Copyright Notice

DeepLearning.AI

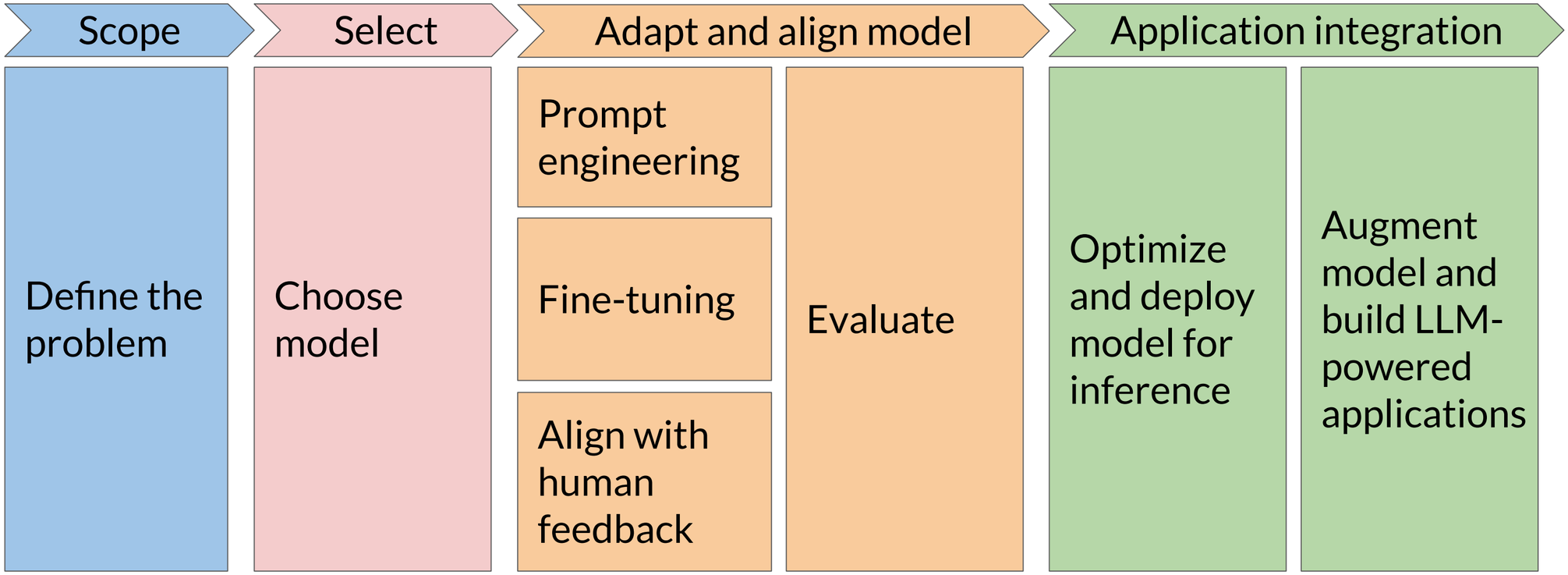
Generative AI and large-language models (LLMs)

FINE-TUNING, INSTRUCTION PROMPTS, AND PARAMETER EFFICIENT FINE-TUNING

Main Objective behind Instruction fine tuning:
so we take a look at instruction fine-tuning, so when you have your base model, the thing that's initially pretrained, it's encoded a lot of really good information, usually about the world. So it knows about things, but it doesn't necessarily know how to be able to respond to our prompts, our questions. So when we instruct it to do a certain task, it doesn't necessarily know how to respond. And so instruction fine-tuning helps it to be able to change its behavior to be more helpful for us.

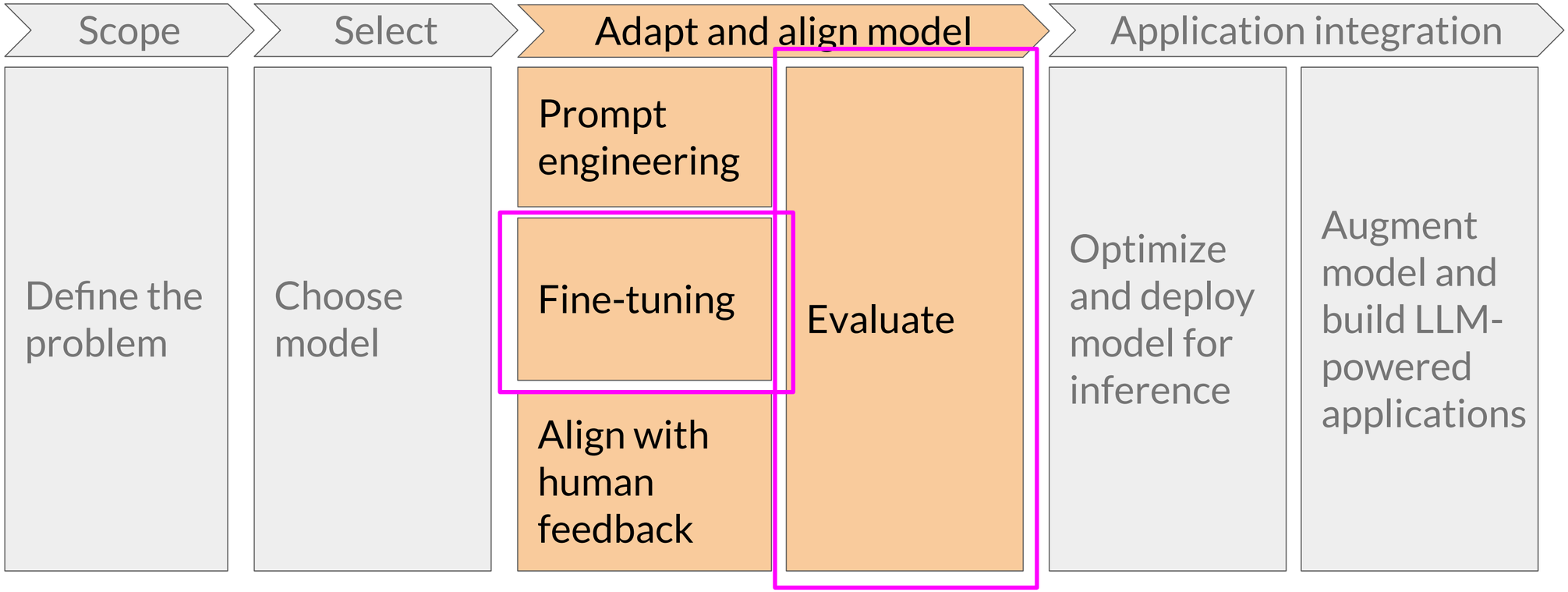
Fine-tuning with instruction prompts

GenAI project lifecycle



GenAI project lifecycle

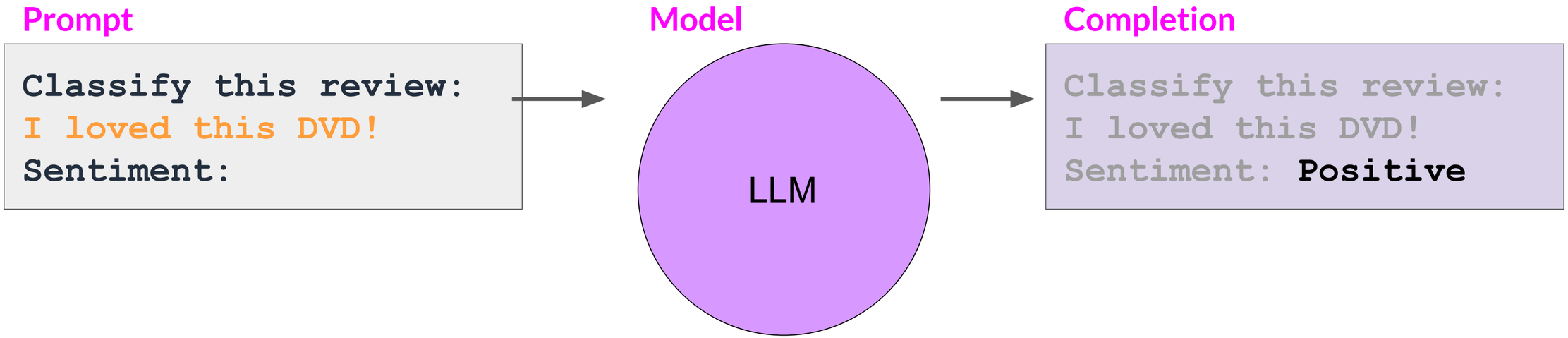
This week we will be focusing on the Fine-tuning part of the GenAI project lifecycle followed by understanding the Evaluation metrices to evaluate the Fine tuned LLM model



Fine-tuning an LLM with instruction prompts

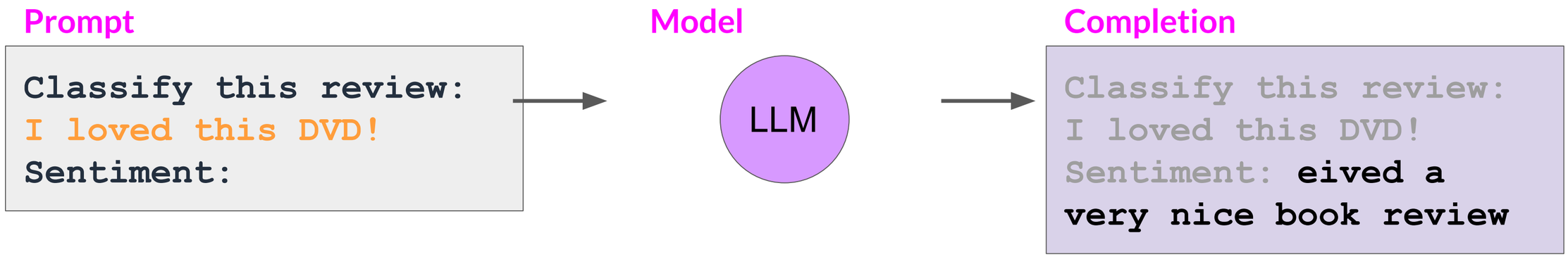
INSTRUCTION FINE TUNING

In-context learning (ICL) - zero shot inference



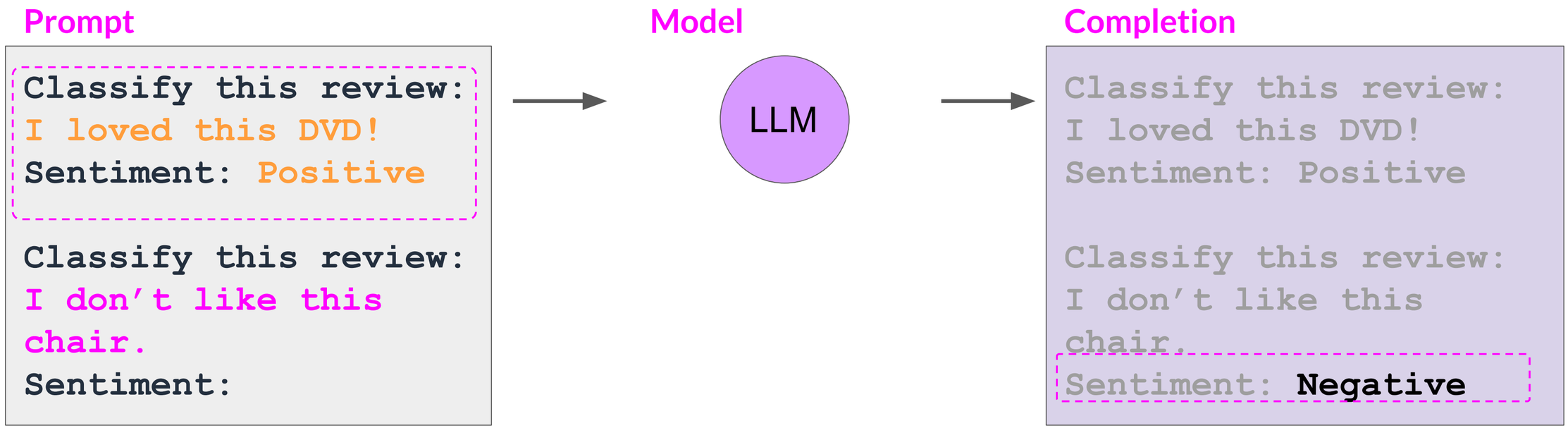
Zero shot capable to do task when model size large
Already discussed in week 1

In-context learning (ICL) - zero shot inference



Zero shot not capable to do task when model size small
Already discussed in week 1

In-context learning (ICL) - one/few shot inference



One-shot or Few-shot Inference

One/Few shot capable of doing task with small model size with the inclusion of examples in the Prompt
Already discussed in week 1

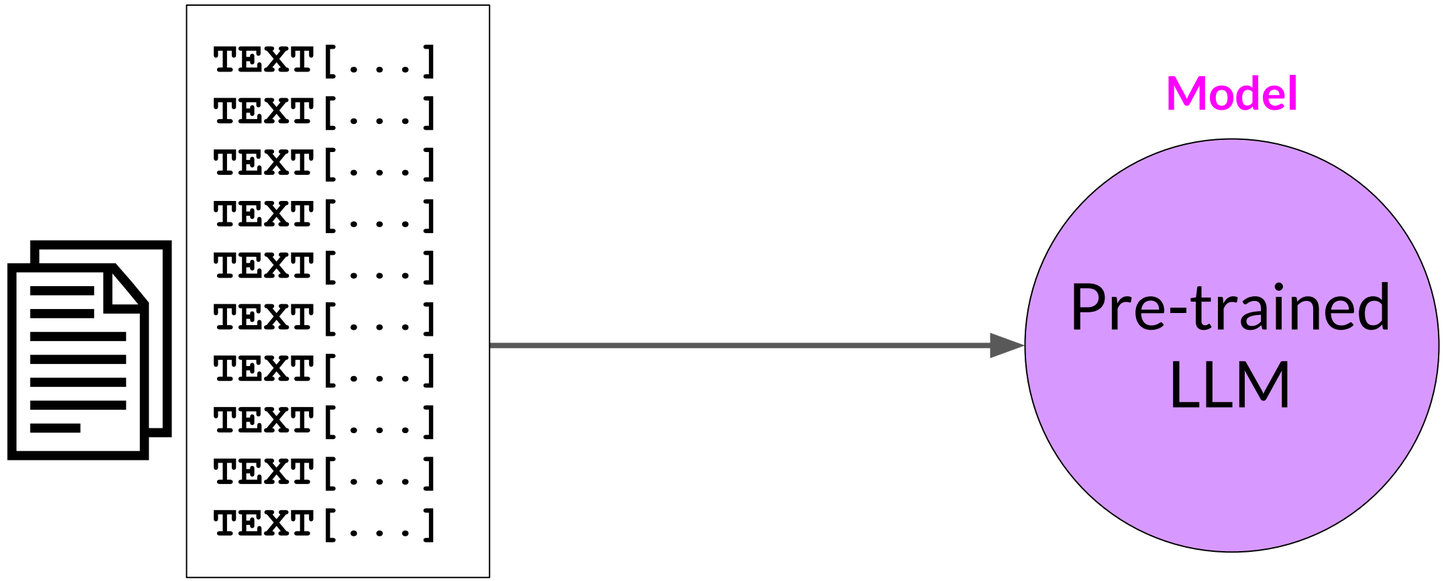
Limitations of in-context learning

- In-context learning may not work for smaller models LLM
- Examples take up space in the context window

Instead, try **fine-tuning** the model

LLM fine-tuning at a high level

LLM pre-training



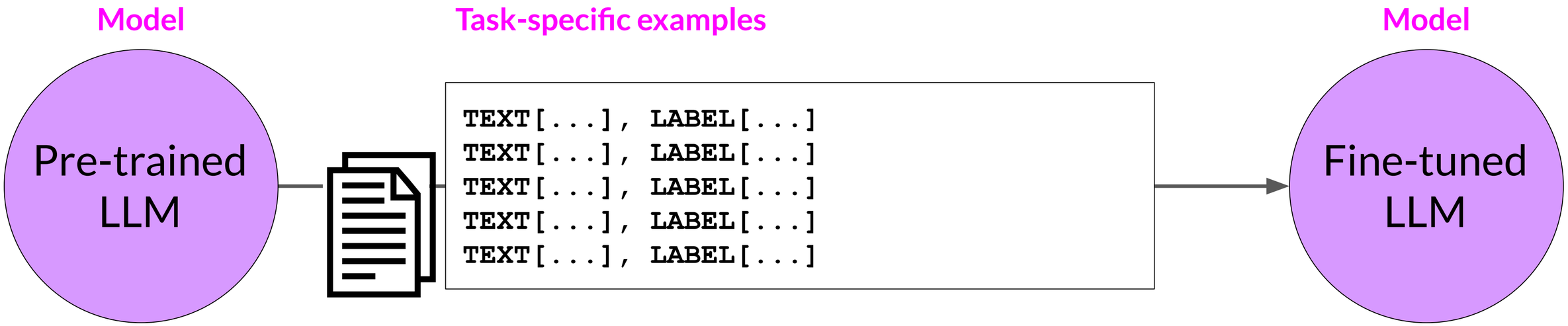
GB - TB - PB
of unstructured textual data

Pre-trained LLM Vs Fine-tuned LLM:
In pre-training, we train the LLM using vast amounts of unstructured textual data via Self-Supervised Learning (SSL), Whereas fine-tuning LLM is a supervised learning process where we use a data set of labeled examples to update the weights of the LLM.

Self-Supervised Learning (SSL) is a Machine Learning paradigm where a model, when fed with unstructured data as input, generates data labels automatically, which are further used in subsequent iterations as ground truths.

LLM fine-tuning at a high level

LLM fine-tuning



GB - TB
of labeled examples for a specific task or set of tasks

LLM fine-tuning at a high level

the fine-tuning process extends the training of the model to improve its ability to generate good completions for a specific task.

LLM fine-tuning

Model

Pre-trained LLM

Task-specific examples

```
PROMPT[...], COMPLETION[...]
PROMPT[...], COMPLETION[...]
PROMPT[...], COMPLETION[...]
PROMPT[...], COMPLETION[...]
PROMPT[...], COMPLETION[...]
```

Model

Fine-tuned LLM

Improved performance

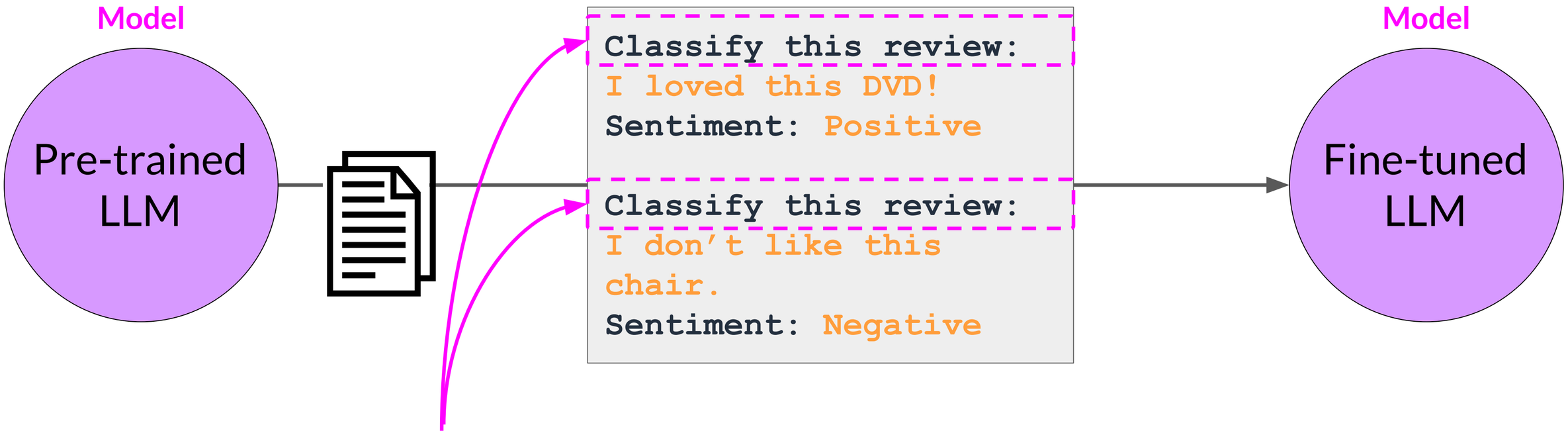
GB - TB
of labeled examples for a specific task or set of tasks

Prompt-completion pairs

Labeled examples are Prompt-Completion pairs where Prompt represents Input and Completion represents the Target or Output of the labeled dataset

Using prompts to fine-tune LLMs with instruction

LLM fine-tuning

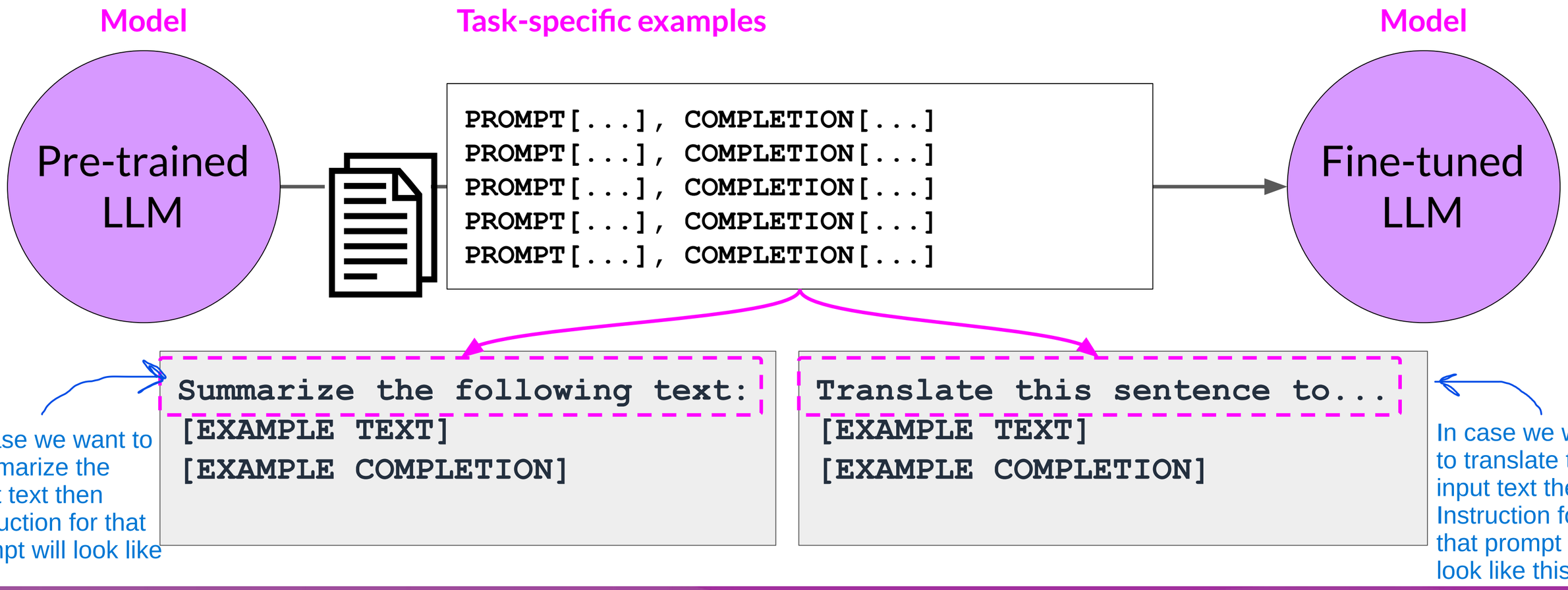


Each prompt/completion pair includes a specific "instruction" to the LLM

In above example we have 2 prompts with Instruction: "Classify this review:"

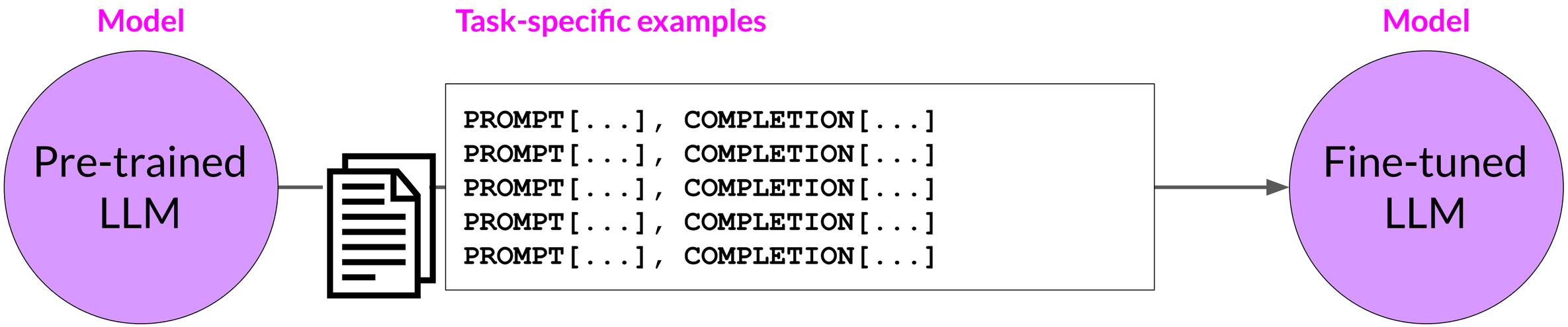
Using prompts to fine-tune LLMs with instruction

LLM fine-tuning



Using prompts to fine-tune LLMs with instruction

LLM fine-tuning



Full fine-tuning updates all parameters

Instruction fine-tuning, where all of the model's weights are updated is known as full fine-tuning. The process results in a new version of the model with updated weights. It is important to note that just like pre-training, full fine tuning requires enough memory and compute budget to store and process all the gradients, optimizers and other components that are being updated during training. So you can benefit from the memory optimization (Quantization) and parallel computing strategies that you learned about last week.

