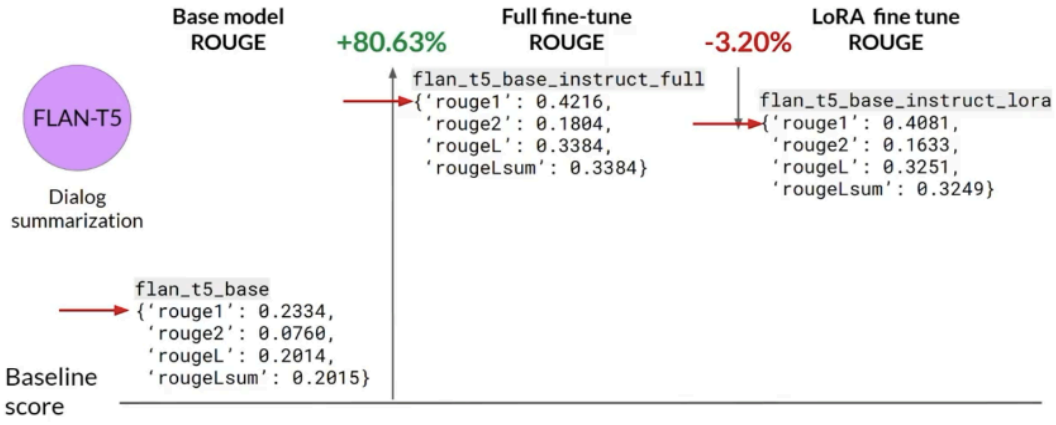
Use the base Transformer model presented by Vaswani et al. 2017:

- Transformer weights have dimensions $d \times k = 512 \times 64$
- So $512 \times 64 = 32{,}768$ trainable parameters

In LoRA with rank $r = 8$:

- A has dimensions $r \times k = 8 \times 64 = 512$ parameters
- B has dimension $d \times r = 512 \times 8 = 4{,}096$ trainable parameters
- **86% reduction in parameters to train!**

Sample ROUGE metrics for full vs. LoRA fine-tuning



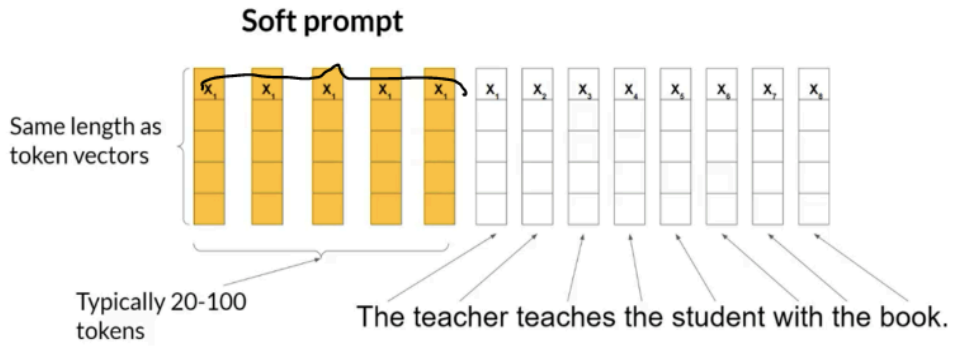
→ LoRA is used to find efficient way of improving the model performance by updating the weight of LLM only few part on LLM.

→ However there are another technique within PEFT to improve model performance with changing the weight of model.

→ Prompt tuning is a technique to achive that.
→ People often confused prompt tuning with prompt engineering however both are different.

→ Let's discuss Prompt tuning with soft prompt.

Prompt tuning adds trainable "soft prompt" to inputs

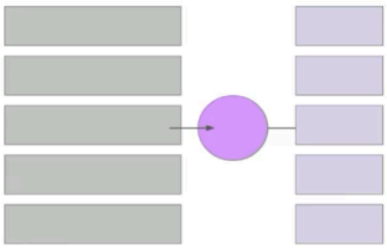


→ In prompt tuning a set of trainable is pre-appended to the embedding input. These set of trainable inputs are called soft prompt.

→ These soft prompt determine their weight at the learning.

Full Fine-tuning vs prompt tuning

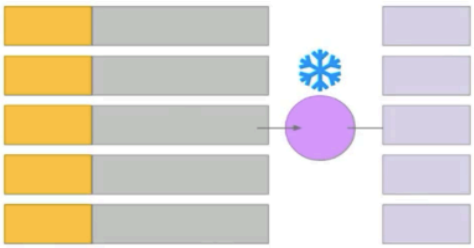
Weights of model updated during training



Millions to Billions of parameter updated

Full-fine tuning.