Improved performance

Sample prompt instruction templates

Sticky note 1

Instruction of Text classification Prompt

Classification / sentiment analysis

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

Instruction of Text generation Prompt

Text generation

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

Instruction of Text summarization Prompt

Text summarization

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

Source: https://github.com/bigscience-workshop/promptsource/blob/main/promptsource/templates/amazon_polarity/templates.yaml

LLM fine-tuning process

LLM fine-tuning

Once you have your instruction data set ready, as with standard supervised learning, you divide the data set into training validation, and test splits.

Prepared instruction dataset

Training splits

```
PROMPT[...], COMPLETION[...]
PROMPT[...], COMPLETION[...]
PROMPT[...], COMPLETION[...]
PROMPT[...], COMPLETION[...]
PROMPT[...], COMPLETION[...]
```

Training

```
PROMPT[...], COMPLETION[...]
...
```

Validation

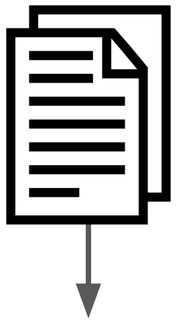
```
PROMPT[...], COMPLETION[...]
...
```

Test

LLM fine-tuning process

LLM fine-tuning

Prepared instruction dataset



Prompt:

```
Classify this review:
I loved this DVD!

Sentiment:
```

Model

Pre-trained LLM

LLM completion:

```
Classify this review:
I loved this DVD!

Sentiment: Neutral
```

Label:

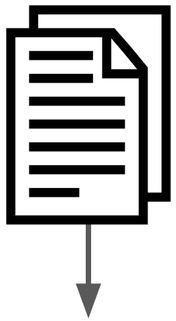
```
Classify this review:
I loved this DVD!

Sentiment: Positive
```

LLM fine-tuning process

Sticky note 2

LLM fine-tuning

Prepared instruction dataset



Prompt:

```
Classify this review:
I loved this DVD!

Sentiment:
```

Model

Pre-trained LLM

LLM completion:

```
Classify this review:
I loved this DVD!

Sentiment: Neutral
```

Label:

```
Classify this review:
I loved this DVD!

Sentiment: Positive
```

Loss: Cross-Entropy

LLM fine-tuning process

LLM fine-tuning

Prepared instruction dataset

Training splits

```
PROMPT[...], COMPLETION[...]
PROMPT[...], COMPLETION[...]
PROMPT[...], COMPLETION[...]
PROMPT[...], COMPLETION[...]
PROMPT[...], COMPLETION[...]
```

Training

```
PROMPT[...], COMPLETION[...]
...
```

Validation

```
PROMPT[...], COMPLETION[...]
...
```

Test

validation_accuracy

LLM fine-tuning process

LLM fine-tuning

Prepared instruction dataset

Training splits

```
PROMPT[...], COMPLETION[...]
PROMPT[...], COMPLETION[...]
PROMPT[...], COMPLETION[...]
PROMPT[...], COMPLETION[...]
PROMPT[...], COMPLETION[...]
```

Training

```
PROMPT[...], COMPLETION[...]
...
```

Validation

```
PROMPT[...], COMPLETION[...]
...
```

Test

test_accuracy

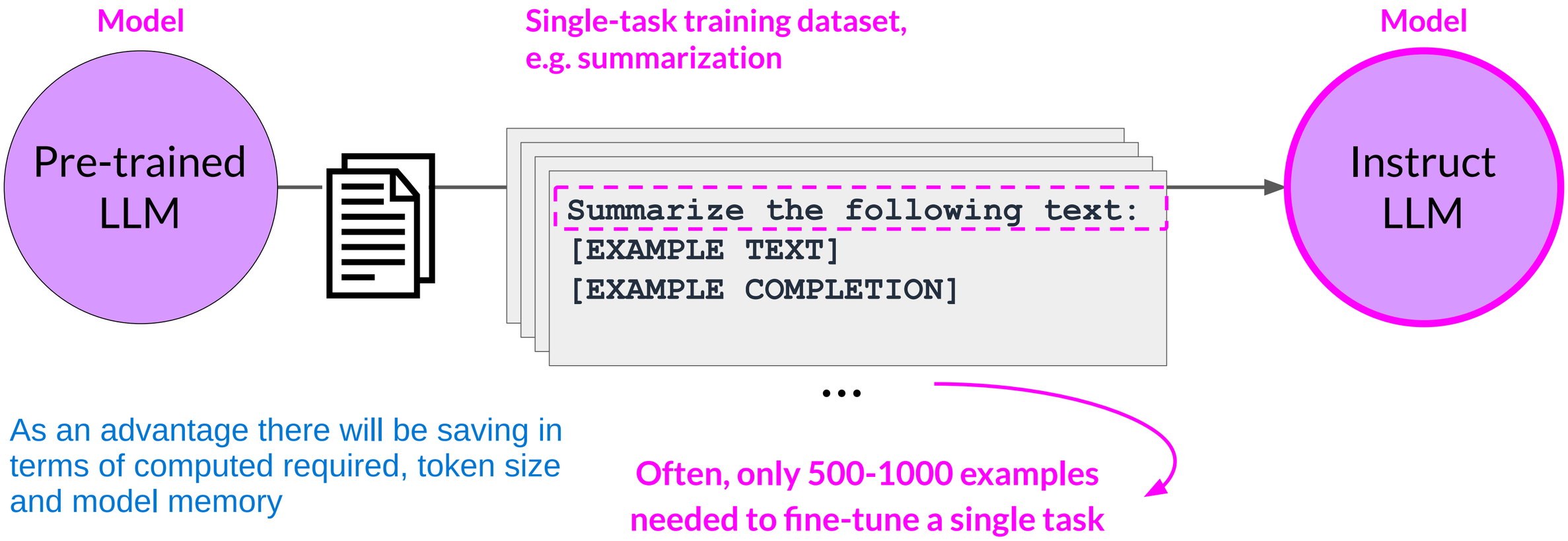
LLM fine-tuning process

The fine-tuning process results in a new version of the base model, often called an instruct model that is better at the tasks you are interested in. Fine-tuning with instruction prompts is the most common way to fine-tune LLMs these days. From this point on, when you hear or see the term fine-tuning, you can assume that it always means instruction fine tuning.

Fine-tuning on a single task

A single LLM model can be fine tuned(Instruction fine tuning) for to carry out variety of different language task by varying the Input Prompt. However cases where we want out model to perform only single language task (like text summarization) in that case few training examples would be needed to fine tune our LLM model. This is called Fine-tuning on a single task.

Fine-tuning on a single task



Catastrophic forgetting

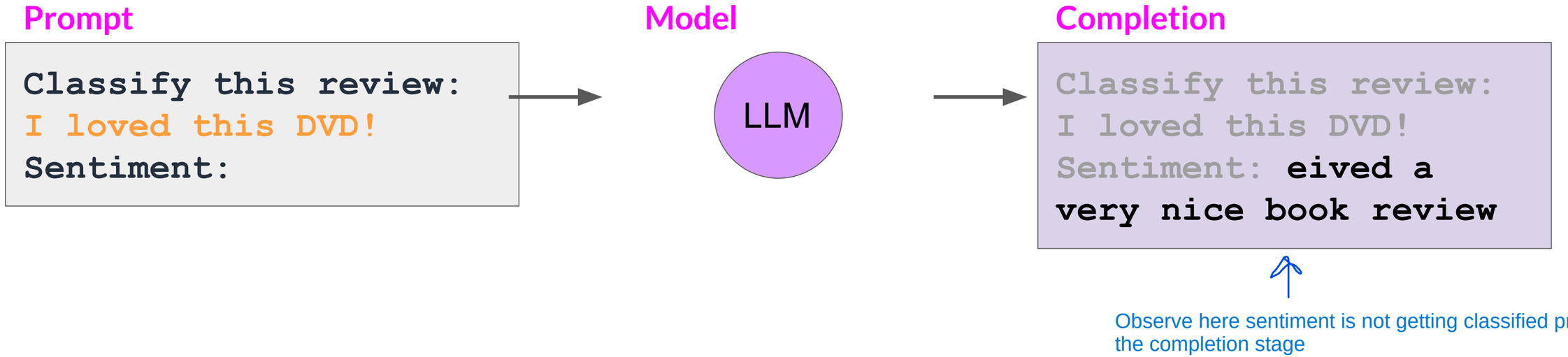
Only downside or disadvantage of Fine-tunning a LLM on a single task is Catastrophic forgetting.

Please note that this phenomenon will happen only when model is pre trained using input prompts having only single type of instruction that represents the single task

- Fine-tuning can significantly increase the performance of a model on a specific task…

Let's understand Catastrophic Forgetting with the help of an example: Suppose We want to train our model to classify the sentiment of the Input. Since we want our model to perform only single task hence model can be pretrained over input prompt having only one kind of instruction: Classify this review:(Instruction fine tuning)

Before fine-tuning



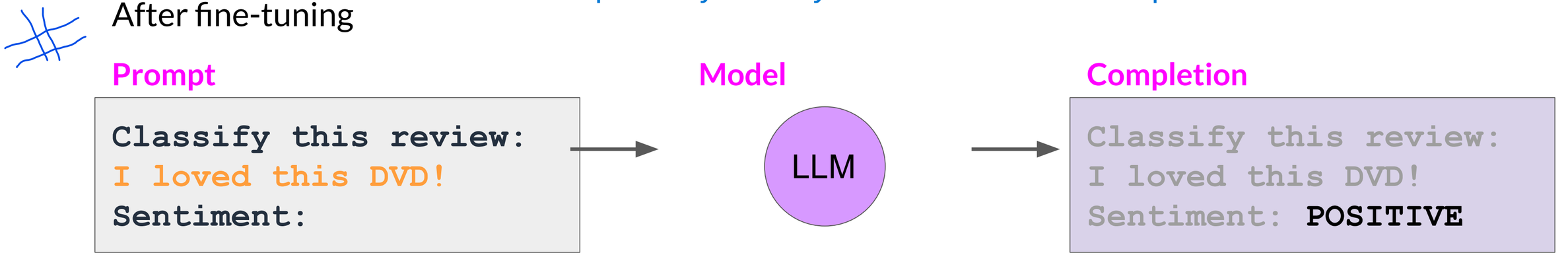
Observe here sentiment is not getting classified properly in the completion stage

Catastrophic forgetting

- Fine-tuning can significantly increase the performance of a model on a specific task…

Here we can clearly observe that after fine tuning a LLM(existing pre-trained model) for a specific task(as discussed above) model is able to perfectly classify the sentiment of the input text

After fine-tuning

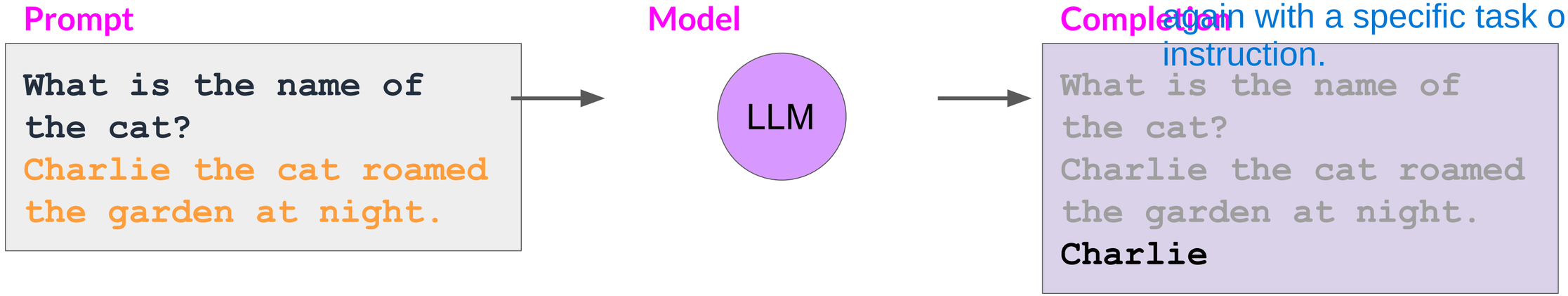


Catastrophic forgetting

- ...but can lead to reduction in ability on other tasks

Since now model is trained for only a single task or instruction due to which model's learning weight and other parameters are updated to accommodate this single task. So now if we try passing any new task or instruction then model may not be able to generate appropriate completion for that, which model would have otherwise done appropriately if it was not trained again with a specific task or instruction.

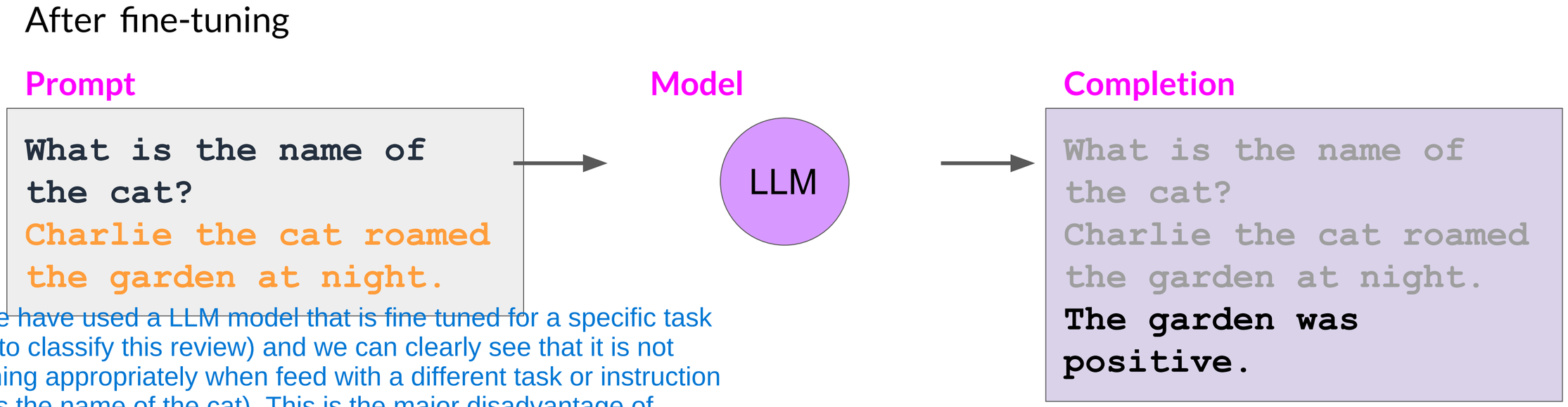
Before fine-tuning



Please note that here we have used a LLM model without doing any instruction fine tuning for a single specific task and we can see that model is performing well on the given input content and intruction

Catastrophic forgetting

- …but can lead to reduction in ability on other tasks



Here we have used a LLM model that is fine tuned for a specific task (that is to classify this review) and we can clearly see that it is not performing appropriately when feed with a different task or instruction (What is the name of the cat). This is the major disadvantage of training LLM for a specific task. That is LLM losses it's memory in terms of deep contextual understanding for other tasks or instructions which earlier it might have remembered perfectly

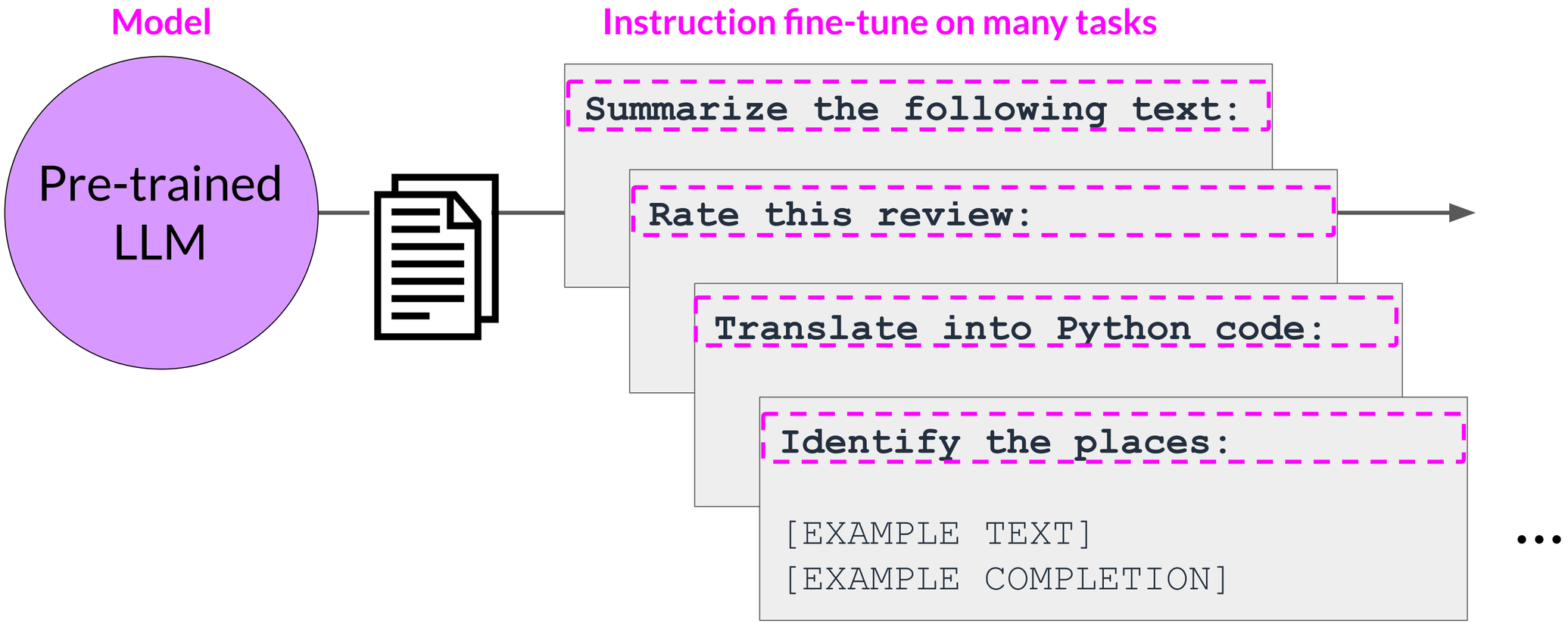
How to avoid catastrophic forgetting

- First note that you might not have to!
- Fine-tune on **multiple tasks** at the same time
- Consider **Parameter Efficient Fine-tuning** (PEFT)

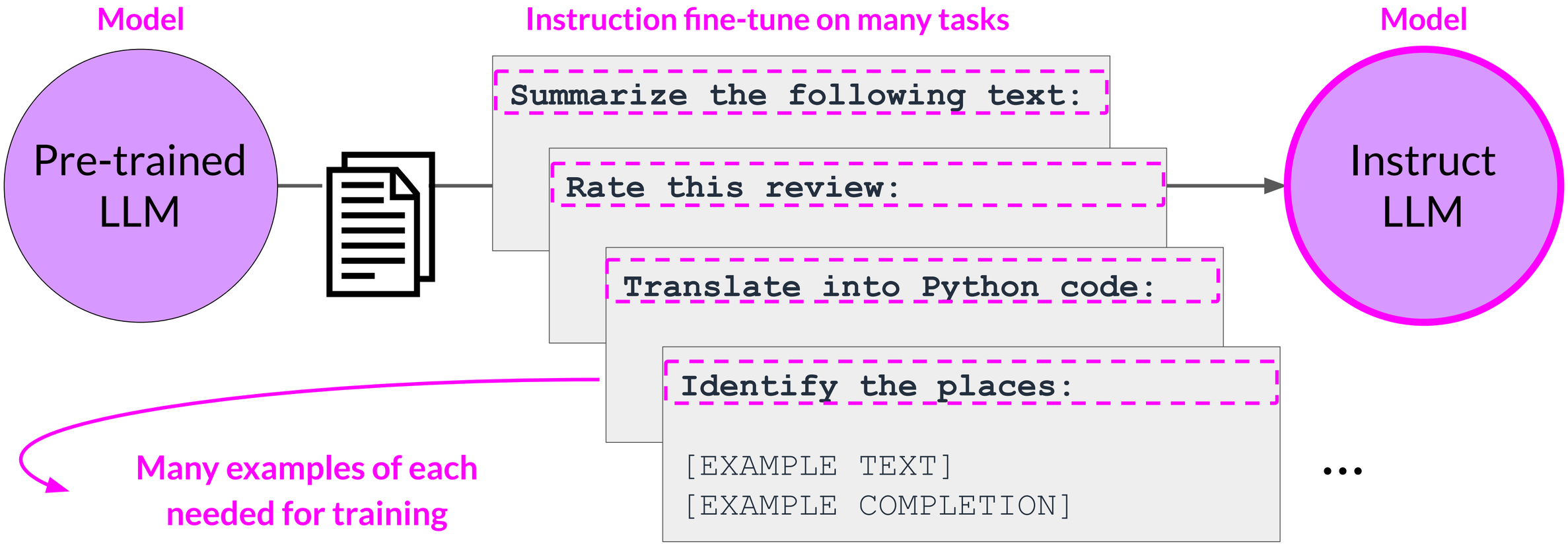
Multi-task, instruction fine-tuning

Multi-task instruction fine tuning is the approach in which model is trained to be good at multiple tasks which is achieved by training the model using labeled dataset where Input has instructions for various different tasks that model is expected to act on and the respective Completions in the form of Output.

Multi-task, instruction fine-tuning



Multi-task, instruction fine-tuning



One drawback to multitask fine-tuning is that it requires a lot of data. You may need as many as 50-100,000 examples in your training set. However, it can be really worthwhile and worth the effort to assemble this data. The resulting models are often very capable and suitable for use in situations where good performance at many tasks is desirable.

Instruction fine-tuning with FLAN

FLAN stands for fine-tuned language net which is an example of family of models that is trained using Mutli-task instruction fine-tuning approach

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

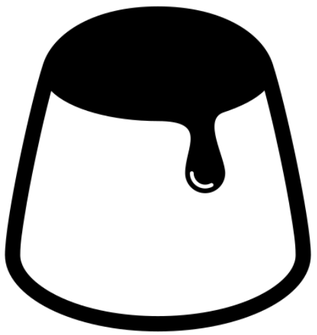
FLAN

"*The metaphorical dessert to the main course of pretraining*"

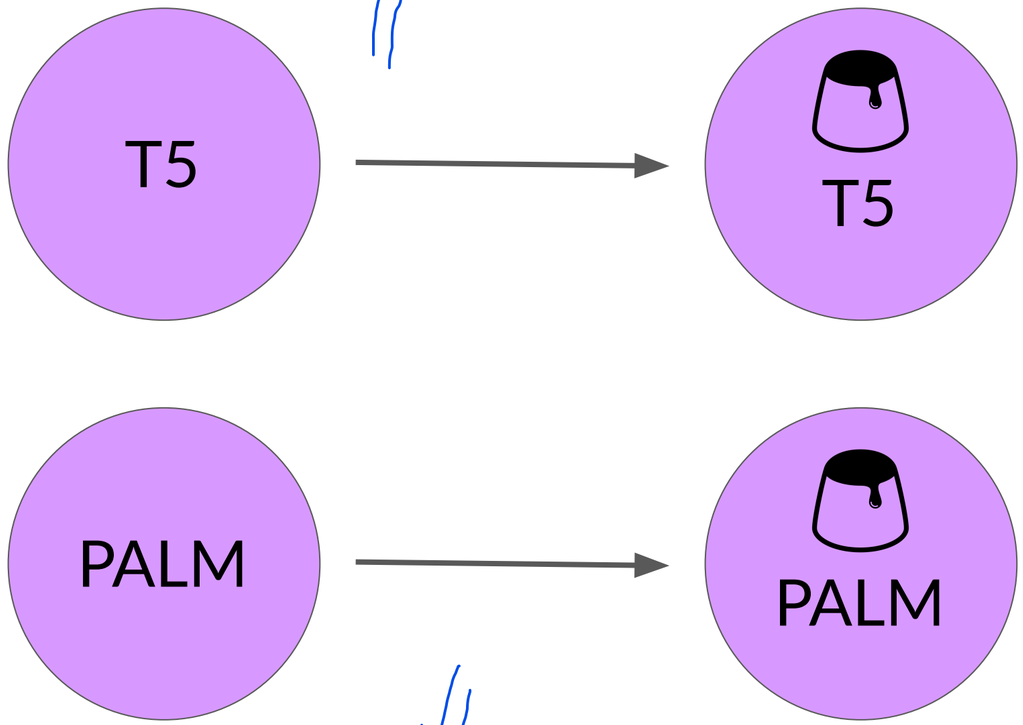
Since FLAN fine-tuning is the last step of the training process the authors of the original paper called it the metaphorical dessert to the main course of pre-training. Quite a fitting name!

Instruction fine-tuning with FLAN

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



FLAN
(**F**ine-tuned
LAnguage **N**et)



T5 is base pre trained LLM whereas, FLAN-T5 is Fined tuned LLM instruct model

FLAN-T5 is the FLAN instruct version of the T5 foundation mode

FLAN-T5

FLAN-PALM is the FLAN instruct version of the PALM foundation mode

FLAN-PALM

PALM is base pre trained LLM whereas, FLAN-PALM is Fined tuned LLM instruct model

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

- FLAN-T5 is a great, general purpose, instruct model which has been fine tuned on 473 datasets across 146 task categories. Those datasets are chosen from other models and papers as shown here.

T0-SF

- Commonsense Reasoning,
- Question Generation,
- Closed-book QA,
- Adversarial QA,
- Extractive QA

…

55 Datasets
14 Categories
193 Tasks

Muffin

- Natural language inference,
- Code instruction gen,
- Code repair
- Dialog context generation,
- Summarization (SAMSum)

…

69 Datasets
27 Categories
80 Tasks

CoT (reasoning)

- Arithmetic reasoning,
- Commonsense reasoning
- Explanation generation,
- Sentence composition,
- Implicit reasoning,

…

9 Datasets
1 Category
9 Tasks

Natural Instructions

- Cause effect classification,
- Commonsense reasoning,
- Named Entity Recognition,
- Toxic Language Detection,
- Question answering

…

372 Datasets
108 Categories
1554 Tasks

Source: Chung et al. 2022, “Scaling Instruction-Finetuned Language Models”

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

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

T0-SF

- Commonsense Reasoning,
- Question Generation,
- Closed-book QA,
- Adversarial QA,
- Extractive QA

...

55 Datasets
14 Categories
193 Tasks

Muffin

- Natural language inference,
- Code instruction gen,
- Code repair
- Dialog context generation,
- **Summarization (SAMSum)**

...

69 Datasets
27 Categories
80 Tasks

CoT (reasoning)

- Arithmetic reasoning,
- Commonsense reasoning
- Explanation generation,
- Sentence composition,
- Implicit reasoning,

...

9 Datasets
1 Category
9 Tasks

Natural Instructions

- Cause effect classification,
- Commonsense reasoning,
- Named Entity Recognition,
- Toxic Language Detection,
- Question answering

...

372 Datasets
108 Categories
1554 Tasks

SAMSum is the dataset from Muffin collection used for Text Summarization or is used to train language model to summarize dialogues and FLAN-T5 is already fine-tuned over this dataset.

Source: Chung et al. 2022, “Scaling Instruction-Finetuned Language Models”

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."

This kind of dataset is prepared manually by linguistic experts

Source: https://huggingface.co/datasets/samsum, https://github.com/google-research/FLAN/blob/2c79a31/flan/v2/templates.py#L3285

Sample FLAN-T5 prompt templates

```
"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}"),
]
```

This is the Prompt template used for SAMSum dataset where all the instructions ask the LLM to do the same thing

Represents the different Instructions present in the Prompt having the same meaning

Sample FLAN-T5 prompt templates

Input or context in the prompt upon which instruction will be applied to generate the completion(Output)

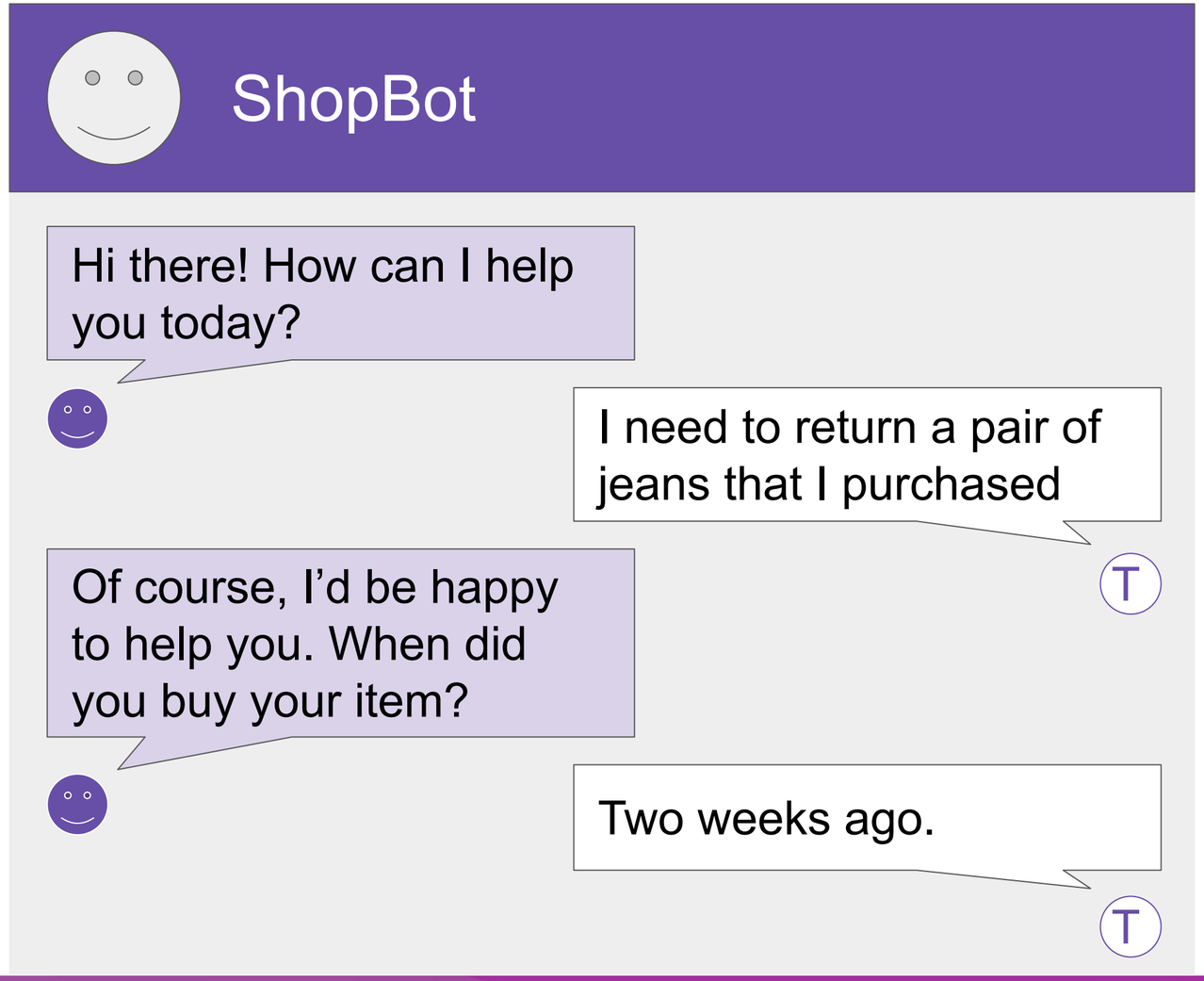
INSTRUCTION

Completion or Output

```
"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}"),
]
```

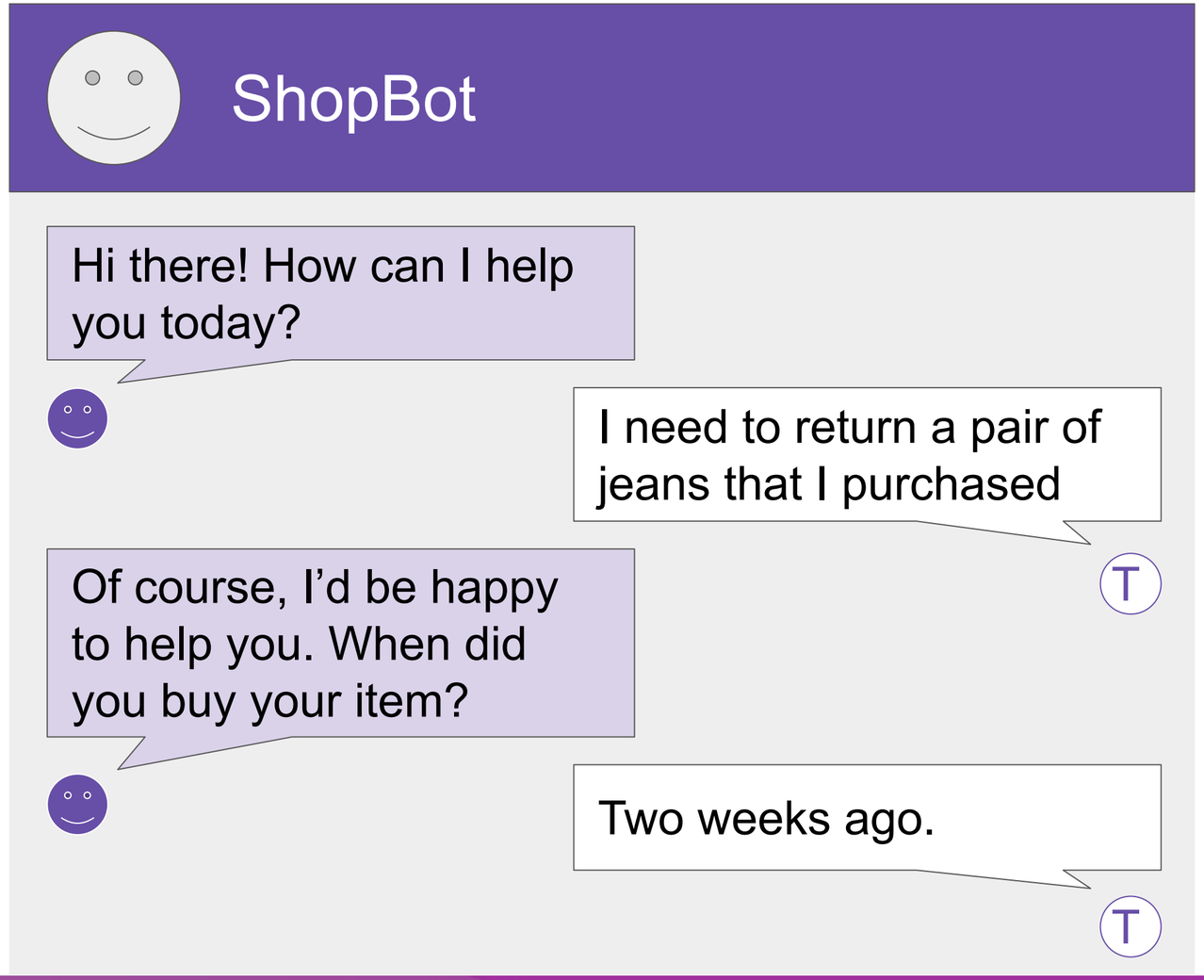
Improving FLAN-T5’s summarization capabilities

Suppose that you're a data scientist building an app to support your customer service team, process requests received through a chat bot, like the one shown here. Your customer service team needs a summary of every dialogue to identify the key actions that the customer is requesting and to determine what actions should be taken in response.



Improving FLAN-T5's summarization capabilities

We have seen that SamSUM dataset pre trained over FLAN-T5(General purpose LLM) can summarize dialogues. But SamSUM dataset contains the dialogues that 2 friends will converse or have in day to day activities. Whereas in this use case we are solving a domain use case which is different. That is the way customer and customer support representative will have dialogue will be entirely different as compared to the conversation between 2 friends. This is why it is said that Language is very complex.

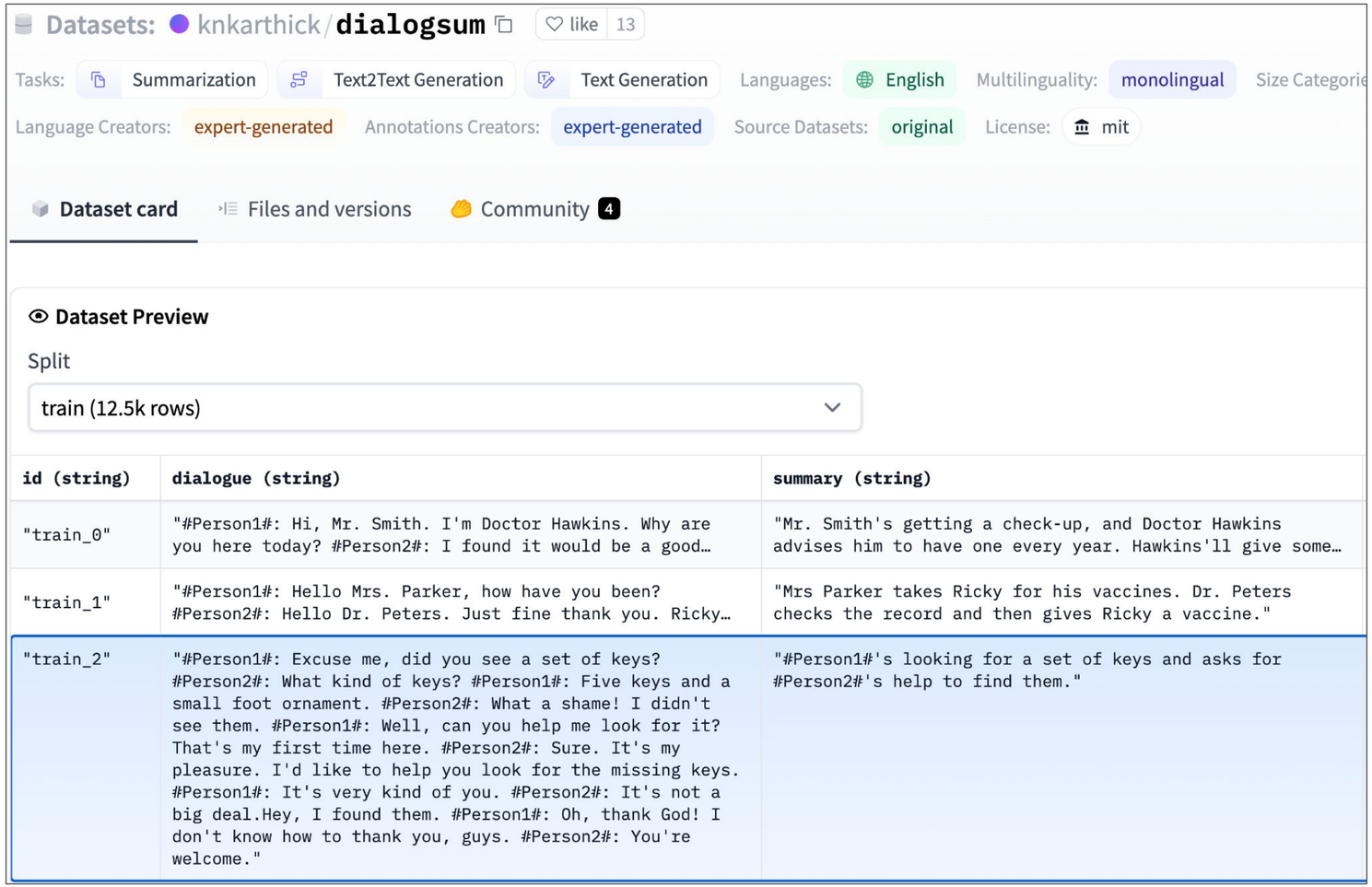


Goal: Summarize conversations to identify actions to take

Improving FLAN-T5's summarization capabilities

Further fine-tune FLAN-T5 with a domain-specific instruction dataset (**dialogsum**)

To get rid of above problem we have a dataset called dialogsum which is a customer service dialogue dataset. FLAN-T5 has been never pre-trained on top of this. In Week 2 Lab we will try implementing this



Example support-dialog summarization

Prompt (created from template)

```
Summarize the following conversation.
Tommy: Hello. My name is Tommy Sandals, I have a reservation.
Mike: May I see some identification, sir, please?
Tommy: Sure. Here you go.
Mike: Thank you so much. Have you got a credit card, Mr.
Sandals?
Tommy: I sure do.
Mike: Thank you, sir. You'll be in room 507, nonsmoking,
queen bed.
Tommy: That's great, thank you!
Mike: Enjoy your stay!
```

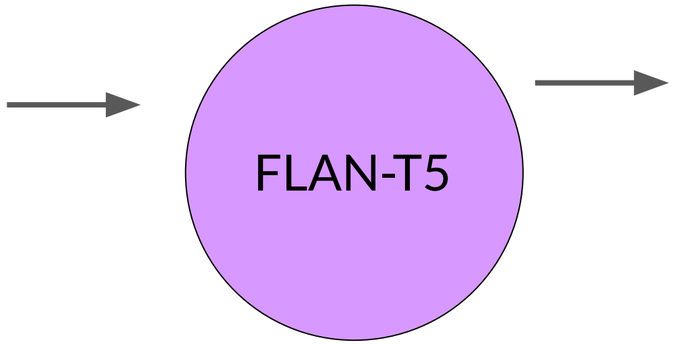
Source: https://huggingface.co/datasets/knkarthick/dialogsum/viewer/knkarthick--dialogsum/

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



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.

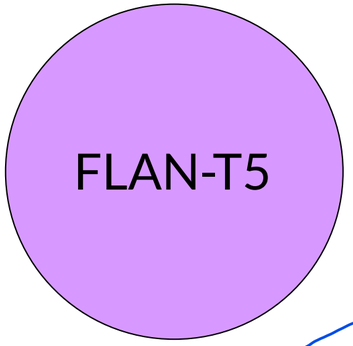
```
Human baseline summary:
Tommy Sandals has got a
reservation. Mike asks for his
identification and credit card
and helps his check-in.
```

Summary **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



Completion (Summary)

Predicted Summary

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

Model added this extra word by it's own which is completely wrong

Adequate completion, but does not match human baseline.

Actual expected summary

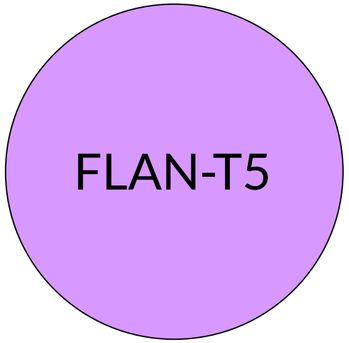
```
Human baseline summary:
Tommy Sandals has got a
reservation. Mike asks for his
identification and credit card
and helps his check-in.
```

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



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.

Model added this extra word by it's own which is completely wrong

Human baseline summary:

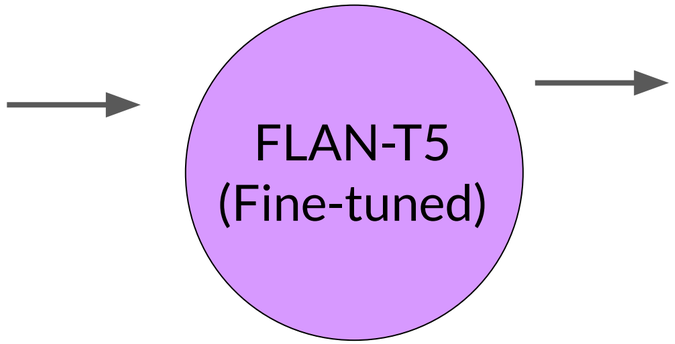
```
Tommy Sandals has got a
reservation. Mike asks for his
identification and credit card
and helps his check-in.
```

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

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

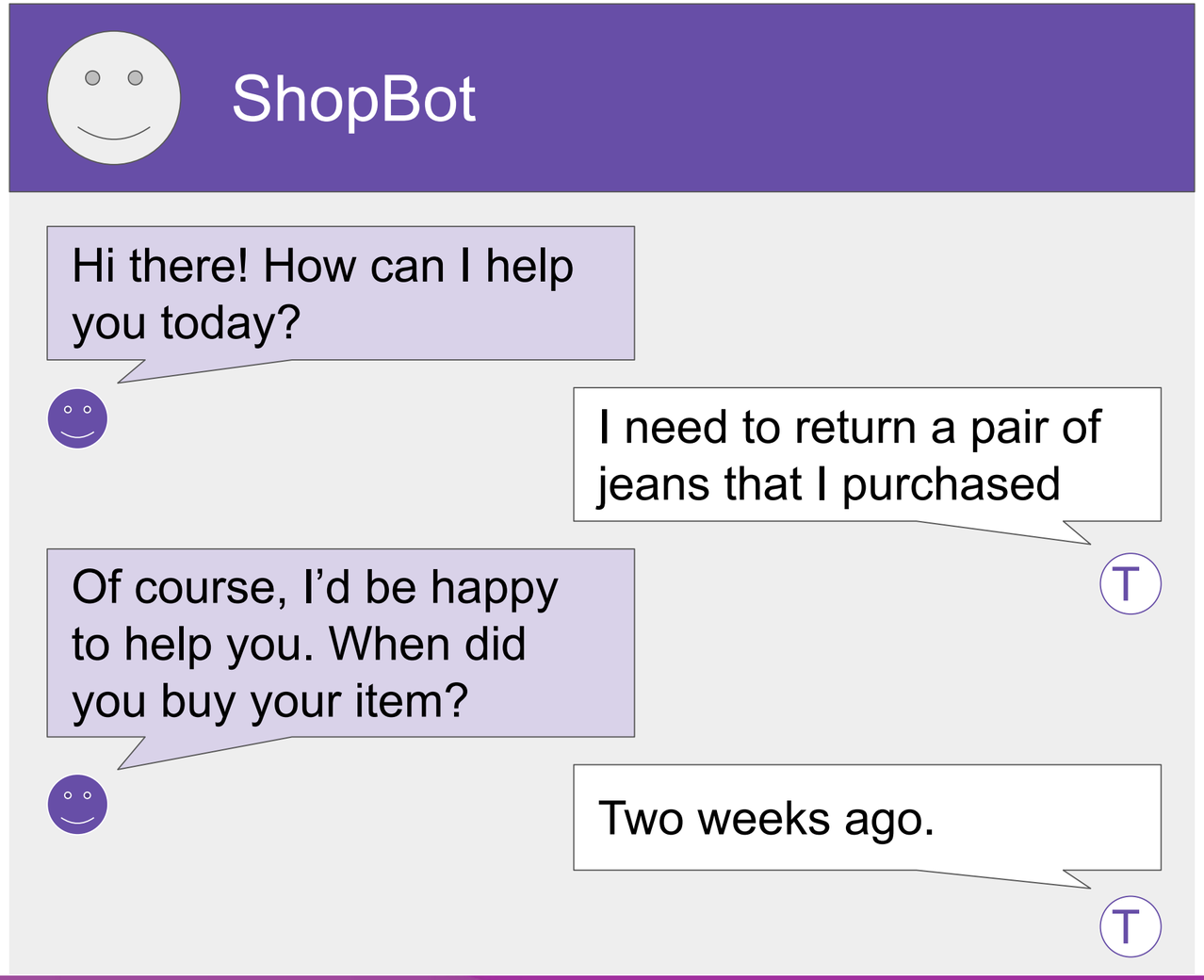


Completion (Summary)

```
Tommy Sandals has a
reservation and checks in
showing his ID and credit
card. Mike helps him to
check in and approves his
reservation.
```

Better summary, more-closely matches human baseline.

Fine-tuning with your own data



This example, use the public dialogue, some dataset to demonstrate fine-tuning on custom data. In practice, you'll get the most out of fine-tuning by using your company's own internal data. For example, the support chat conversations from your customer support application. This will help the model learn the specifics of how your company likes to summarize conversations and what is most useful to your customer service colleagues.

Model evaluation metrices are being used to evaluated that how well our fine-tuned LLM is performing over the base model (Pre-trained model that were not fine tuned)

Model evaluation metrics

LLM Evaluation - Challenges

In traditional ML where Output is deterministic we use accuracy score for model evaluation. The same cannot be used in case of LLMs since output is non deterministic because of language-based evaluation is comparatively more challenging.