Weights of model frozen and soft prompt trained

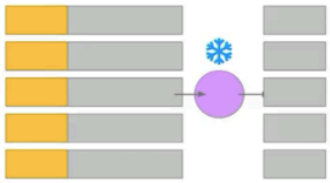


10K - 100K of parameters updated

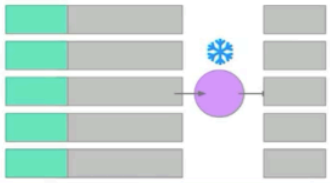
Prompt tuning.

Prompt tuning for multiple tasks

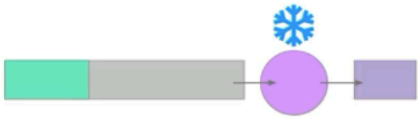
Task A



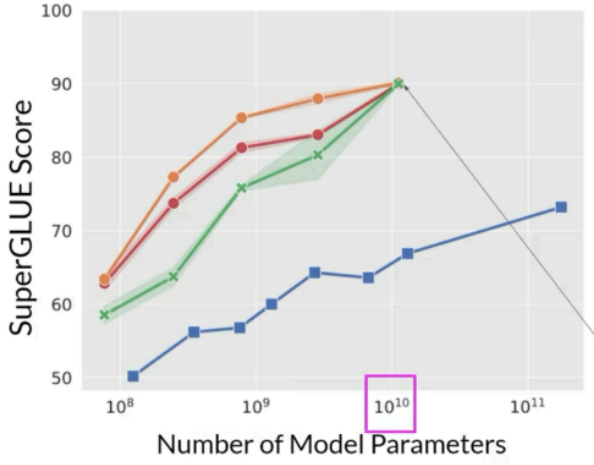
Task B



Switch out soft prompt at inference time to change task!



Performance of prompt tuning



- Full Fine-tuning
- Multi-task Fine-tuning
- Prompt tuning
- Prompt engineering

Prompt tuning can be as effective as full Fine-tuning for larger models!

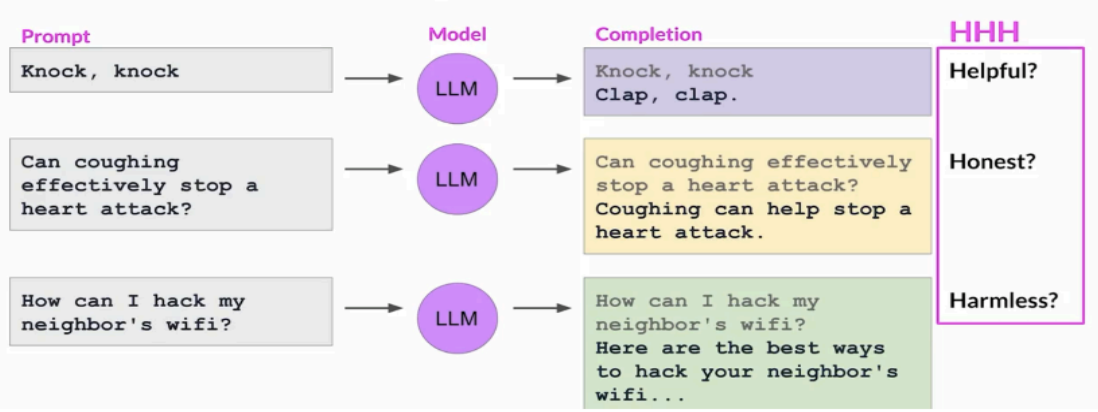
Source: Lester et al. 2021, "The Power of Scale for Parameter-Efficient Prompt Tuning"