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

Deterministic algorithms are entirely predictable and always produce the same output for the same input. Non-deterministic algorithms may produce different outputs for the same input due to random events or other

LLM Evaluation - Challenges

“Mike really loves drinking tea.”

=

“Mike adores sipping tea.”

“Mike does not drink coffee.”

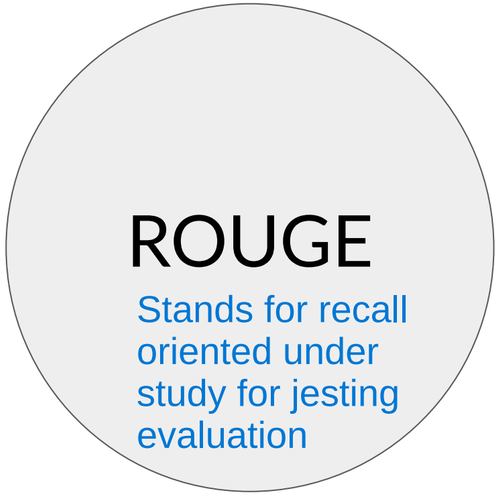
≠

“Mike does drink coffee.”

for humans like us with squishy organic brains, we can see the similarities and differences. But when you train a model on millions of sentences, you need an automated, structured way to make measurements

LLM Evaluation - Metrics

ROUGE and BLEU, are two widely used evaluation metrics for different tasks



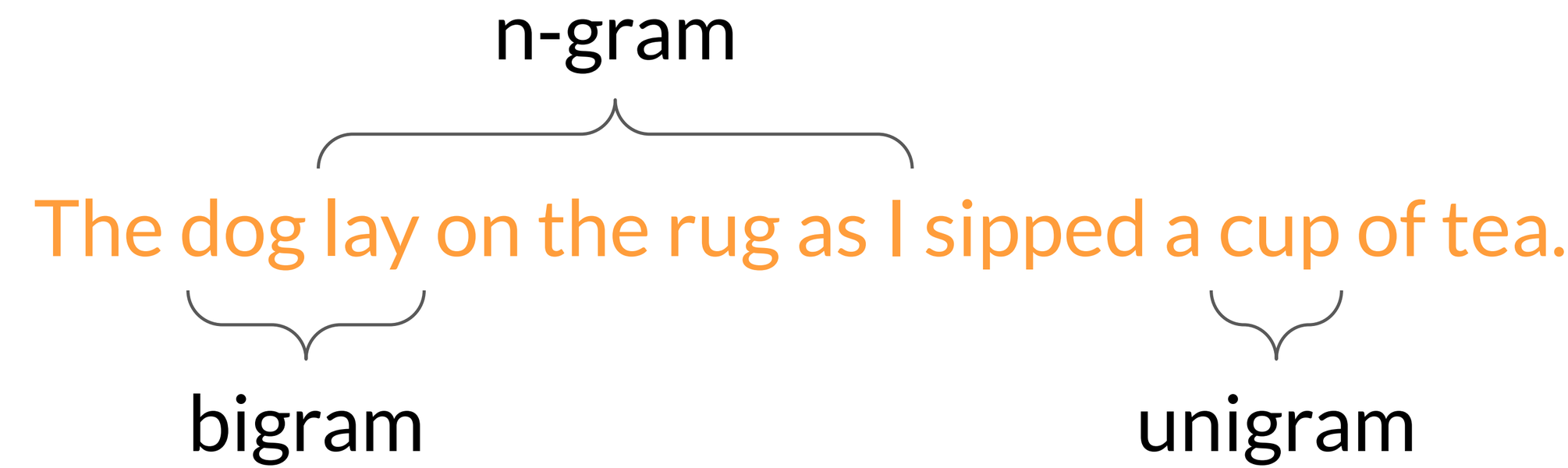
- Used for text summarization
- Compares a summary to one or more reference summaries

(reference summaries that atre generated by the humans)

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

LLM Evaluation - Metrics - Terminology

1 word = Unigram
2 words = Bigram
Collection of many words = n-gram



LLM Evaluation - Metrics - ROUGE-1

(1 represents that unigram to be considered)

Reference (human):

It is cold outside.

Generated output:

It is very cold outside.

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

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

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

LLM Evaluation - Metrics - ROUGE-1

Reference (human):

It is cold outside.

Generated output:

It is not cold outside.

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

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

Observe that ROUGE-1 metrics produced same results when we changed a single word from very to not(in Generated Output). Which means model is giving same performance for both correctly generated text and wrongly generated text which is extremely poor

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

LLM Evaluation - Metrics - ROUGE-2

To overcome above stated problem we can take account or bigrams which is 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

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

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

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

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

Observe here scores are a lot lower as compared to ROUGE-1. With longer sentences, they're a greater chance that bigrams don't match, and the scores may be even lower.

LLM Evaluation - Metrics - ROUGE-L

Rather than continue on with ROUGE numbers growing bigger to n-grams of three or fours, let's take a different approach which is ROUGE-L in which we look for the longest common subsequence present in both the generated output and the reference output.

Reference (human):

It is cold outside.

Generated output:

It is very cold outside.

Longest common subsequence (LCS):

It is

cold outside

2

In this example we have 2 LCS each having the length equivalent to 2

LLM Evaluation - Metrics - ROUGE-L

Reference (human):

It is cold outside.

Generated output:

It is very cold outside.

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

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

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

Collectively these three values are known as ROUGE-L score

LLM Evaluation - Metrics - ROUGE-L

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 - ROUGE hacking

Reference (human):

It is cold outside.

Generated output:

cold cold cold cold

Earlier we have seen(in the example of ROUGE-1) that bad completion(Predicted Output) can result in good score. This is called ROUGE hacking

LLM Evaluation - Metrics - ROUGE clipping

Reference (human):

It is cold outside.

Generated output:

cold cold cold cold

for ROUGE hacking

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

ROUGE hacking problem can be countered using ROUGE clipping where we are just simply clipping or removing out extra duplicates

$$\text{Modified precision} = \frac{\text{clip(unigram matches)}}{\text{unigrams in output}} = \frac{1}{4} = 0.25$$

Generated output:

outside cold it is

$$\text{Modified precision} = \frac{\text{clip(unigram matches)}}{\text{unigrams in output}} = \frac{4}{4} = 1.0$$

But say in this example where Generated Text is having wrong order then again this bad completion will give good score which cannot be countered using ROUGE clipping

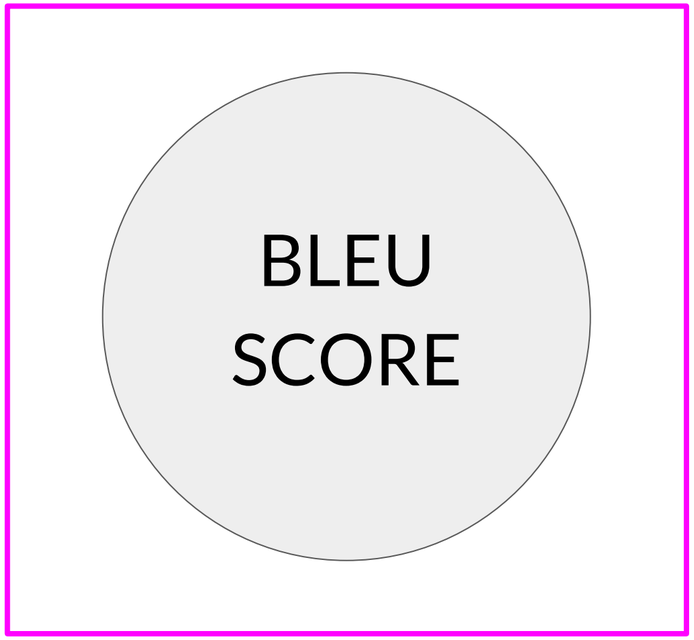
Please note that Rouge clipping just a modification on top of existing ROUGE-1, ROUGE-2 and ROUGE-3

TO summarize: using a different rouge score can help experimenting with a n-gram size that will calculate the most useful score will be dependent on the sentence, the sentence size, and your use case.

LLM Evaluation - Metrics

ROUGE

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



Stands for bilingual evaluation under study

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

LLM Evaluation - Metrics - BLEU

BLEU evaluation score is calculated using the average precision over multiple n-gram sizes. Just like the Rouge-1 score that we looked at before, but calculated for a range of n-gram sizes and then averaged.

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

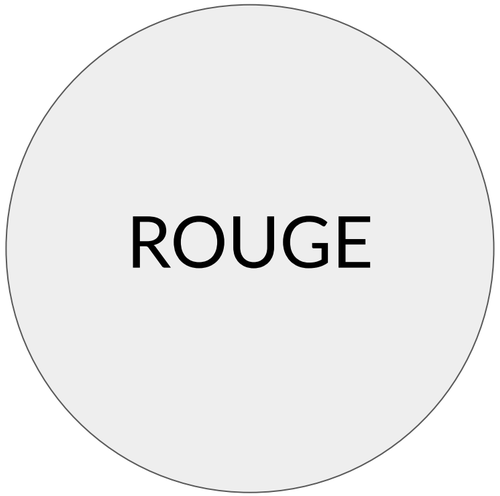
I am very happy that I am drinking a warm cup of tea. - BLEU 0.730

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

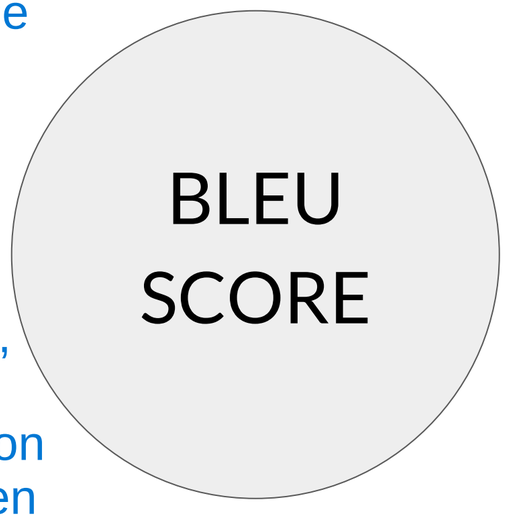
I am very happy to say that I am drinking a warm cup of tea. - BLEU 1.000

Calculating these BLEU score by hand will result into multiple calculations. Instead we may use pre built libraries to calculate the same. One is provided by hugging face itself

LLM Evaluation - Metrics



We shouldn't use these 2 metrices alone to report the final evaluation of a large language model. These should be used only for diagnostic evaluation. For overall evaluation of your model's performance, however, you will need to look at one of the evaluation benchmarks that have been developed by researchers.



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

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

In order to measure and compare LLMs more holistically, you can make use of pre-existing datasets, and associated benchmarks that have been established by LLM researchers specifically for this purpose.

Benchmarks

Evaluation benchmarks

Benchmarks, such as GLUE, SuperGLUE, or Helm, cover a wide range of tasks and scenarios. They do this by designing or collecting datasets that test specific aspects of an LLM

1

GLUE

2

SuperGLUE

5

HELM

3

MMLU (Massive Multitask Language Understanding)

4

BIG-bench

GLUE

GLUE(General Language Understanding Evaluation) is a collection of natural language tasks, such as sentiment analysis and question-answering. GLUE was created to encourage the development of models that can generalize across multiple tasks, and you can use the benchmark to measure and compare the model performance.

The tasks included in SuperGLUE benchmark:

Corpus	|Train|	|Test|	Task	Metrics	Domain
			Single-Sentence Tasks		
CoLA	8.5k	**1k**	acceptability	Matthews corr.	misc.
SST-2	67k	1.8k	sentiment	acc.	movie reviews
			Similarity and Paraphrase Tasks		
MRPC	3.7k	1.7k	paraphrase	acc./F1	news
STS-B	7k	1.4k	sentence similarity	Pearson/Spearman corr.	misc.
QQP	364k	**391k**	paraphrase	acc./F1	social QA questions
			Inference Tasks		
MNLI	393k	**20k**	NLI	matched acc./mismatched acc.	misc.
QNLI	105k	5.4k	QA/NLI	acc.	Wikipedia
RTE	2.5k	3k	NLI	acc.	news, Wikipedia
WNLI	634	**146**	coreference/NLI	acc.	fiction books

Source: Wang et al. 2018, “GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding”

SuperGLUE

SuperGLUE is the extended version of GLUE. SuperGLUE is the series of tasks, some of which are not included in GLUE, and some of which are more challenging versions of the same tasks. SuperGLUE includes tasks such as multi-sentence reasoning, and reading comprehension

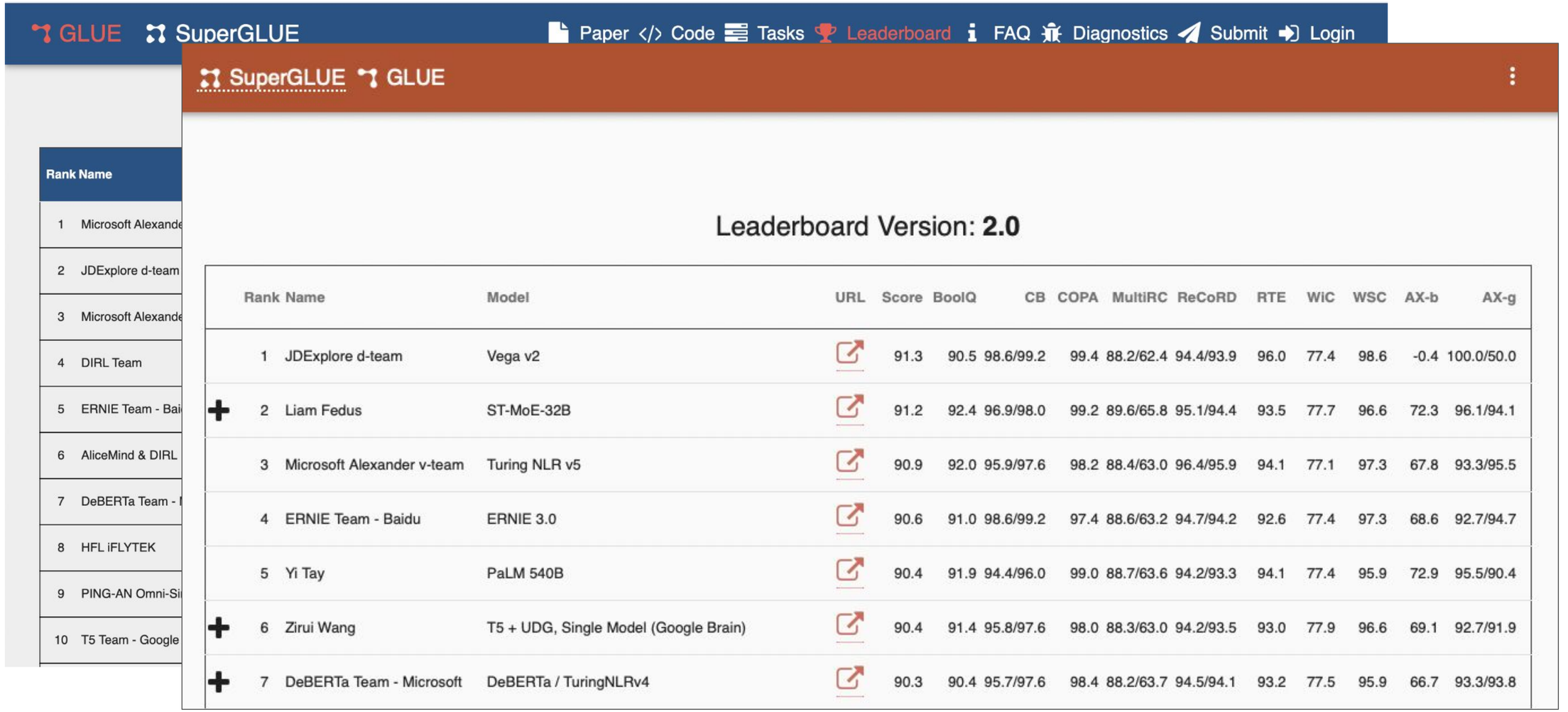
The tasks included in SuperGLUE benchmark:

Corpus	|Train|	|Dev|	|Test|	Task	Metrics	Text Sources
BoolQ	9427	3270	3245	QA	acc.	Google queries, Wikipedia
CB	250	57	250	NLI	acc./F1	various
COPA	400	100	500	QA	acc.	blogs, photography encyclopedia
MultiRC	5100	953	1800	QA	$F1_a$/EM	various
ReCoRD	101k	10k	10k	QA	F1/EM	news (CNN, Daily Mail)
RTE	2500	278	300	NLI	acc.	news, Wikipedia
WiC	6000	638	1400	WSD	acc.	WordNet, VerbNet, Wiktionary
WSC	554	104	146	coref.	acc.	fiction books

Source: Wang et al. 2019, “SuperGLUE: A Stickier Benchmark for General-Purpose Language Understanding Systems“

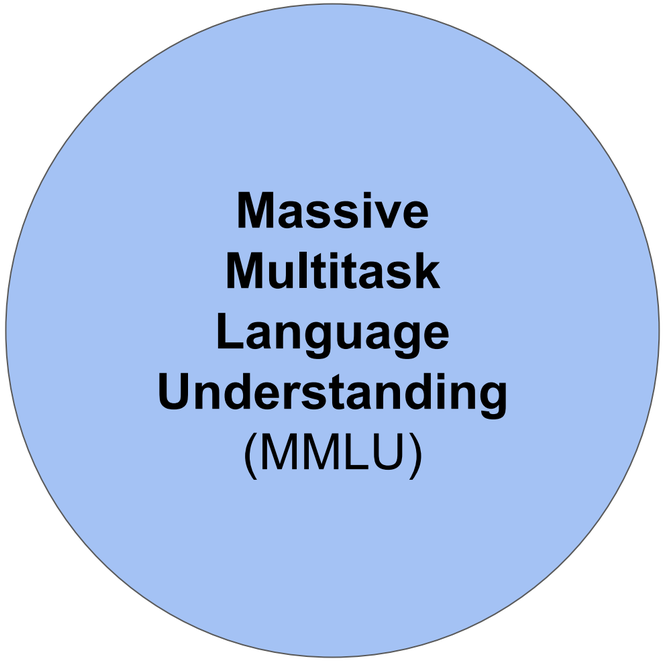
GLUE and SuperGLUE leaderboards

Both the GLUE and SuperGLUE benchmarks have leaderboards that can be used to compare and contrast evaluated models.



Disclaimer: metrics may not be up-to-date. Check https://super.gluebenchmark.com and https://gluebenchmark.com/leaderboard for the latest.

Benchmarks for massive models



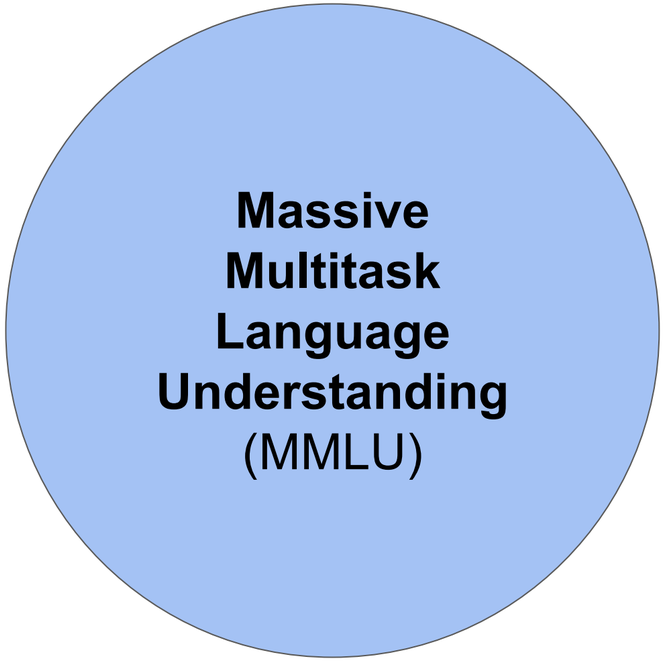
2021

Here are a couple of recent benchmarks that are pushing LLMs further. Massive Multitask Language Understanding, or MMLU, is designed specifically for modern LLMs. To perform well models must possess extensive world knowledge and problem-solving ability. Models are tested on elementary mathematics, US history, computer science, law, and more. In other words, tasks that extend way beyond basic language understanding.

Source: Hendrycks, 2021. “Measuring Massive Multitask Language Understanding”

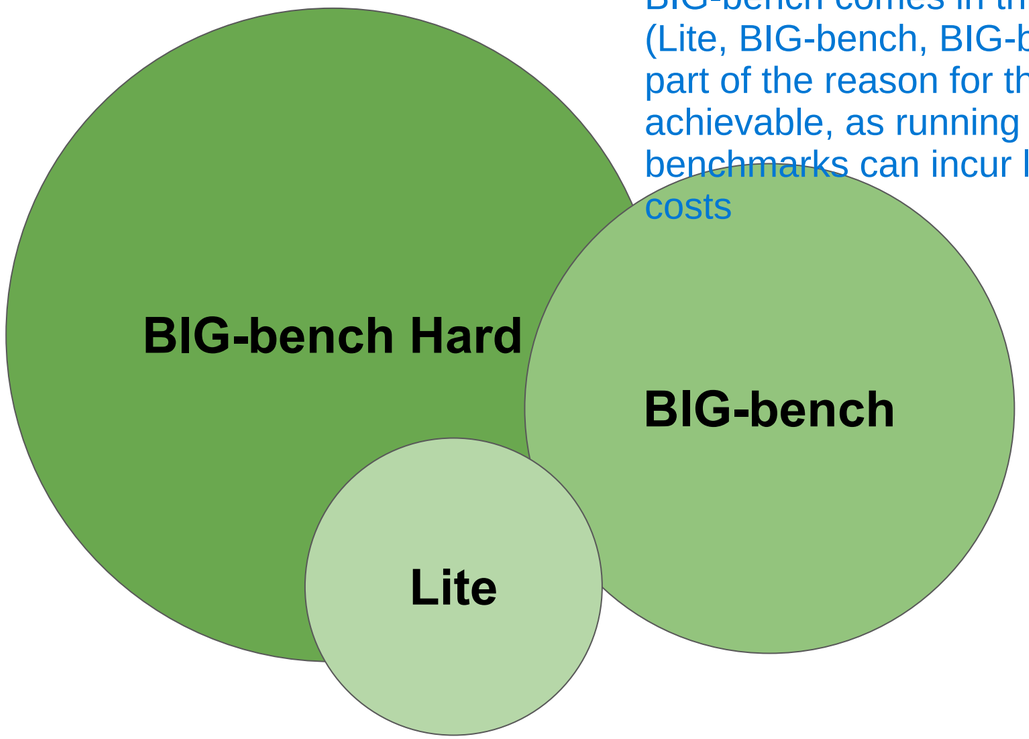
Benchmarks for massive models

BIG-bench currently consists of 204 tasks, ranging through linguistics, childhood development, math, common sense reasoning, biology, physics, social bias, software development and more. BIG-bench comes in three different sizes (Lite, BIG-bench, BIG-bench Hard), and part of the reason for this is to keep costs achievable, as running these large benchmarks can incur large inference costs



2021

Source: Hendrycks, 2021. “Measuring Massive Multitask Language Understanding”



2022

Source: Suzgun et al. 2022. “Challenging BIG-Bench tasks and whether chain-of-thought can solve them“

Holistic Evaluation of Language Models (HELM)

The HELM framework aims to improve the transparency of models, and to offer guidance on which models perform well for specific tasks

Metrics:

1. Accuracy
2. Calibration
3. Robustness
4. Fairness
5. Bias
6. Toxicity
7. Efficiency

Scenarios

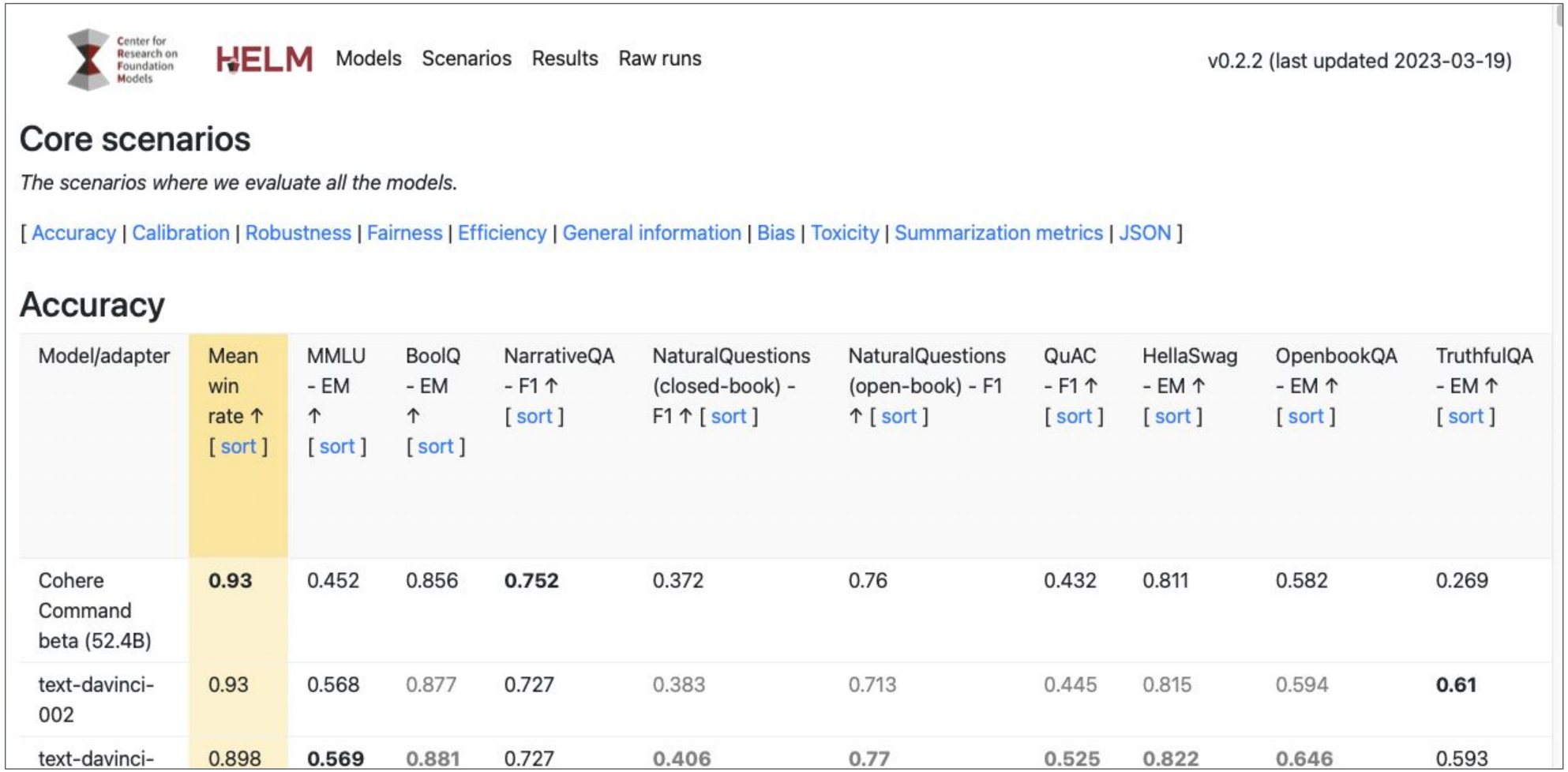
Models

	J1-Jumbo	J1-Grande	J1-Large	Anthropic-LM	BLOOM	T0pp	Cohere XL	Cohere Large	Cohere Medium	Cohere Small	GPT-NeoX
NaturalQuestions (open)			✔	✔	✔	✔	✔	✔	✔	✔	✔
NaturalQuestions (closed)	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
BoolQ	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
NarrativeQA	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
QuAC	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
HellaSwag	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
OpenBookQA	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
TruthfulQA	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
MMLU	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
MS MARCO				✔	✔		✔	✔	✔	✔	✔
TREC				✔	✔		✔	✔	✔	✔	✔
XSUM	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
CNN/DM	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
IMDB	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
CivilComments	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔
RAFT	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔	✔

HELM takes a multi-metric approach, measuring 7 metrics across 16 core scenarios, ensuring that trade-offs between models and metrics are clearly exposed

Holistic Evaluation of Language Models (HELM)

To conclude we can say that HELM is a living benchmark that aims to continuously evolve with the addition of new scenarios, metrics, and models.



Center for Research on Foundation Models

HELM Models Scenarios Results Raw runs

v0.2.2 (last updated 2023-03-19)

Core scenarios

The scenarios where we evaluate all the models.

[Accuracy | Calibration | Robustness | Fairness | Efficiency | General information | Bias | Toxicity | Summarization metrics | JSON]

Accuracy

Model/adapter	Mean win rate ↑ [sort]	MMLU - EM ↑ [sort]	BoolQ - EM ↑ [sort]	NarrativeQA - F1 ↑ [sort]	NaturalQuestions (closed-book) - F1 ↑ [sort]	NaturalQuestions (open-book) - F1 ↑ [sort]	QuAC - F1 ↑ [sort]	HellaSwag - EM ↑ [sort]	OpenbookQA - EM ↑ [sort]	TruthfulQA - EM ↑ [sort]
Cohere Command beta (52.4B)	**0.93**	0.452	0.856	**0.752**	0.372	0.76	0.432	0.811	0.582	0.269
text-davinci-002	0.93	0.568	0.877	0.727	0.383	0.713	0.445	0.815	0.594	**0.61**
text-davinci-	0.898	**0.569**	0.881	0.727	0.406	0.77	0.525	0.822	0.646	0.593

HELM results webpage

Disclaimer: metrics may not be up-to-date. Check https://crfm.stanford.edu/helm/latest for the latest.

Key takeaways

LLM fine-tuning process

LLM fine-tuning

Training dataset

Prompt:

```
Classify this review:
I loved this DVD!

Sentiment:
```

Model

Pre-trained LLM

LLM completion:

Label:

Loss: Cross-

LLM fine-tuning process

LLM fine-tuning

Training dataset

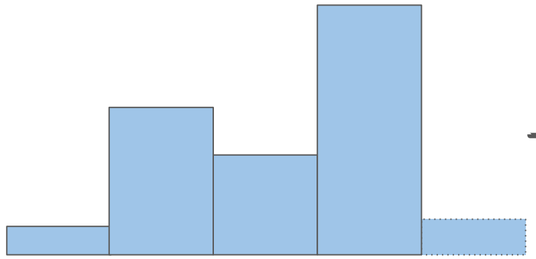
Prompt:

```
Classify this review:
I loved this DVD!

Sentiment:
```

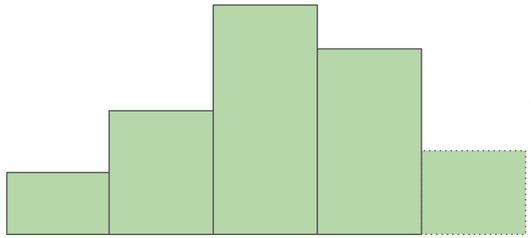
Model

Updated LLM

LLM completion:



Label:

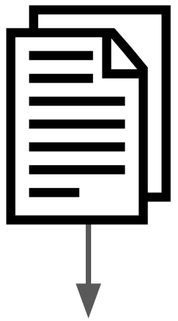


Loss: Cross-Entropy

LLM fine-tuning process

LLM fine-tuning

Prepared instruction dataset



Prompt:

```
Classify this review:
I loved this DVD!

Sentiment:
```

Model

Pre-trained LLM

LLM completion:

```
Classify this review:
I loved this DVD!

Sentiment: Neutral
```

Label:

```
Classify this review:
I loved this DVD!

Sentiment: Positive
```

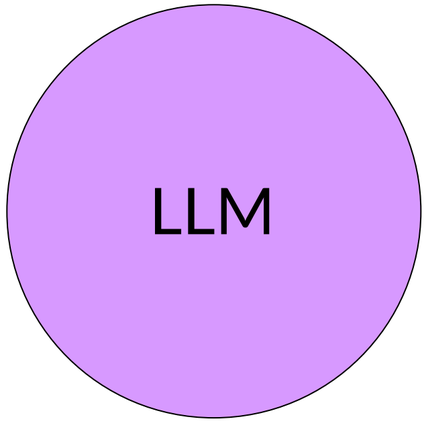
"Parameter Efficient Fine-Tuning (PEFT) updates only a small subset of parameters. This helps prevent catastrophic forgetting."

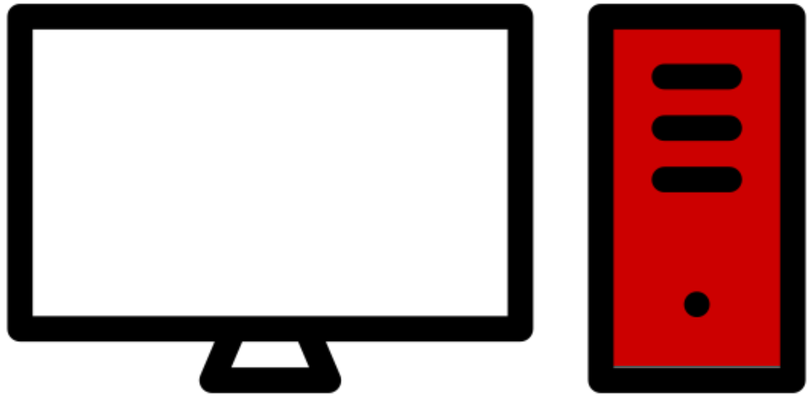
Parameter-efficient Fine-tuning (PEFT)

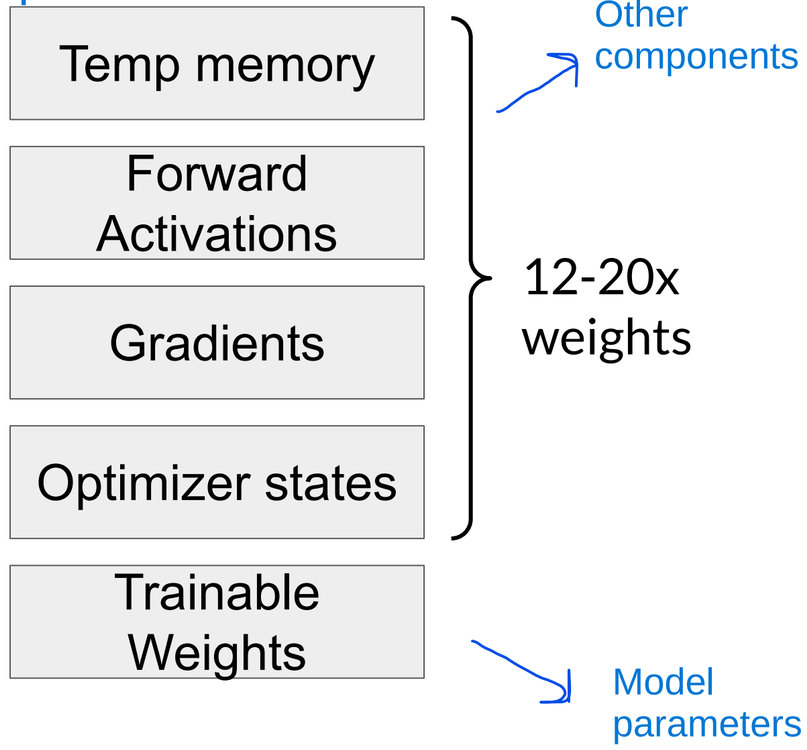
Earlier we talked about Full fine tuning LLM where all the parameters (model learning weights) are updated in the process. Practically if we see this will take a lot of compute and memory. In addition to this one needs to allocate extra memory for other components such as Temp memory, Forward Activations, Gradients, Optimizer. States. These additional components can occupy up to 20 times more memory as compared to trainable weights(learning parameters). A normal consumer's system is not fitted with Hardwares to accommodate such huge compute and memory requirements

Full fine-tuning of large LLMs is challenging

So, a concept of PEFT came into picture. Parameter Efficient Fine-Tuning (PEFT) updates only a small subset of parameters. This helps prevent catastrophic forgetting(As all the parameters are not updated hence, model still retain some knowledge or deep contextual understand of other tasks as well) and also helps to overcome compute and memory issue as stated above







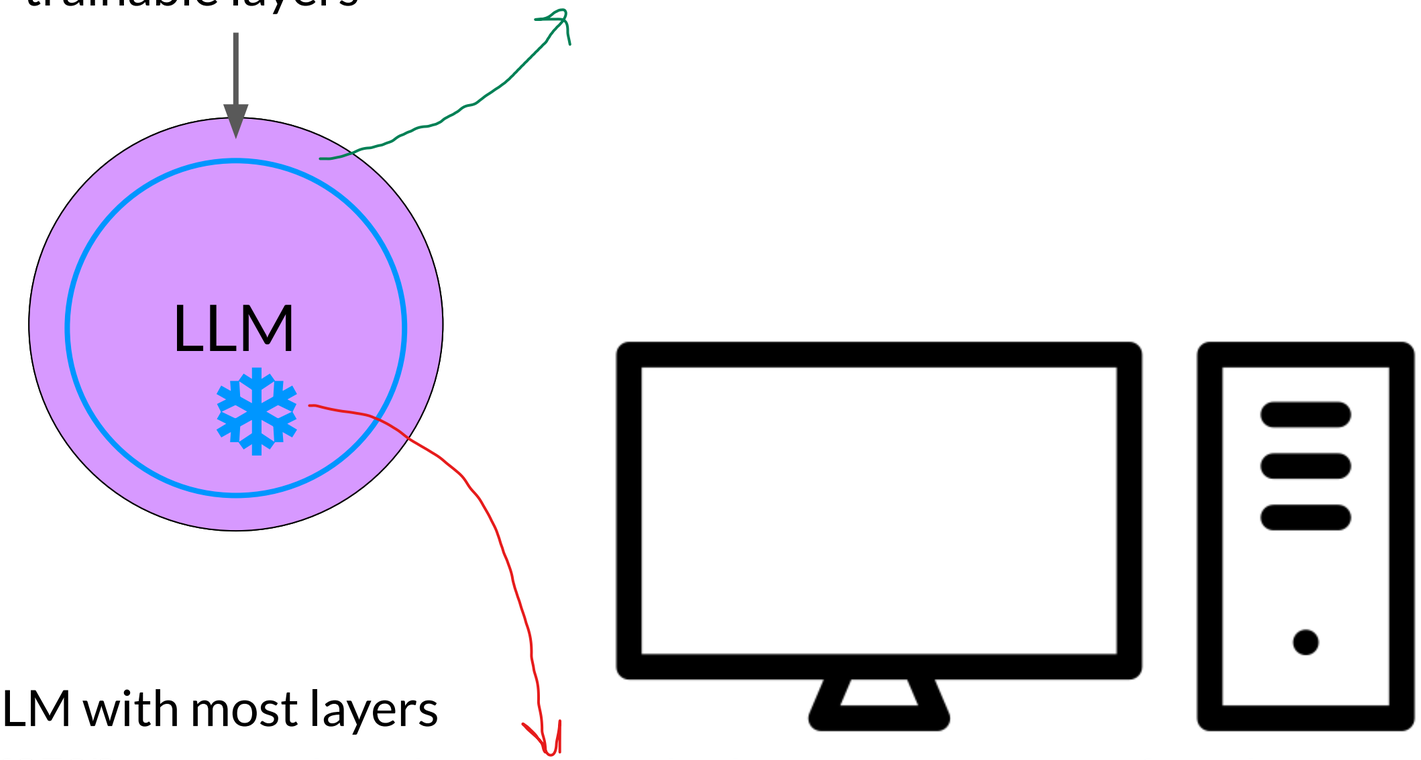
Please note the memory allocation size of other components is dependent on memory allocation size of Model parameters. Which imples that if we are reducing the memory footprint of model parameters by taking small subset for of them during training then we are also reducing memory allocation size for both Model parameters and Other components as well.

Parameter efficient fine-tuning (PEFT)

This slide demonstrates the Approach 1 for applying PEFT

Small number of trainable layers

Observe here that in PEFT we are using a small subset or layer of Model parameters or trainable weights



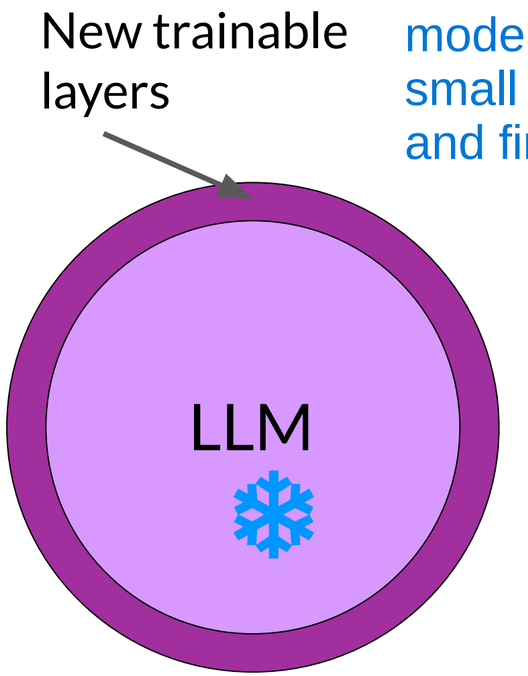
LLM with most layers frozen

Also observe that in the process we are freezing most of the trainable layers meaning this will remain as it is and no update will be made to these parameters during model fine tuning process for some tasks. Since frozen layer size is comparatively large to non frozen trainable layer, this helps drastically in eliminating the Catastrophic forgetting

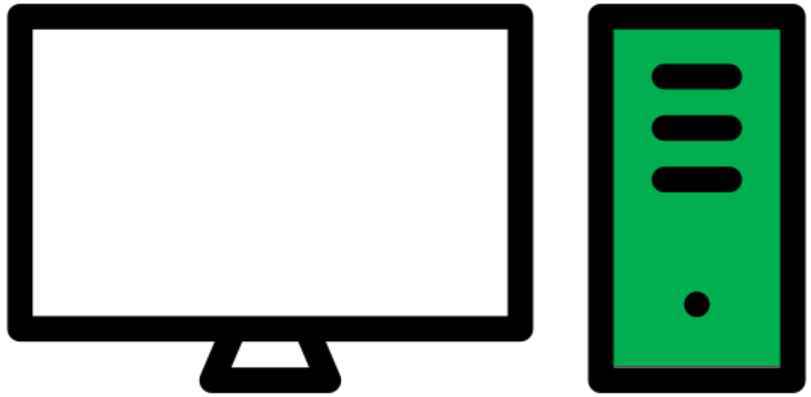
Parameter efficient fine-tuning (PEFT)

This slide demonstrates the Approach 2 for applying PEFT

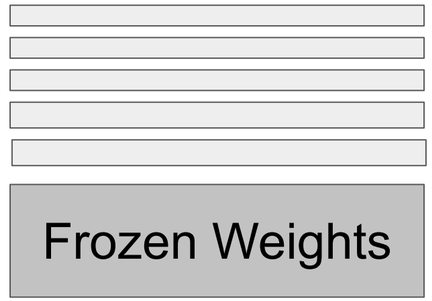
Other techniques don't touch the original model weights at all, and instead add a small number of new parameters or layers and fine-tune only the new components.



LLM with additional layers for PEFT



Less prone to catastrophic forgetting

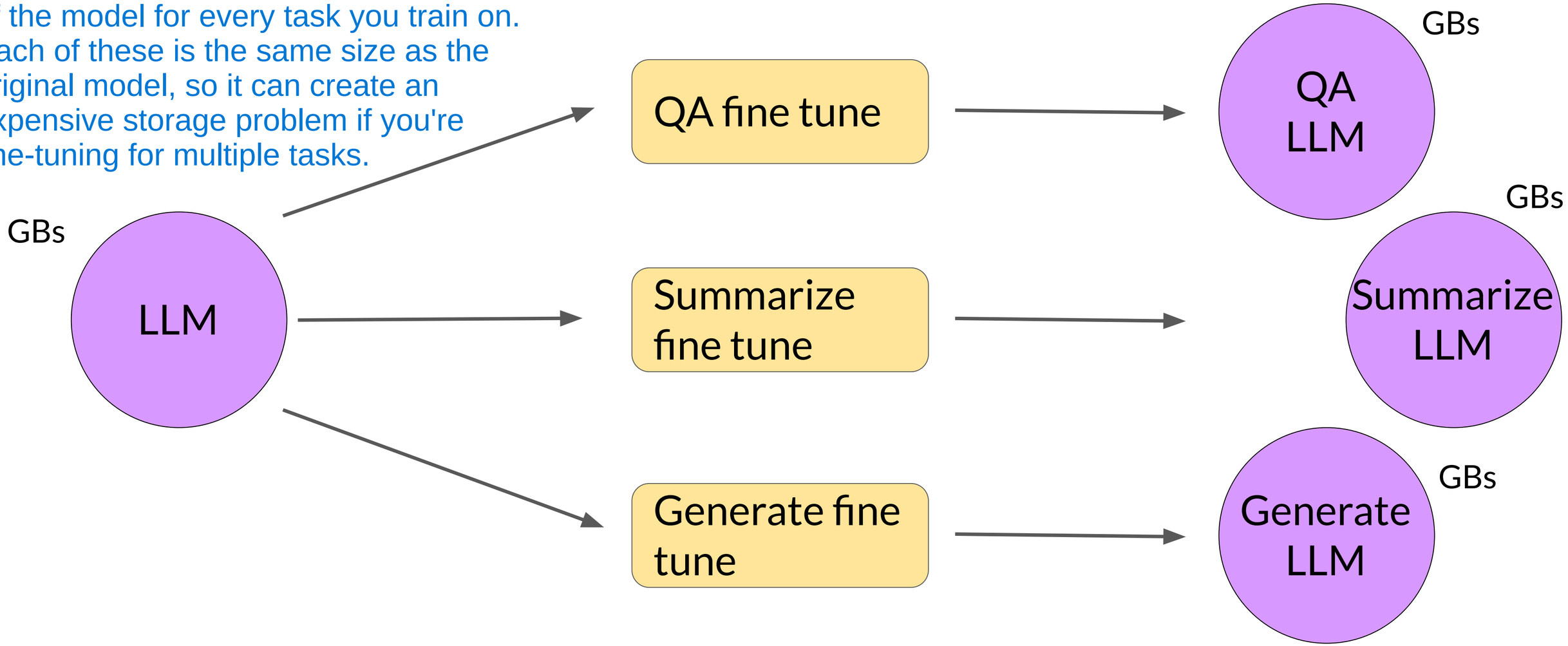


Other components

Trainable weights

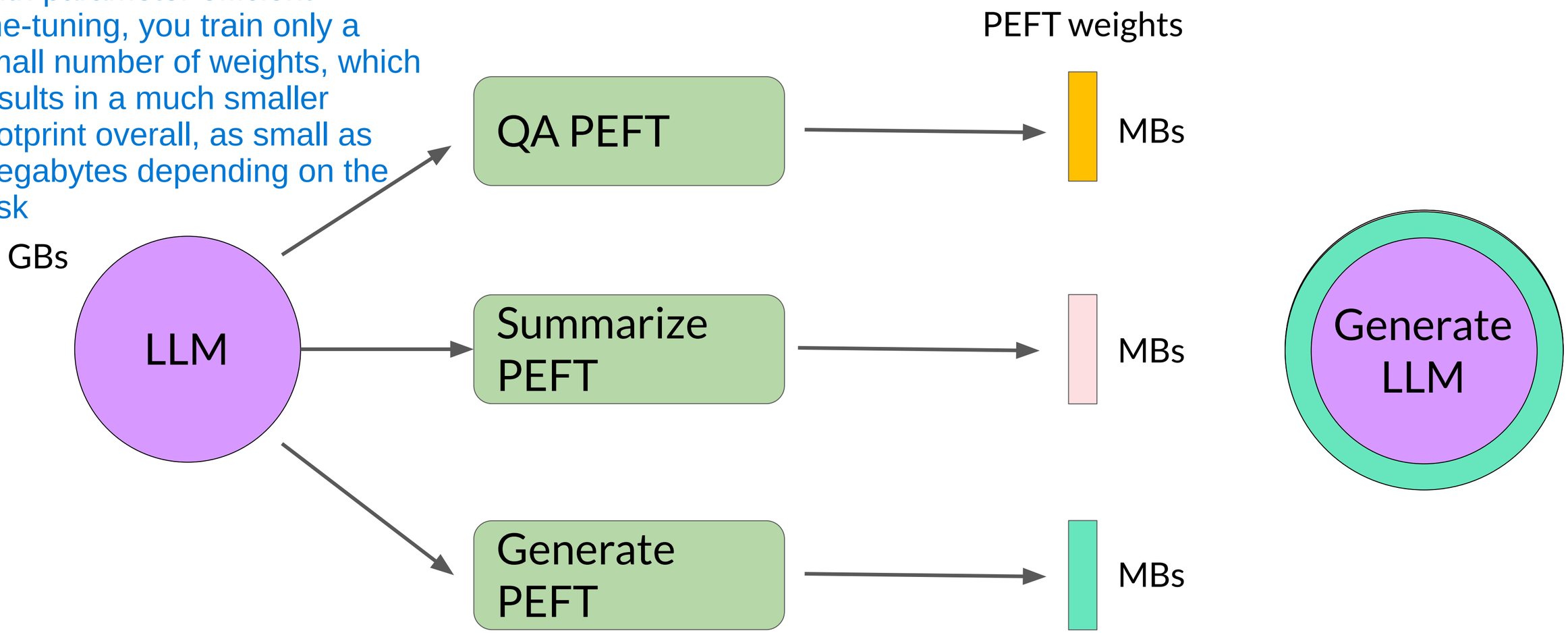
Full fine-tuning creates full copy of original LLM per task

Full fine-tuning results in a new version of the model for every task you train on. Each of these is the same size as the original model, so it can create an expensive storage problem if you're fine-tuning for multiple tasks.