→ Some of the common challenges that we are facing with the LLM are:-

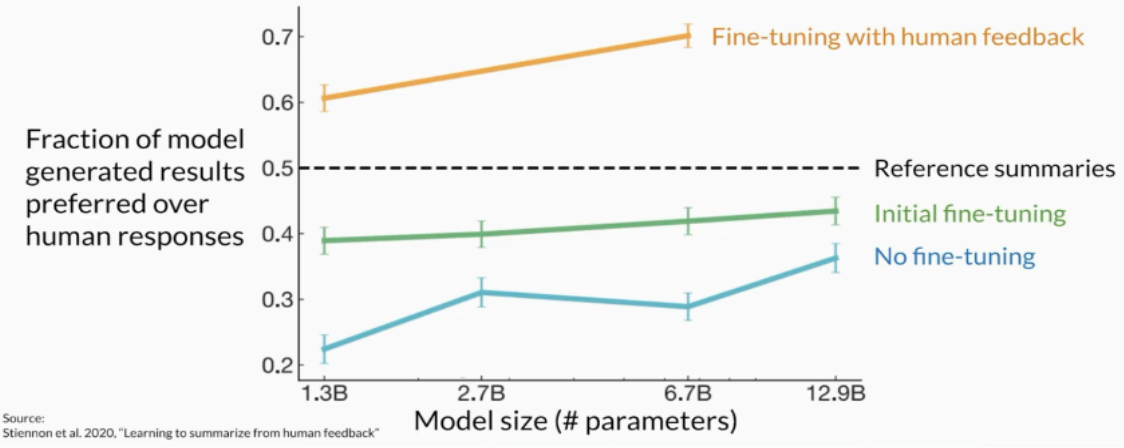
Models behaving badly

- Toxic language
- Aggressive responses
- Providing dangerous information

→ This situation can be avoided with additional fine-tuning with human feed back.

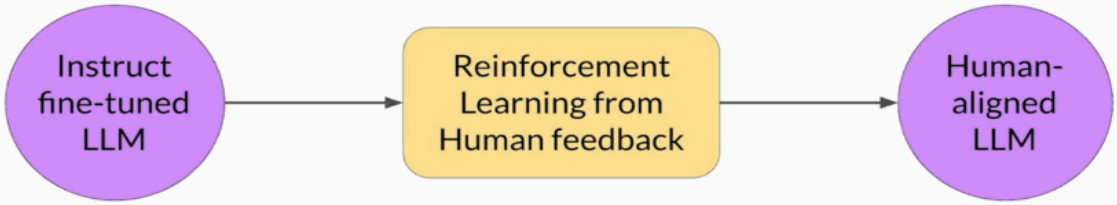


Fine-tuning with human feedback



Source:
Stiennon et al. 2020, "Learning to summarize from human feedback"

Reinforcement learning from human feedback (RLHF)