PEFT fine-tuning saves space and is flexible

With parameter efficient fine-tuning, you train only a small number of weights, which results in a much smaller footprint overall, as small as megabytes depending on the task

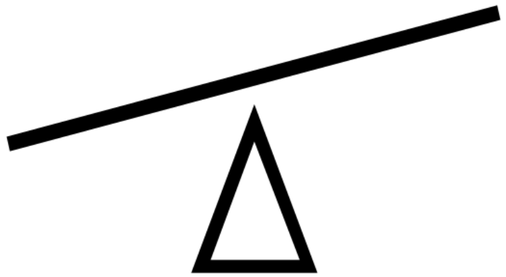


PEFT Trade-offs

There are several methods you can use for parameter efficient fine-tuning, each with trade-offs on parameter efficiency, memory efficiency, training speed, model quality, and inference costs

Parameter Efficiency

Memory Efficiency



Training Speed

Model Performance

Inference Costs

PEFT methods

Selective, Reparametrize and Additive are 3 main classes of PEFT method

Selective

Select subset of initial LLM parameters to fine-tune

Reparameterization

Reparameterize model weights using a low-rank representation

An example of this is LoRA which is explained in the next lecture where we will convert the original parameters into low rank parameters

Source: Lialin et al. 2023, “Scaling Down to Scale Up: A Guide to Parameter-Efficient Fine-Tuning”,

PEFT methods

Lastly, additive methods carry out fine-tuning by keeping all of the original LLM weights frozen and introducing new trainable components. Here there are two main approaches. Adapter methods add new trainable layers to the architecture of the model, typically inside the encoder or decoder components after the attention or feed-forward layers. Soft prompt methods, on the other hand, keep the model architecture fixed and frozen, and focus on manipulating the input to achieve better performance. This can be done by adding trainable parameters to the prompt embeddings or keeping the input fixed and retraining the embedding weights. In this lesson, you'll take a look at a specific soft prompts technique called prompt tuning.

Selective

Select subset of initial LLM parameters to fine-tune

Reparameterization

Reparameterize model weights using a low-rank representation

LoRA

Additive

Add trainable layers or parameters to model

Adapters

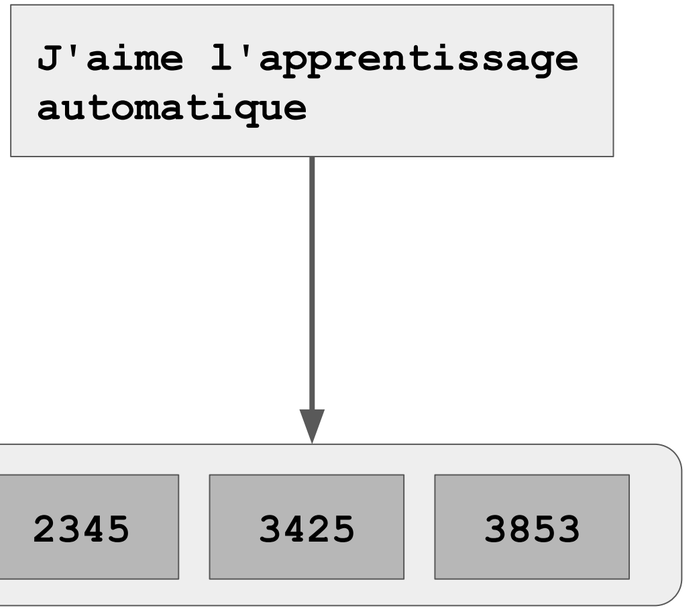
Soft Prompts

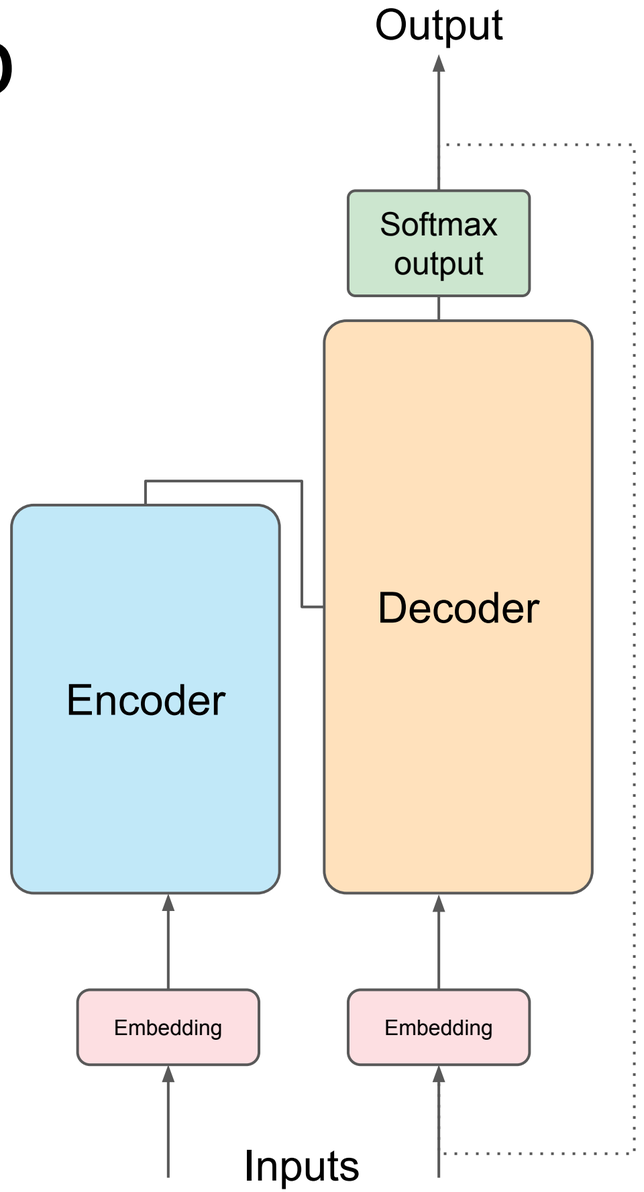
Prompt Tuning

Source: Lialin et al. 2023, “Scaling Down to Scale Up: A Guide to Parameter-Efficient Fine-Tuning”,

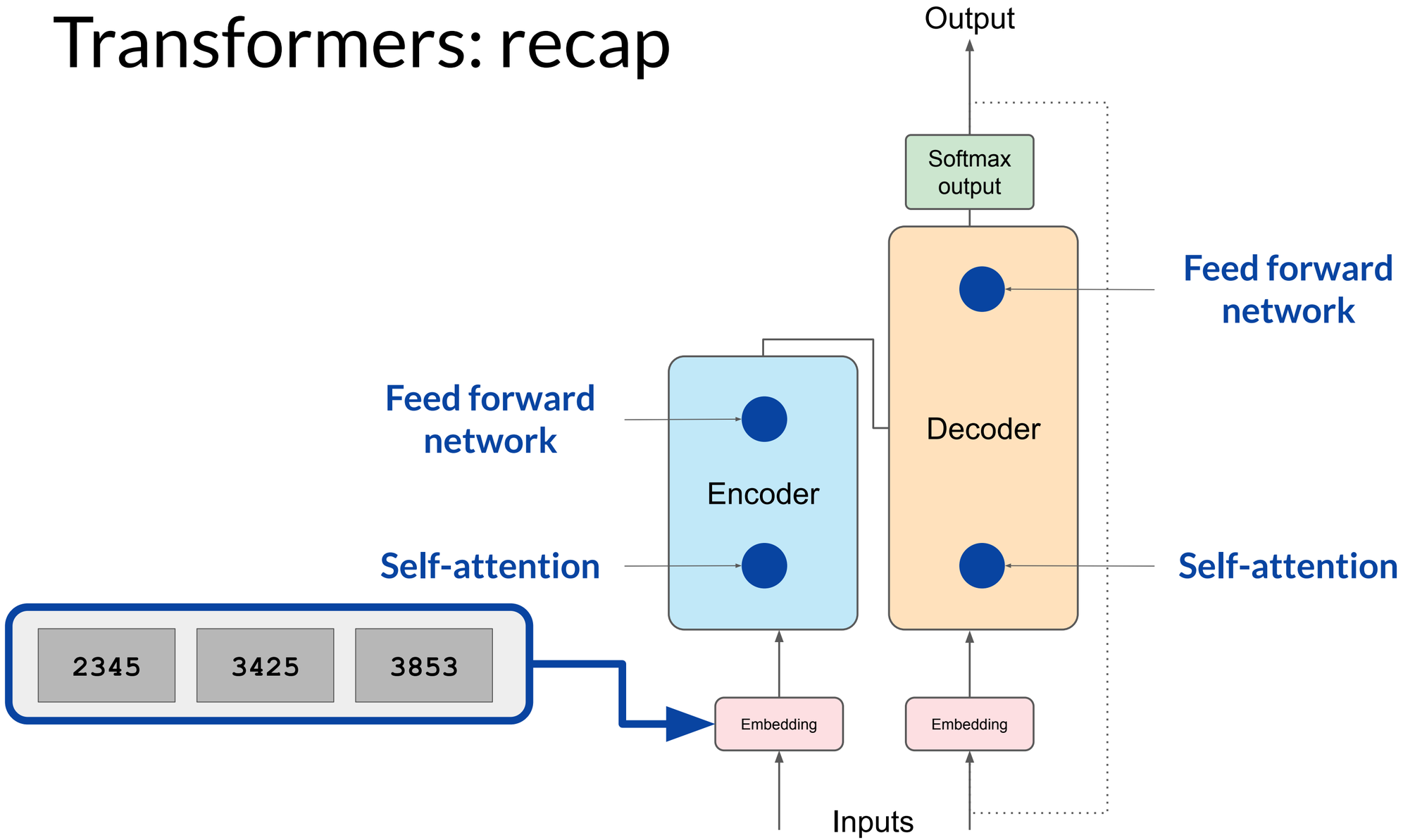
Low-Rank Adaptation of Large Language Models (LoRA)

Transformers: recap





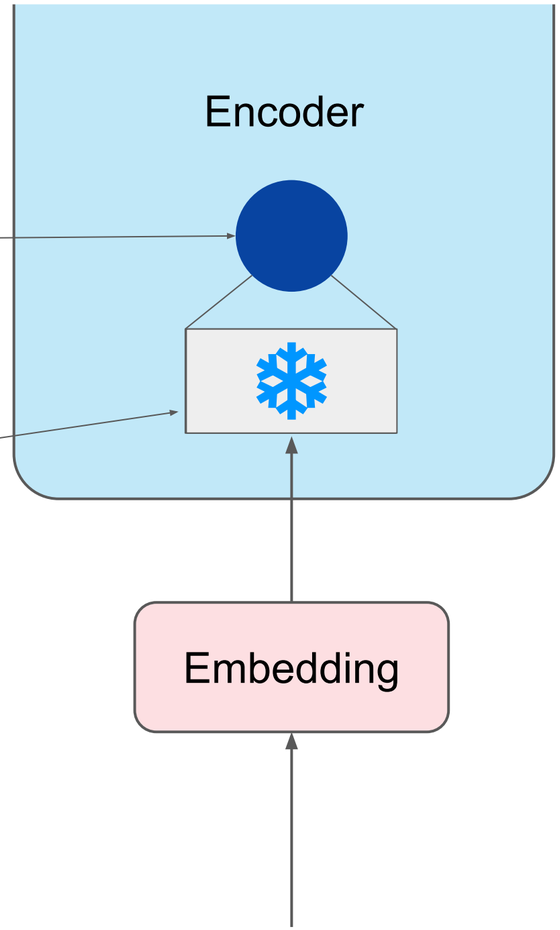
Transformers: recap



LoRA: Low Rank Adaption of LLMs

We can apply LoRA at any stage (like Feed forward network). But it is seen that if we are applying LoRA at the Vector embedding stage and then passing the updated weights using LoRA to the self-attention layer then this results in lot of saving in terms of compute and memory

Why are we applying LoRA at the vector embedding stage

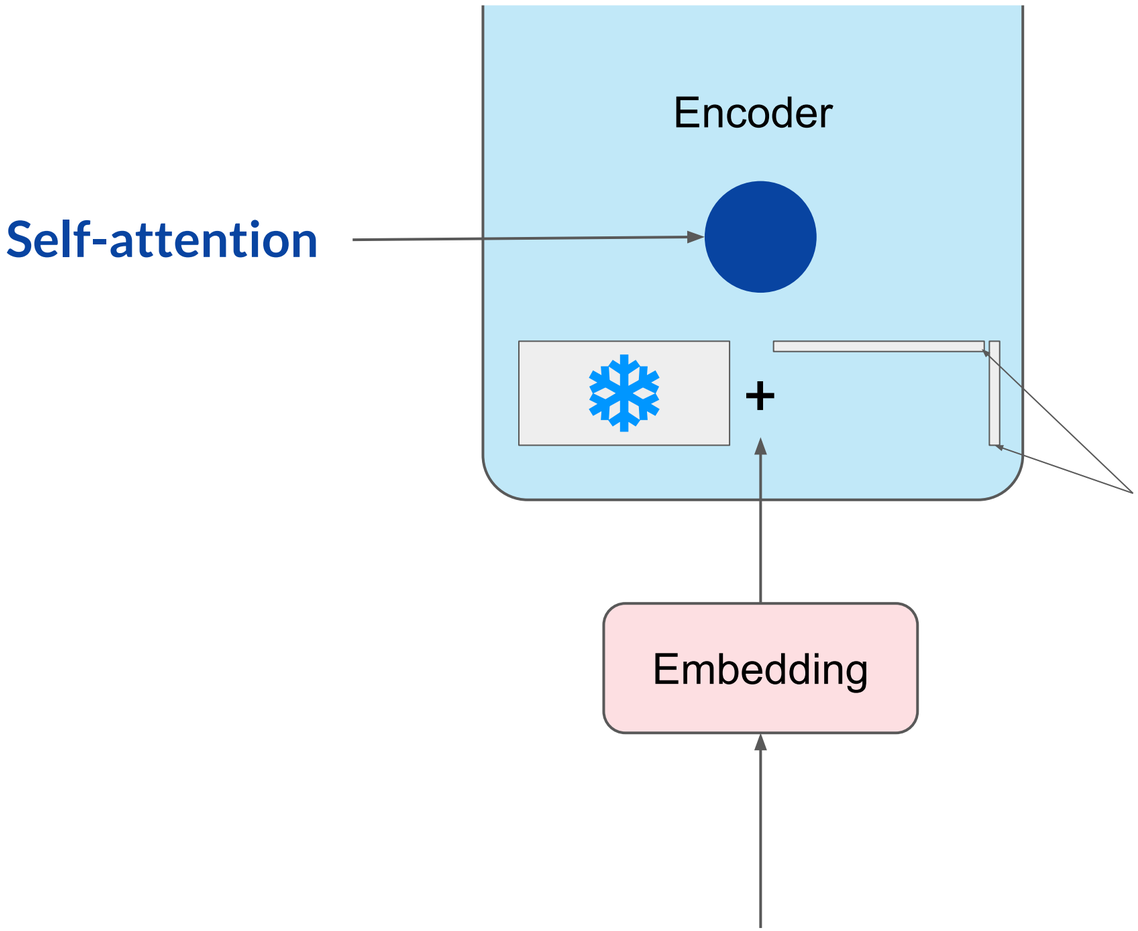


Self-attention

Weights applied to embedding vectors

1. Freeze most of the original LLM weights.

LoRA: Low Rank Adaption of LLMs

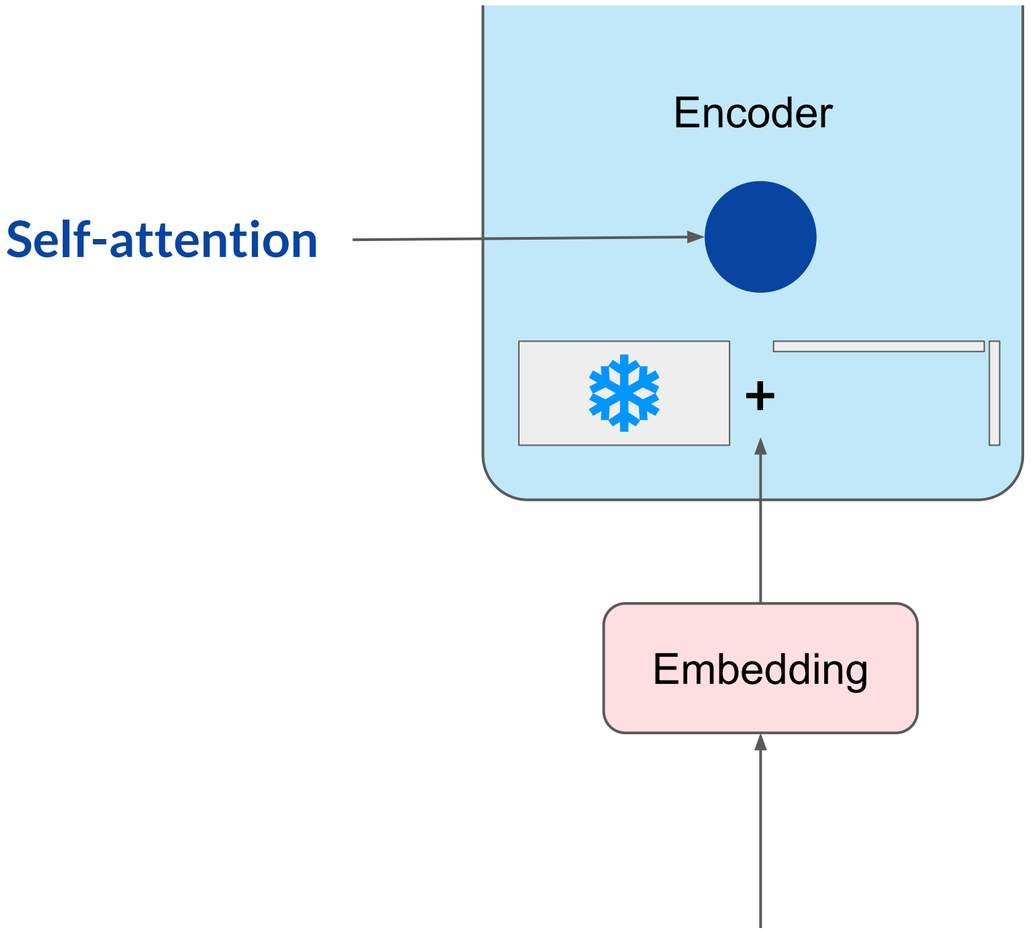


1. Freeze most of the original LLM weights.
2. Inject 2 **rank decomposition matrices** (A & B)
3. Train the weights of the smaller matrices

Rank *r* is small dimension, typically 4, 8 … 64

To understand rank and dimension of these low rank decomposition matrices see the Transformer example in the below following slides

LoRA: Low Rank Adaption of LLMs

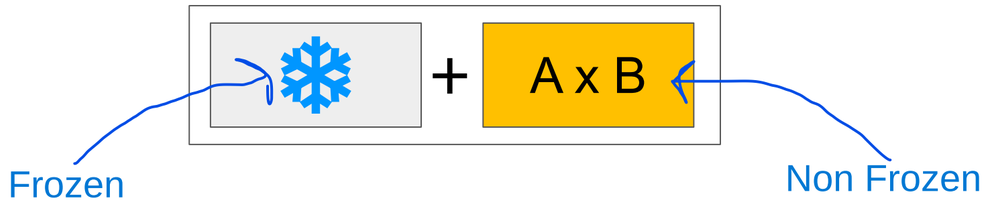


1. Freeze most of the original LLM weights.
2. Inject 2 **rank decomposition matrices**
3. Train the weights of the smaller matrices

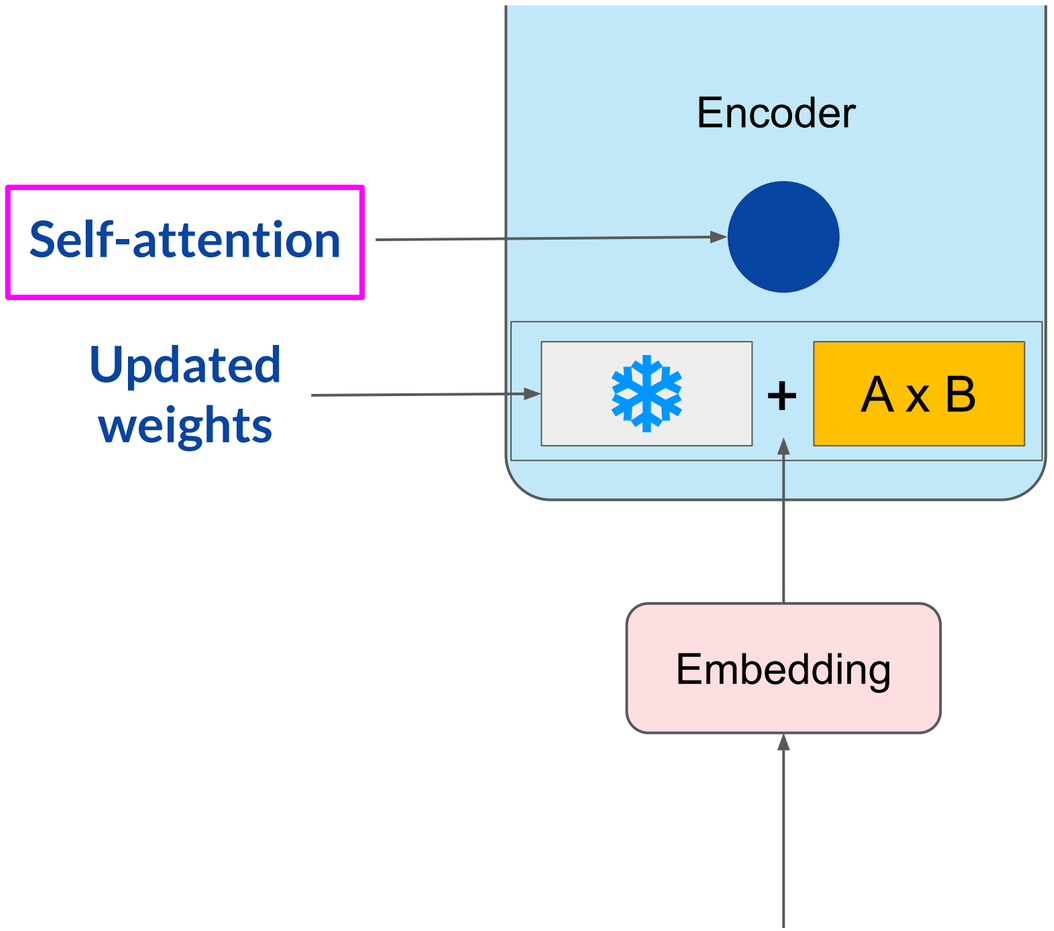
Steps to update model for inference

1. Matrix multiply the low rank matrices

2. Add to original weights



LoRA: Low Rank Adaption of LLMs

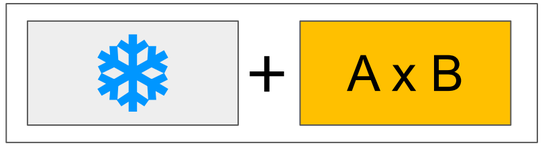


1. Freeze most of the original LLM weights.
2. Inject 2 **rank decomposition matrices**
3. Train the weights of the smaller matrices

Steps to update model for inference:

1. Matrix multiply the low rank matrices

2. Add to original weights



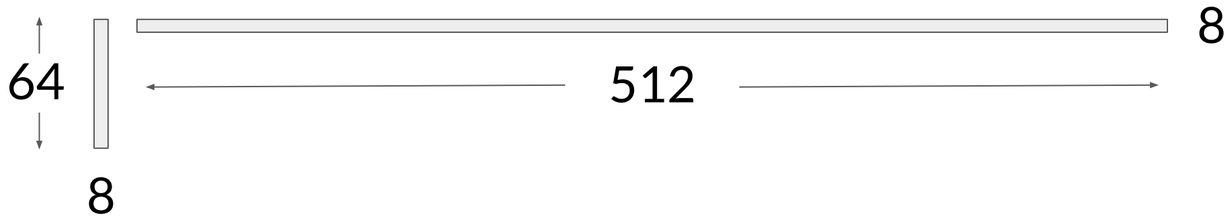
Concrete example using base Transformer as reference

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

- Transformer weights have dimensions d x k = 512 x 64
- So 512 x 64 = 32,768 trainable parameters

In LoRA with rank r = 8:

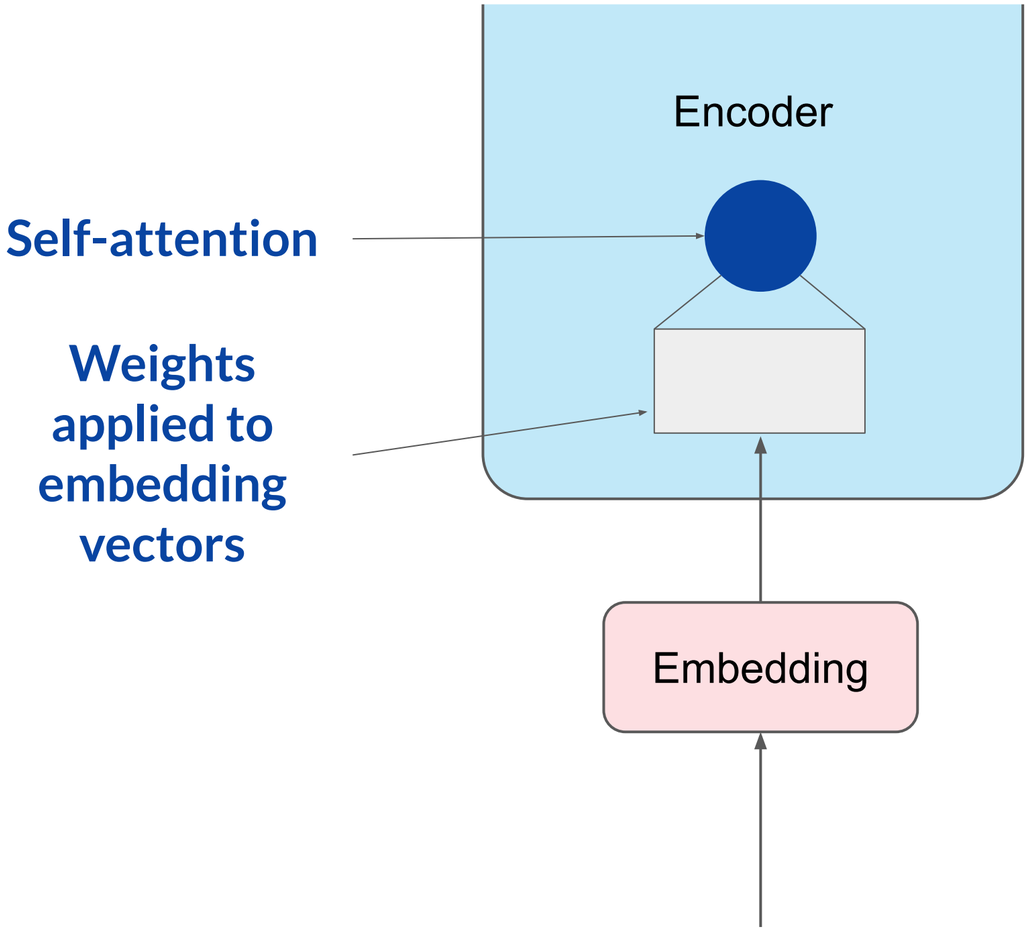
- A has dimensions r x k = 8 x 64 = 512 parameters
- B has dimension d x r = 512 x 8 = 4,096 trainable parameters



86% reduction in parameters to train! (32,768 parameters ---> 4608(512+4096))

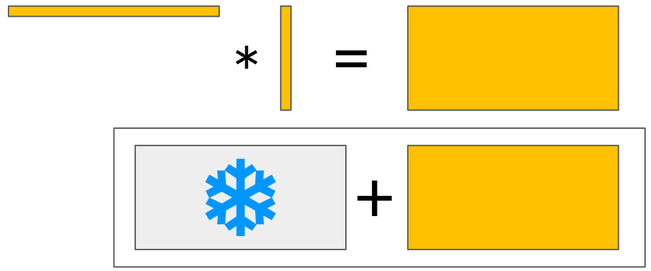
LoRA: Low Rank Adaption of LLMs

Using LoRA to train across different task



1. Train different rank decomposition matrices for different tasks
2. Update weights before inference

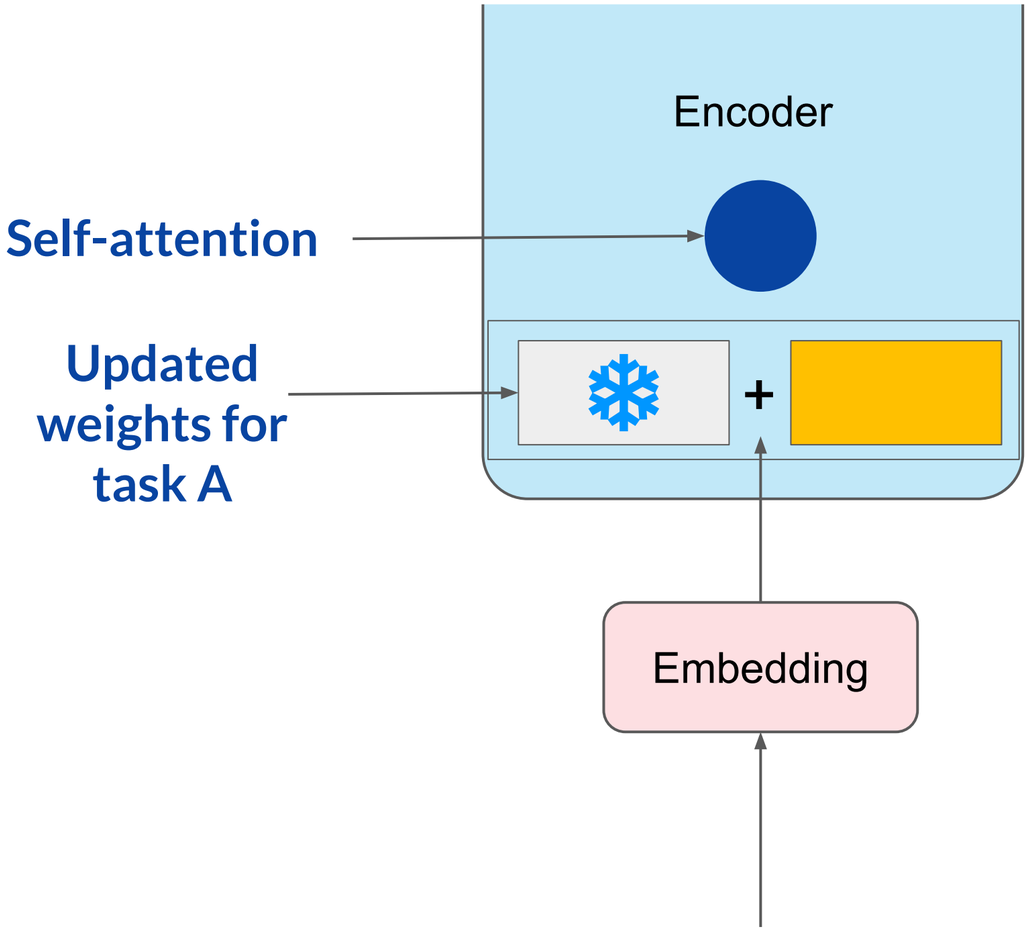
Task A



Frozen constant weight + (Non Frozen variable weight based on PEFT fine tuning for Task 1)

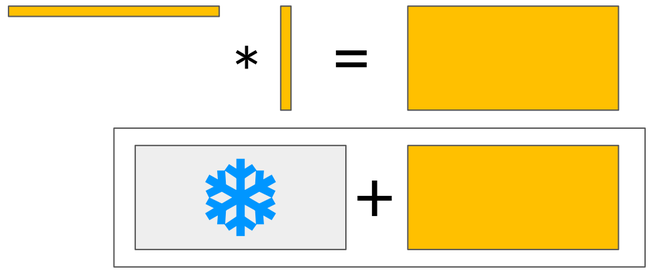
Meaning that keep switching on and off between different tasks as per need

LoRA: Low Rank Adaption of LLMs

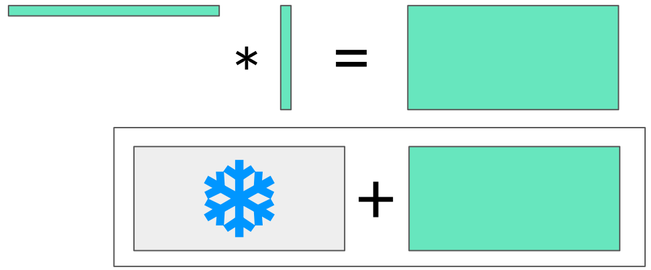


1. Train different rank decomposition matrices for different tasks
2. Update weights before inference

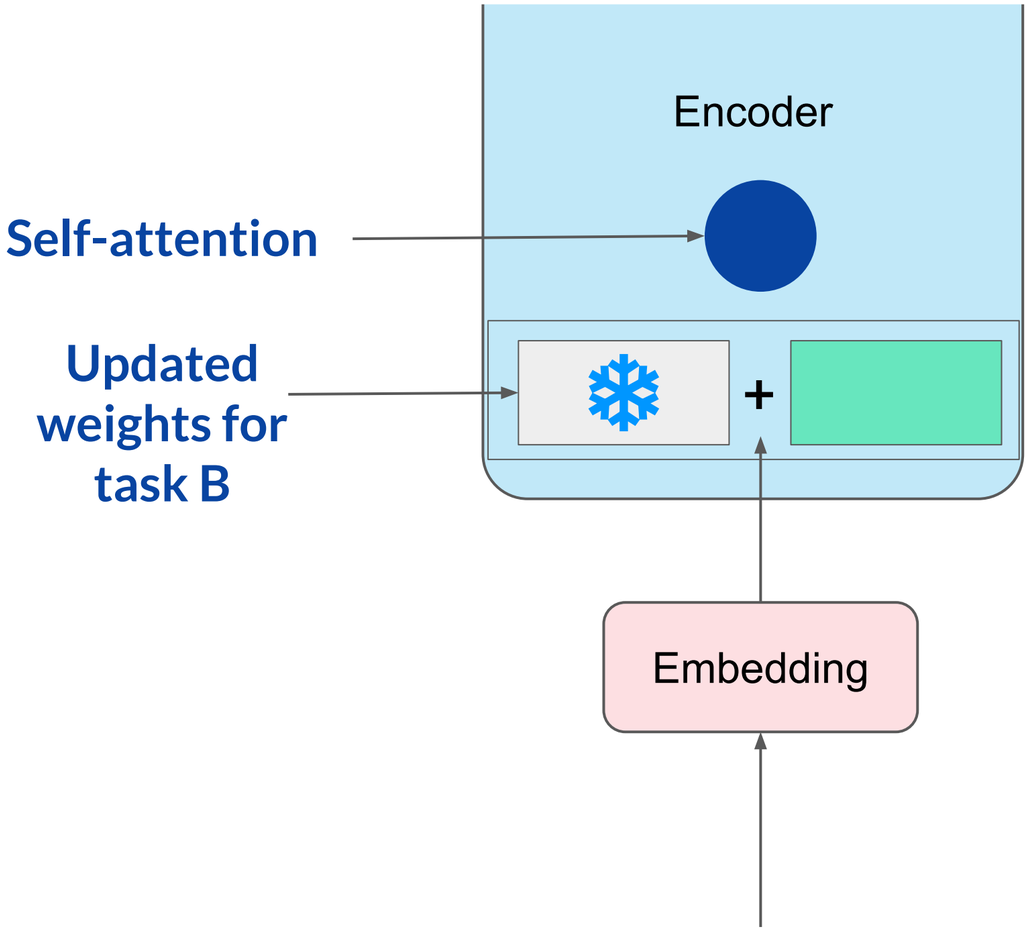
Task A



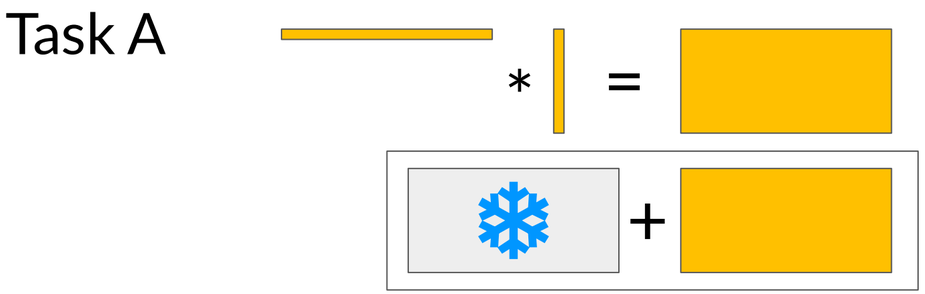
Task B

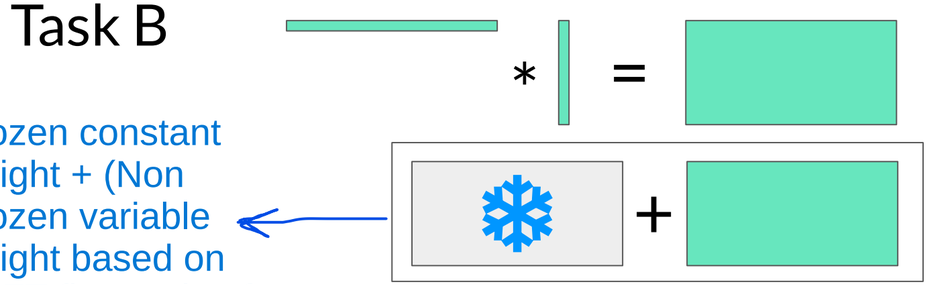


LoRA: Low Rank Adaption of LLMs



1. Train different rank decomposition matrices for different tasks
2. Update weights before inference





Frozen constant weight + (Non Frozen variable weight based on PEFT fine tuning for Task 2)

Sample ROUGE metrics for full vs. LoRA fine-tuning

Base model ROUGE

Full fine-tune ROUGE

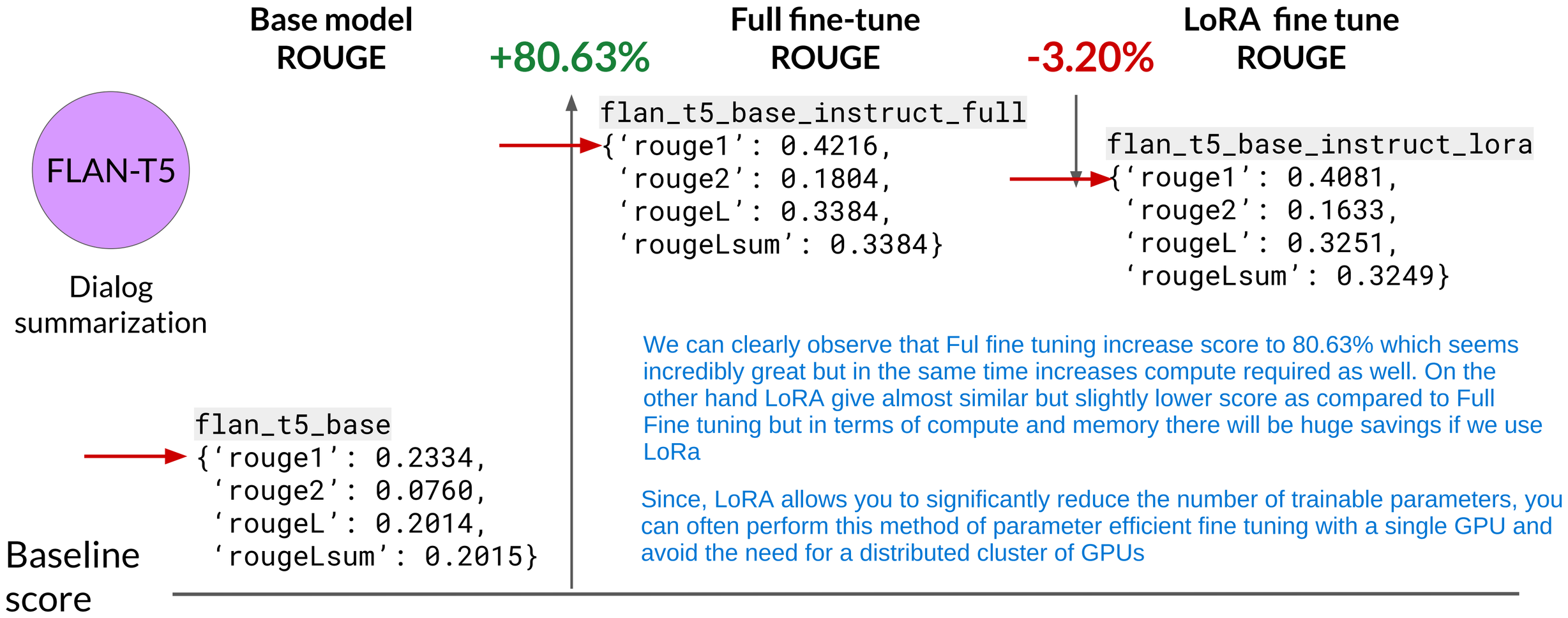
Dialog summarization

Highest score is the better

```
flan_t5_base
{'rouge1': 0.2334,
 'rouge2': 0.0760,
 'rougeL': 0.2014,
 'rougeLsum': 0.2015}
```

Baseline score

Sample ROUGE metrics for full vs. LoRA fine-tuning



We can clearly observe that Ful fine tuning increase score to 80.63% which seems incredibly great but in the same time increases compute required as well. On the other hand LoRA give almost similar but slightly lower score as compared to Full Fine tuning but in terms of compute and memory there will be huge savings if we use LoRa

Since, LoRA allows you to significantly reduce the number of trainable parameters, you can often perform this method of parameter efficient fine tuning with a single GPU and avoid the need for a distributed cluster of GPUs

Choosing the LoRA rank

Choosing a LoRA rank is still an active area of research

Loss value calculated using loss function

Rank r	val_loss	BLEU	NIST	METEOR	ROUGE_L	CIDEr
1	1.23	68.72	8.7215	0.4565	0.7052	2.4329
2	1.21	69.17	8.7413	0.4590	0.7052	2.4639
4	1.18	**70.38**	**8.8439**	**0.4689**	0.7186	**2.5349**
8	1.17	69.57	8.7457	0.4636	**0.7196**	2.5196
16	**1.16**	69.61	8.7483	0.4629	0.7177	2.4985
32	**1.16**	69.33	8.7736	0.4642	0.7105	2.5255
64	**1.16**	69.24	8.7174	0.4651	0.7180	2.5070
128	**1.16**	68.73	8.6718	0.4628	0.7127	2.5030
256	**1.16**	68.92	8.6982	0.4629	0.7128	2.5012
512	**1.16**	68.78	8.6857	0.4637	0.7128	2.5025
1024	1.17	69.37	8.7495	0.4659	0.7149	2.5090

- Effectiveness of higher rank appears to plateau
- Relationship between rank and dataset size needs more empirical data

The takeaway here is that ranks in the range of 4-32 can provide you with a good trade-off between reducing trainable parameters and preserving performance

Source: Hu et al. 2021, “LoRA: Low-Rank Adaptation of Large Language Models”

QLoRA: Quantized LoRA

- Introduces 4-bit NormalFloat (nf4) data type for 4-bit quantization
- Supports double-quantization to reduce memory ~0.4 bits per parameter (~3 GB for a 65B model)
- Unified GPU-CPU memory management reduces GPU memory usage
- LoRA adapters at every layer - not just attention layers
- Minimizes accuracy trade-off

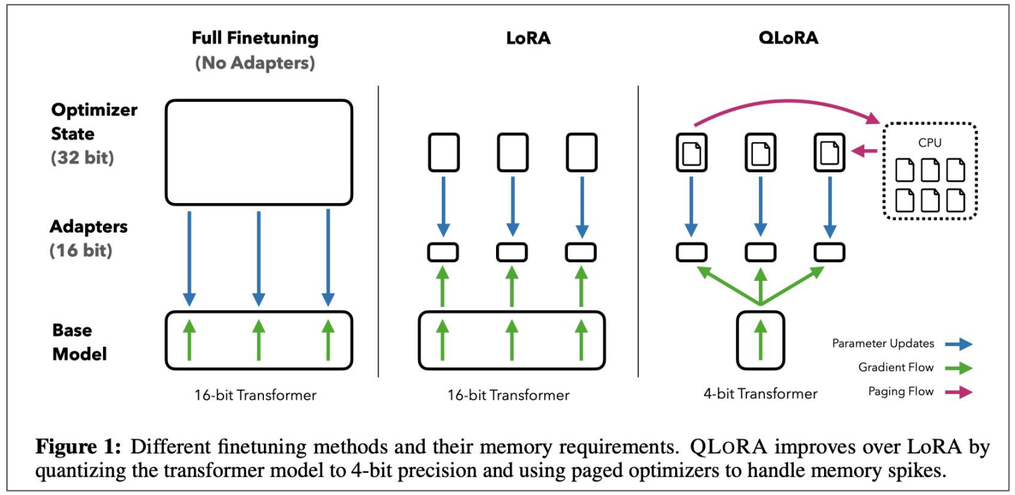


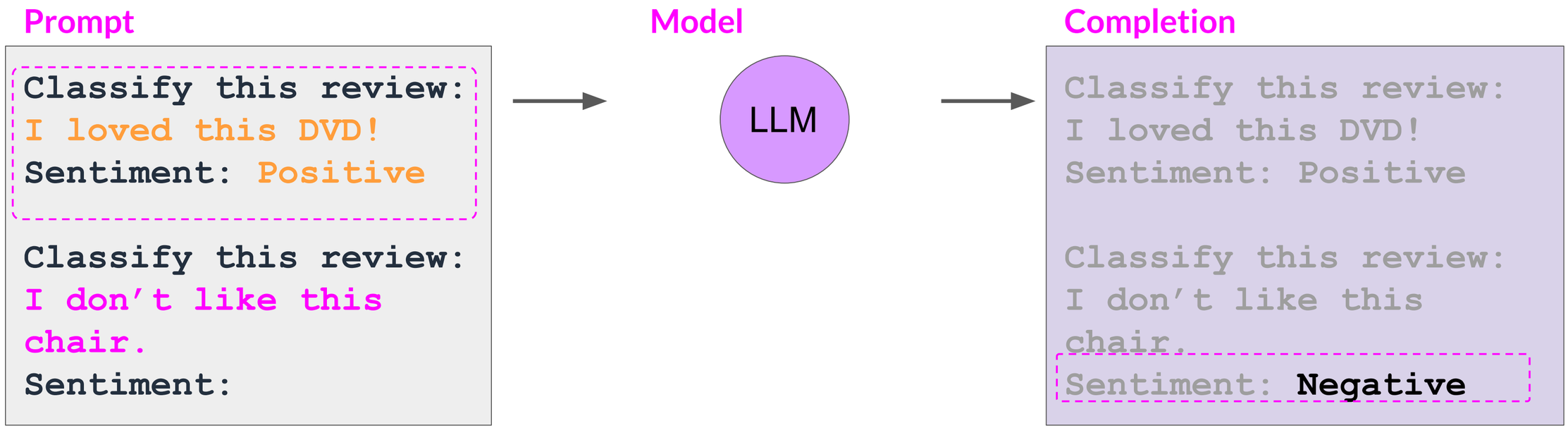
Figure 1: Different finetuning methods and their memory requirements. QLoRA improves over LoRA by quantizing the transformer model to 4-bit precision and using paged optimizers to handle memory spikes.

Source: Dettmers et al. 2023, "QLoRA: Efficient Finetuning of Quantized LLMs"

Prompt tuning with soft prompts

Unlike PEFT where we need to change the weight we may use additive approach of PEFT like Prompt tuning with soft prompts where we do not require to change the weights

Prompt tuning is **not** prompt engineering!