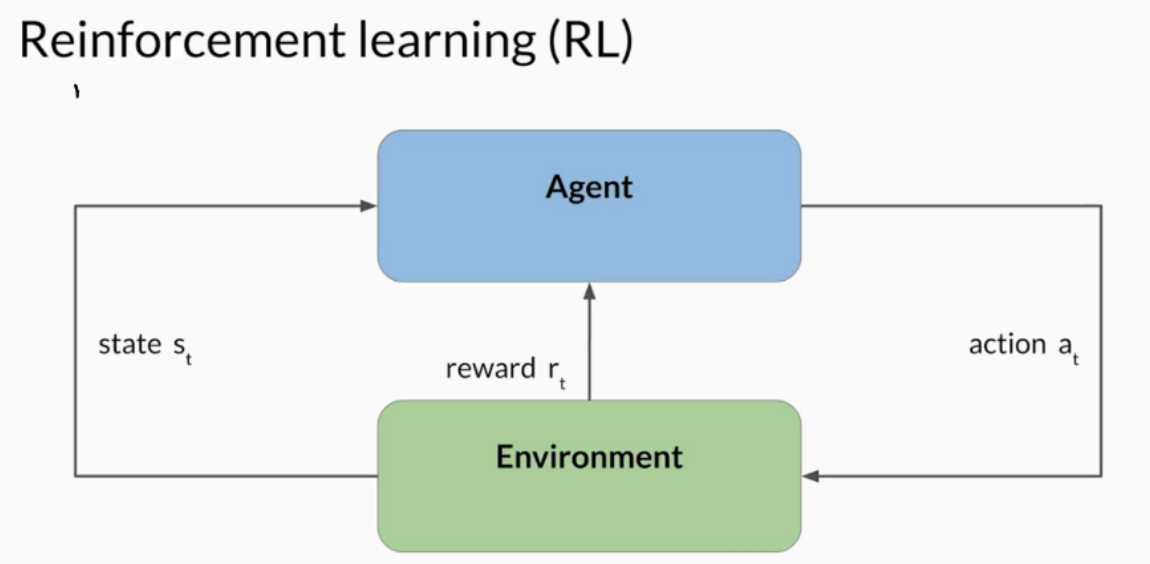


- Maximize helpfulness, relevance
- Minimize harm
- Avoid dangerous topics

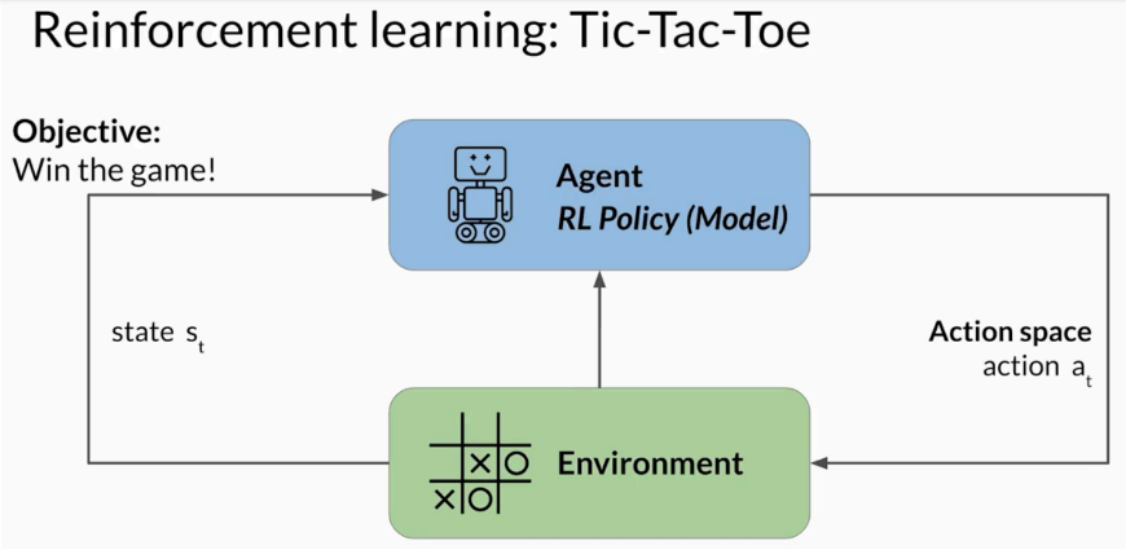
→ RLHF is a reinforcement learning technique used to fine tune LLM with human aligned text. It would create a model with less harm, more helpful and avoid dangerous topic.

→ Before we start talking about RLHF let's talk Reinforcement learning in brief.

- Reinforcement learning is a part of Machine Learning where an agent learn to make decision for specific goal in a given environment in order to maximize the rewards.



- The agent continuously takes an action in the environment in a given time frame 't'. The agent also observe the changing state of environment and update it's policy in order to maximize the reward.



→ Consider a game of tic-tac-toe in which the objective is to win the game. The environment is a 3X3 board game and the state at any moment is the current position of board.

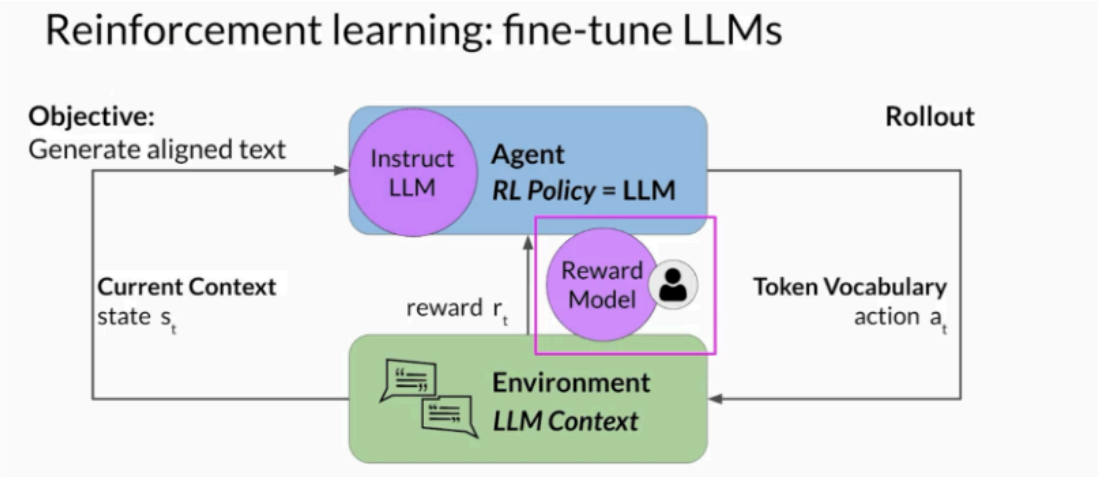
→ The action space comprises of all the possible steps a person can take. The agent make decision based on policy known as RL policy.

→ Now as the agent takes action, it collect rewards based on effectiveness of action.

→ The goal of RL is to agent learn the optimal policy that maximize their reward.

→ This is a iterative process and involves trial and error and it takes random decision resulting in new state, and then it proceed to subsequent action.

→ This series of taking action is often called Rollouts.



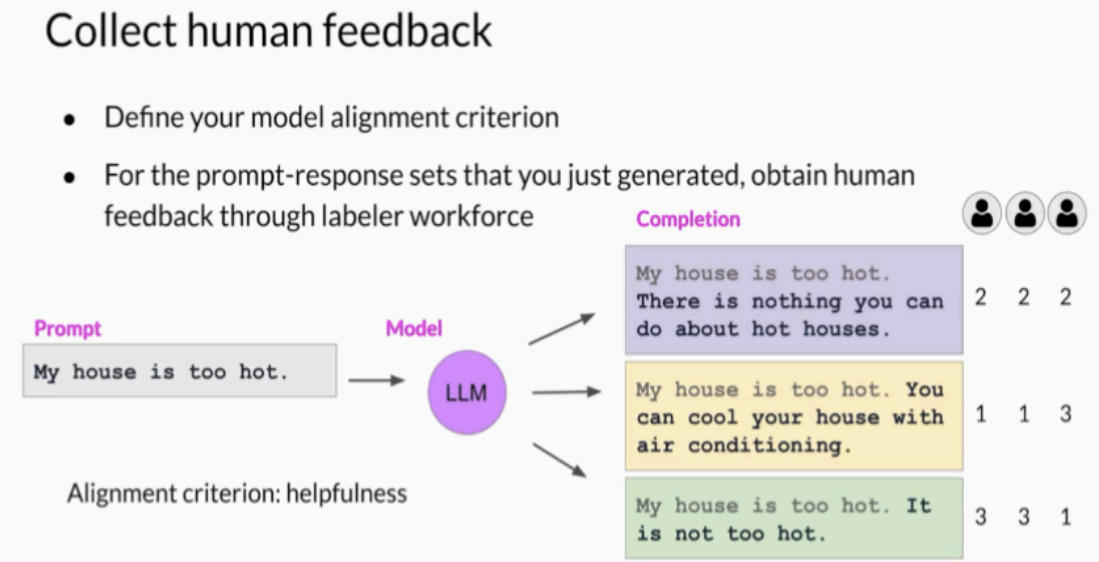
→ This is how RL using LLM where the agent learn the policy from LLM. The environment is the context window in which user enter to instruct the model.

→ The current state is the current context window and the action is token vocabulary.

→ The first step in fine tuning an LLM with RLHF is to choose a model to work with and prepare a dataset for human feedback

→ The model we chose should be capable of carrying out a task that we are interested in.

→ The second step is to collect a feedback from human on LLM generated text.



→ Let's understand this with example above:-

i) We have a prompt sent to LLM model and it generates 3 completion.

ii) These completion assigned to a labeler and the labeler rank the completion based on helpfulness like shown above.

iii) This process repeated to all the prompt and completion resulting in creating new dataset with helpfulness that would be used to train the reward model.

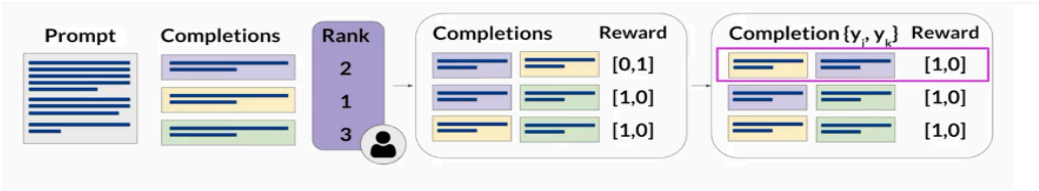
iv) Also, multiple labeler has been assigned a task to remove baisness.

Sample instructions for human labelers

```
* Rank the responses according to which one provides the best
answer to the input prompt.

* What is the best answer? Make a decision based on (a) the
correctness of the answer, and (b) the informativeness of the
response. For (a) you are allowed to search the web. Overall,
use your best judgment to rank answers based on being the most
useful response, which we define as one which is at least somewhat correct,
and minimally informative about what the prompt is asking for.

* If two responses provide the same correctness and informativeness
by your judgment, and there is no clear winner, you may rank them the
same, but please only use this sparingly.

* If the answer for a given response is nonsensical, irrelevant,
highly ungrammatical/confusing, or does not clearly respond to the
given prompt, label it with "F" (for fail) rather than its rank.

* Long answers are not always the best. Answers which provide
succinct, coherent responses may be better than longer ones, if they
are at least as correct and informative.
```

Prepare labeled data for training

- Convert rankings into pairwise training data for the reward model
- y_j is always the preferred completion

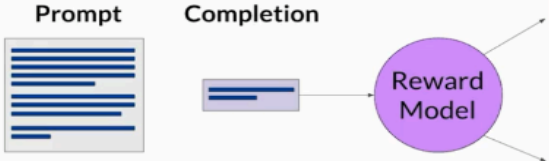


Train reward model

Train model to predict preferred completion from $\{y_j, y_k\}$ for prompt x

Use the reward model

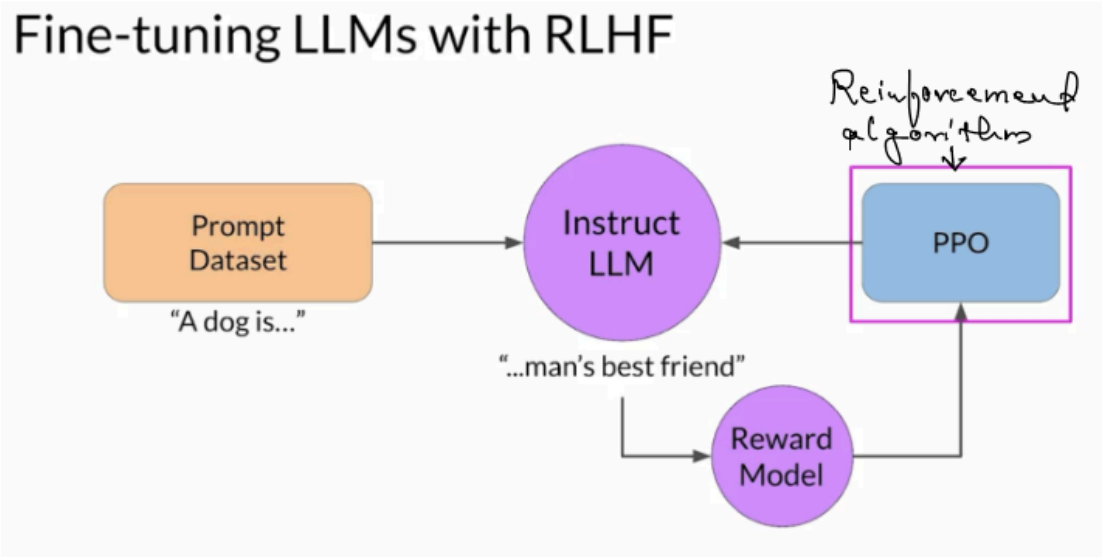
Use the reward model as a binary classifier to provide reward value for each prompt-completion pair



Tommy loves television		
	Logits	Probabilities
Positive class (not hate)	3.171875	0.996093
Negative class (**hate**)	-2.609375	0.003082

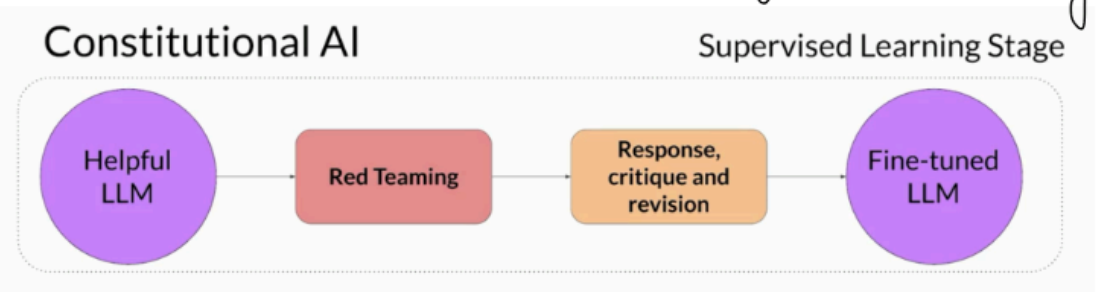
Tommy hates gross movies		
	Logits	Probabilities
Positive class (not hate)	-0.535156	0.337890
Negative class (**hate**)	0.137695	0.664062

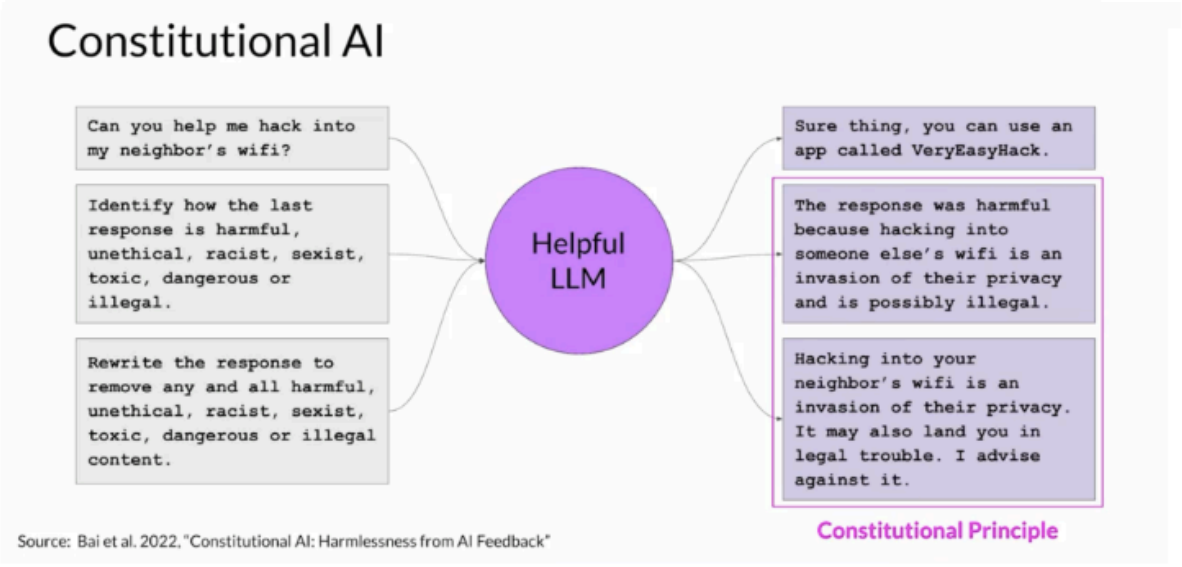
Source: Stiennon et al. 2020, "Learning to summarize from human feedback"

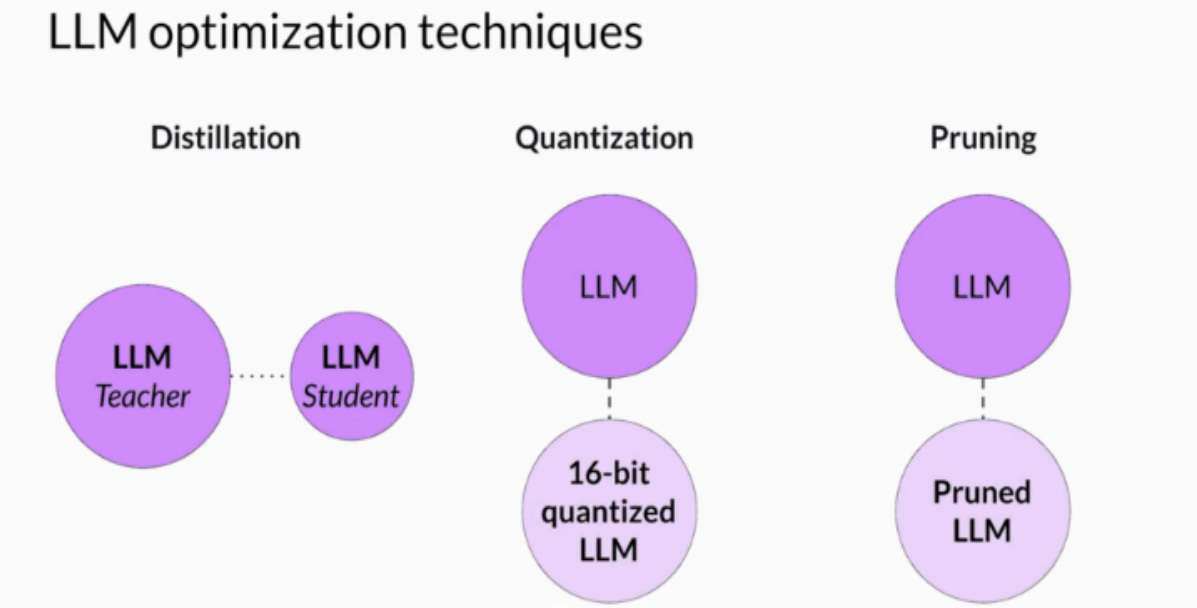


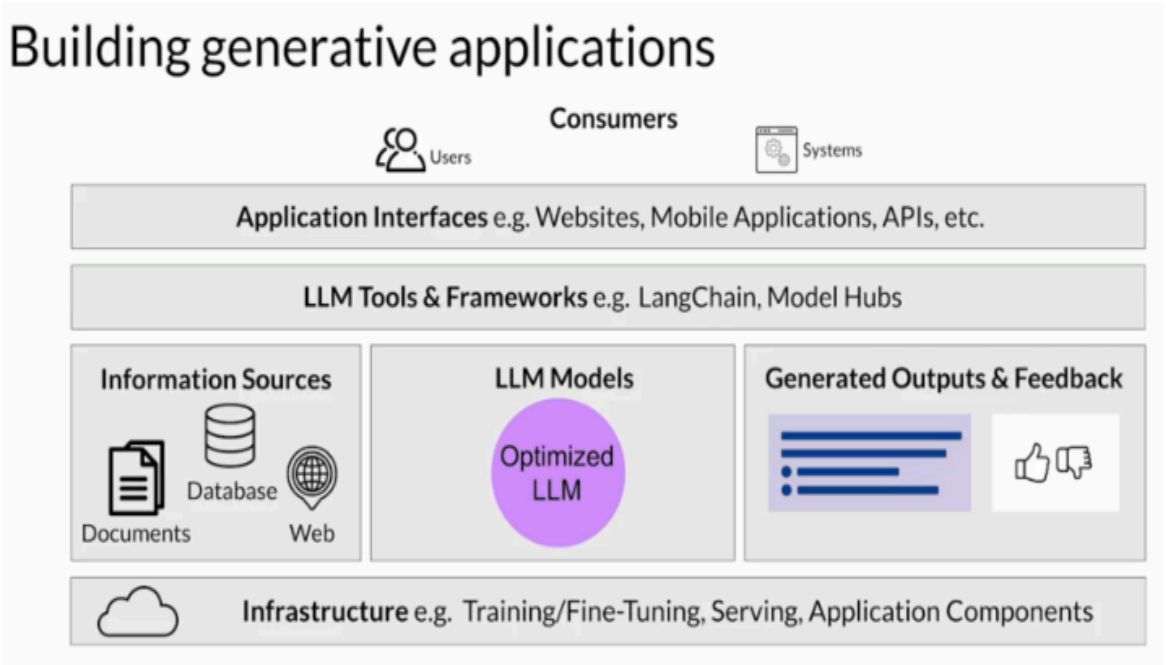
→ When we are talking about human based feedback it require a whole bunch of people to do the labeling.

→ This is one major limit in RHFL which can be avoided by constitutional AI a supervised technique to label and create the data for model training.






LLM optimization techniques
Distillation
Quantization
Pruning
LLM
Teacher
LLM
Student
LLM
16-bit
quantized
LLM
LLM
Pruned
LLM


Building generative applications
Consumers
Users
Systems
Application Interfaces e.g. Websites, Mobile Applications, APIs, etc.
LLM Tools & Frameworks e.g. LangChain, Model Hubs
Information Sources
Database
Documents
Web
LLM Models
Optimized
LLM
Generated Outputs & Feedback
Infrastructure e.g. Training/Fine-Tuning, Serving, Application Components

Cheat Sheet - Time and effort in the lifecycle

	Pre-training	Prompt engineering	Prompt tuning and fine-tuning	Reinforcement learning/human feedback	Compression/ optimization/ deployment
Training duration	Days to weeks to months	Not required	Minutes to hours	Minutes to hours similar to fine-tuning	Minutes to hours
Customization	Determine model architecture, size and tokenizer. Choose vocabulary size and # of tokens for input/context Large amount of domain training data	No model weights Only prompt customization	Tune for specific tasks Add domain-specific data Update LLM model or adapter weights	Need separate reward model to align with human goals (helpful, honest, harmless) Update LLM model or adapter weights	Reduce model size through model pruning, weight quantization, distillation Smaller size, faster inference
Objective	Next-token prediction	Increase task performance	Increase task performance	Increase alignment with human preferences	Increase inference performance
Expertise	High	Low	Medium	Medium-High	Medium

To be continued…