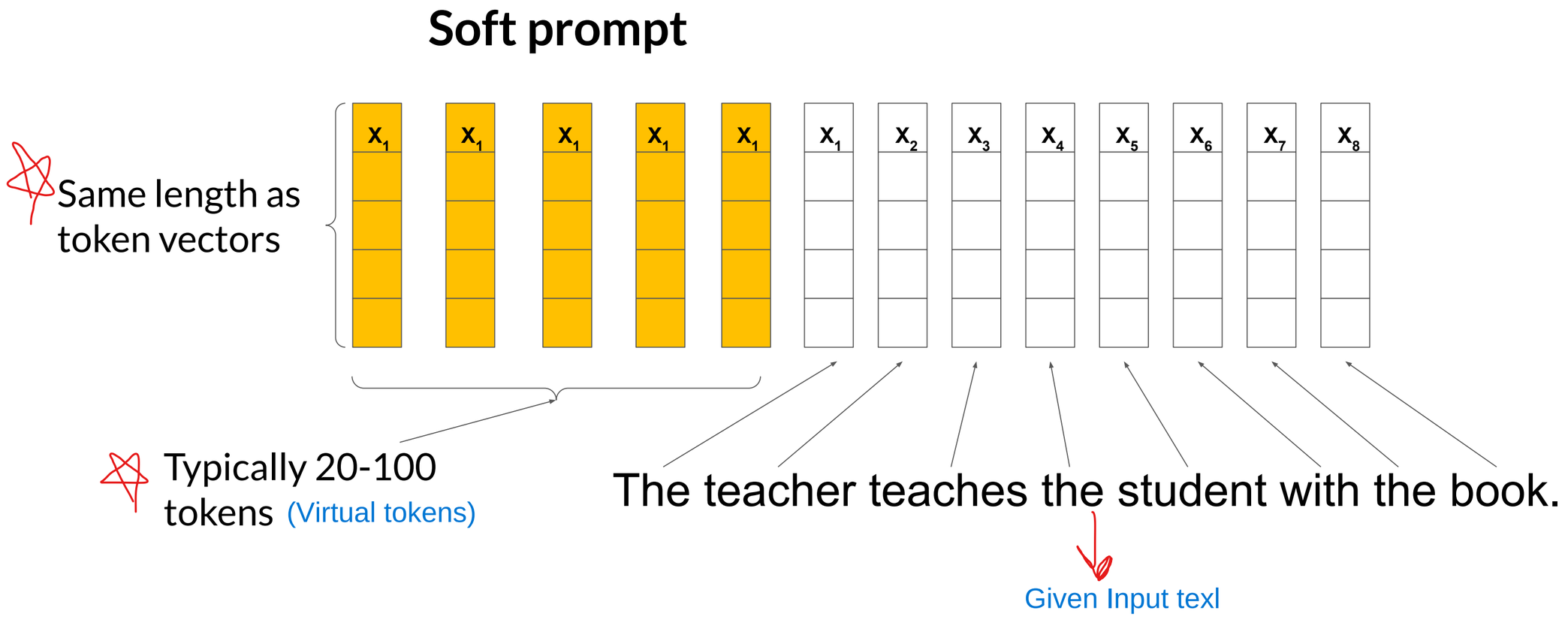


One-shot or Few-shot Inference

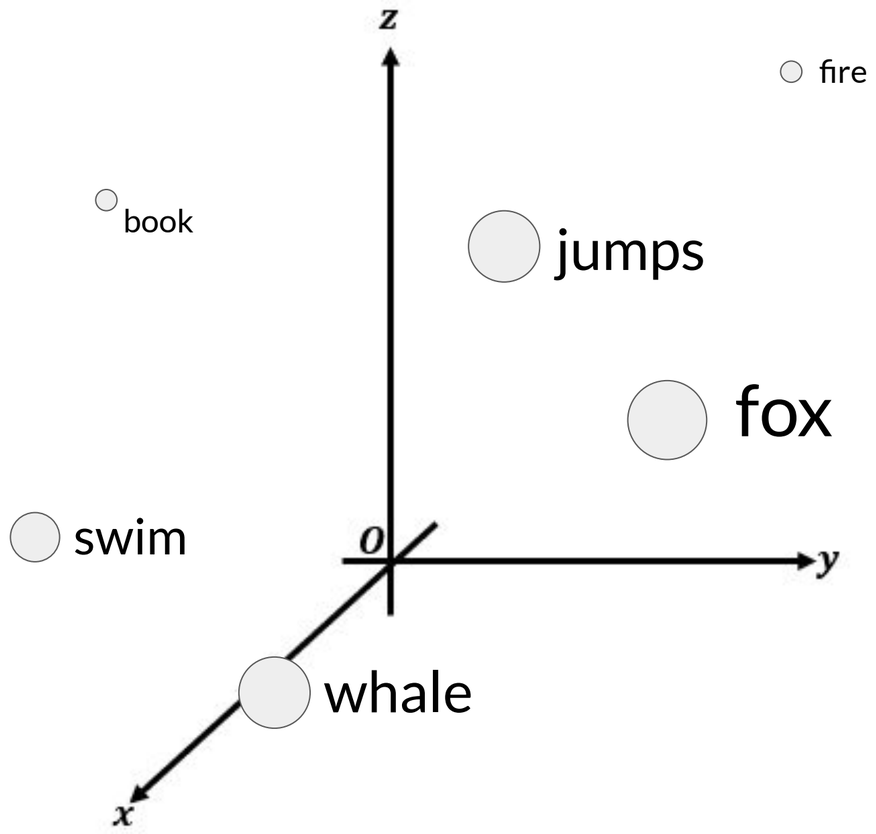
With prompt engineering, you work on the language of your prompt to get the completion you want. This could be as simple as trying different words or phrases or more complex, like including examples for one or Few-shot Inference. The goal is to help the model understand the nature of the task you're asking it to carry out and to generate a better completion. However, there are some limitations to prompt engineering, as it can require a lot of manual effort to write and try different prompts. You're also limited by the length of the context window, and at the end of the day, you may still not achieve the performance you need for your task

Prompt tuning adds trainable “soft prompt” to inputs

With prompt tuning, you add additional trainable tokens to your prompt and leave it up to the supervised learning process to determine their optimal values. The set of trainable tokens is called a soft prompt which is then prepended or passed to the vector embedding section that represents the resultant input text



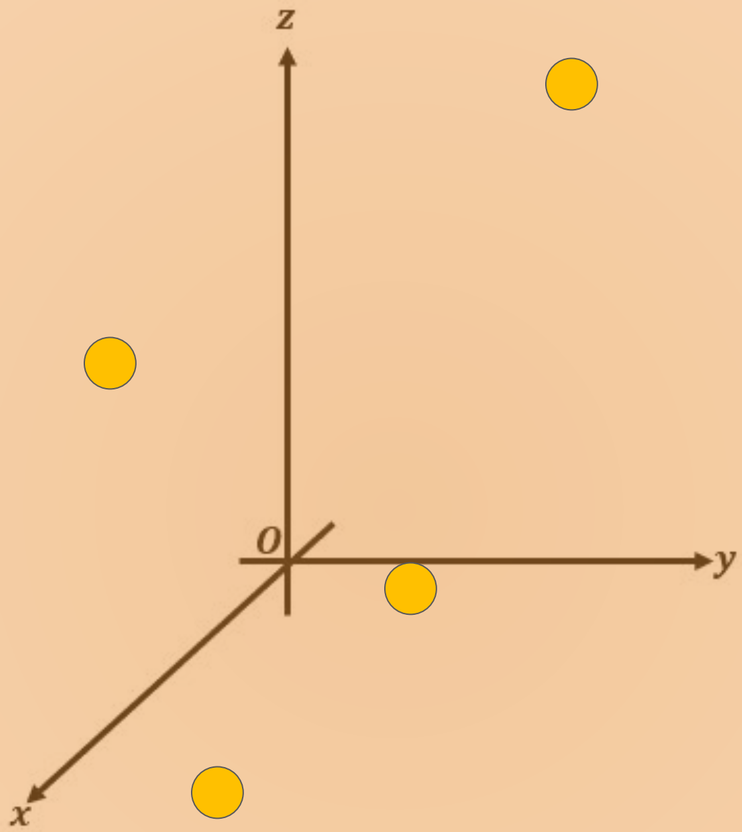
Soft prompts



The tokens that represent natural language are hard in the sense that they each correspond to a fixed location in the embedding vector space.

Embeddings of each token exist at unique point in multi-dimensional space

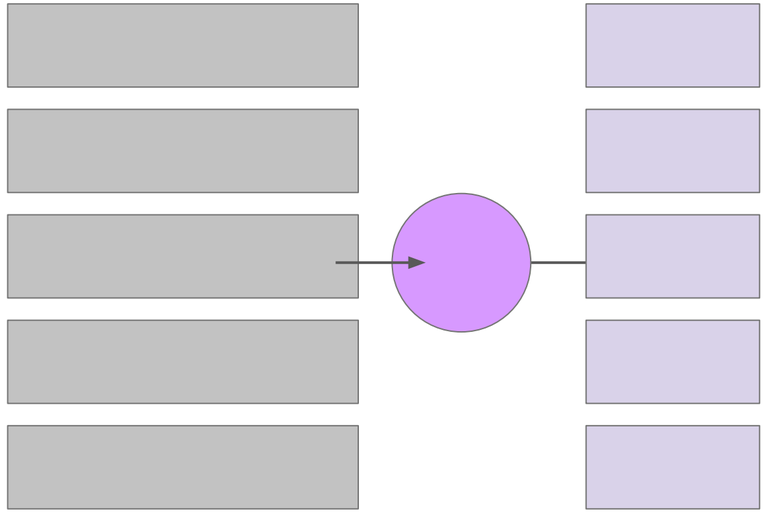
Soft prompts



However, the soft prompts are not fixed discrete words of natural language. Instead, you can think of them as virtual tokens that can take on any value within the continuous multidimensional embedding space. And through supervised learning, the model learns the values for these virtual tokens that maximize performance for a given task

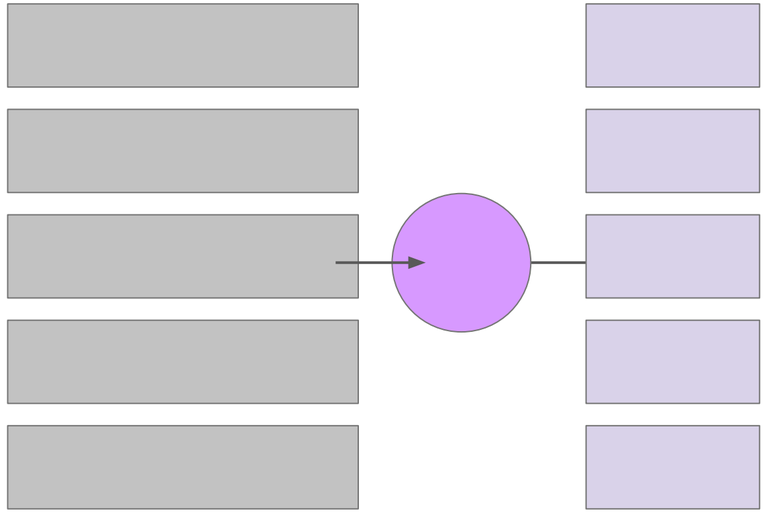
Full Fine-tuning vs prompt tuning

Weights of model updated during training



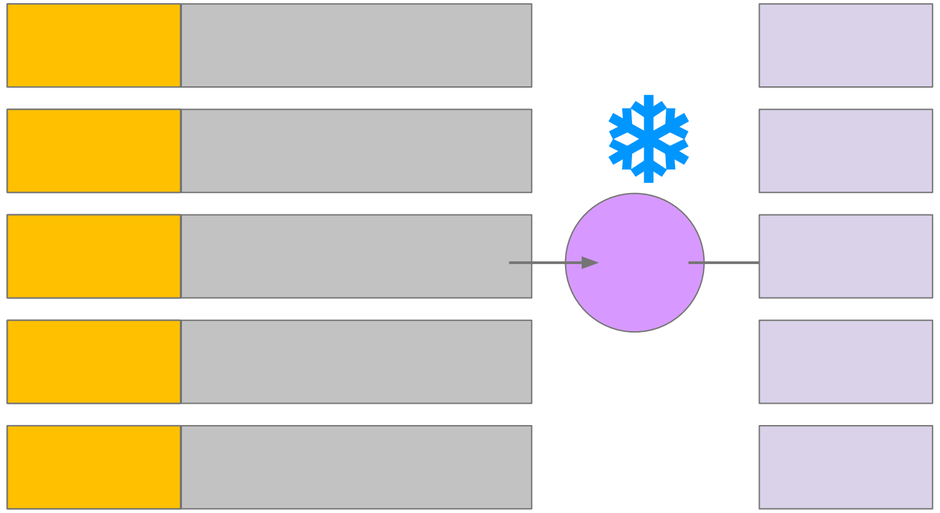
Full Fine-tuning vs prompt tuning

Weights of model updated during training



Millions to Billions of parameter updated

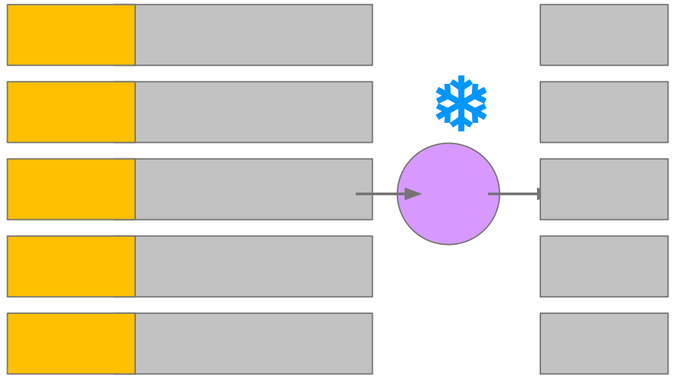
Weights of model frozen and soft prompt trained



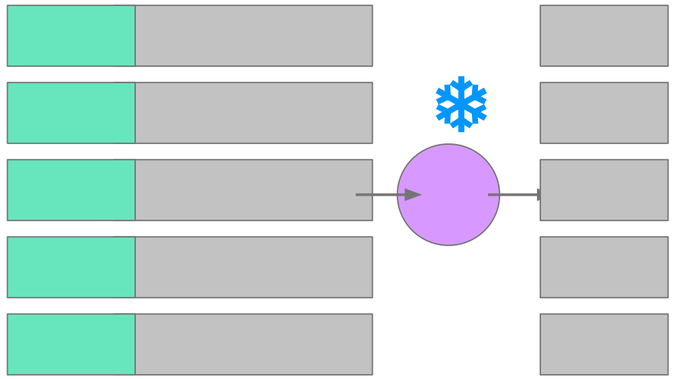
10K - 100K of parameters updated

Prompt tuning for multiple tasks

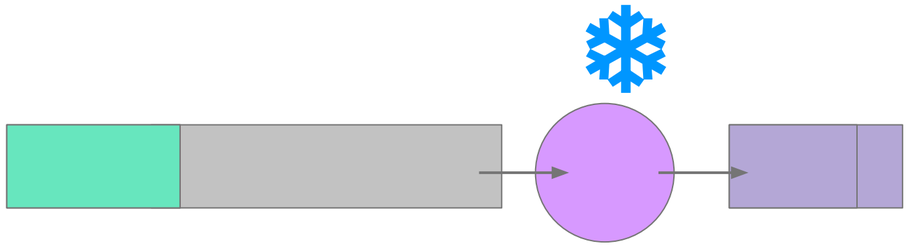
Task A



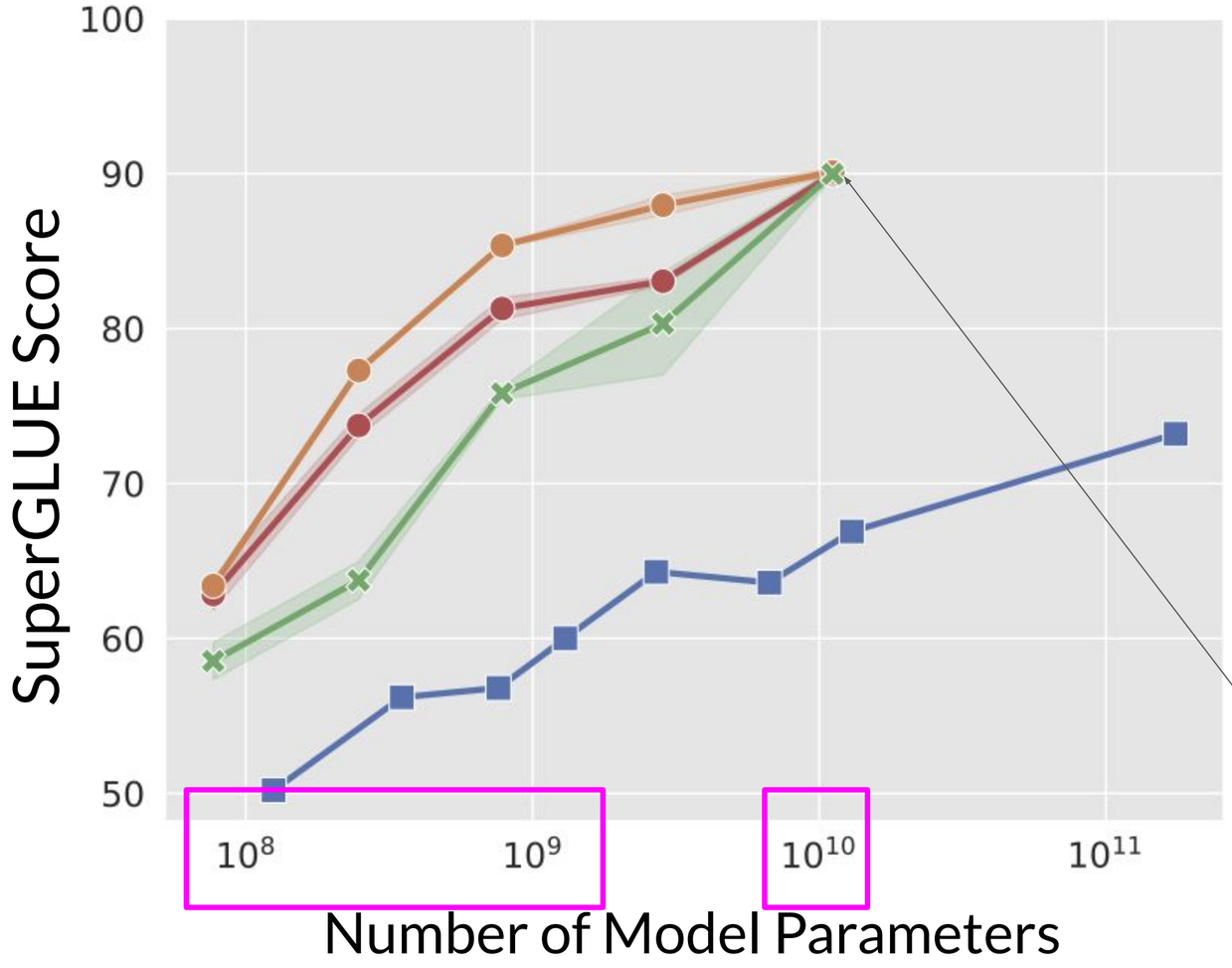
Task B



Switch out soft prompt at inference time to change task!



Performance of prompt tuning



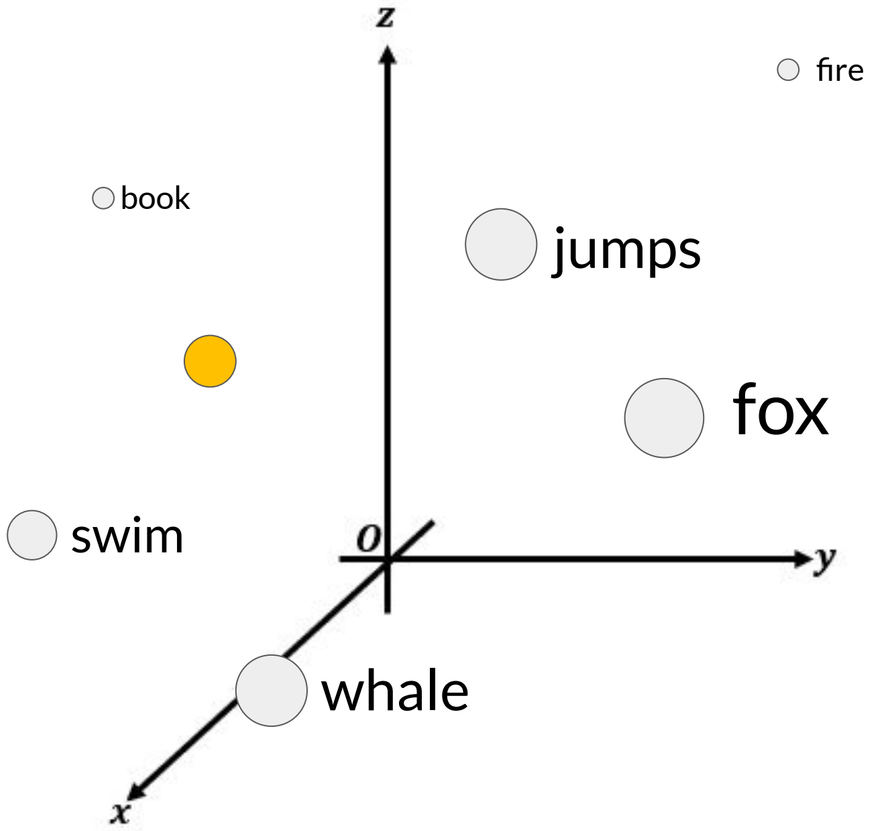
- Full Fine-tuning
- Multi-task Fine-tuning
- Prompt tuning
- Prompt engineering

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”

Interpretability of soft prompts

One potential issue to consider is the interpretability of learned virtual tokens. Remember, because the soft prompt tokens can take any value within the continuous embedding vector space. The trained tokens don't correspond to any known token, word, or phrase in the vocabulary of the LLM



Trained soft-prompt embedding does not correspond to a known token…

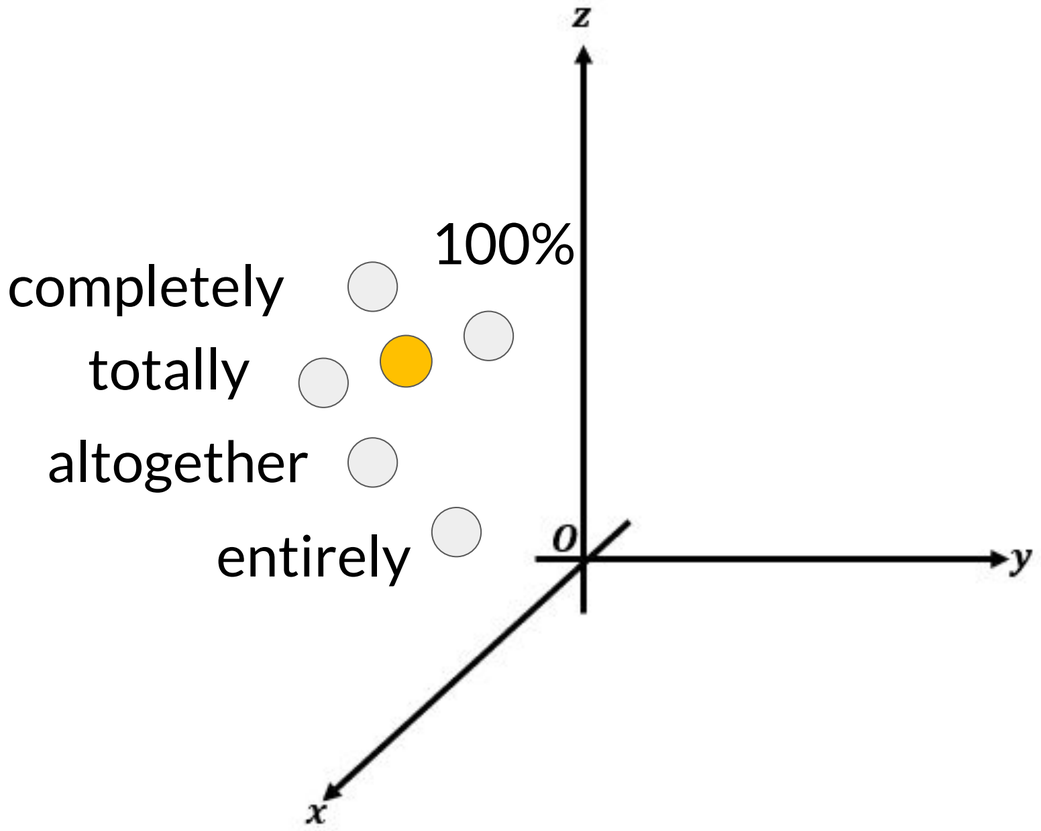
Interpretability of soft prompts

PEFT overcomes following over Full Fine tuning:
- Catastrophic forgetting

- Storage requirements

- Computational constraints

- Not model performance. Full fine-tuned still gives better performance over PEFT. But we have seen PEFT's performance is not very less



Just observe the diagram and the text written below will be very clear

...but nearest neighbors form a semantic group with similar meanings.

PEFT methods summary

Most important takeaway of PEFT

Both methods enable you to fine tune models with the potential for improved performance on your tasks while using much less compute than full fine tuning methods. LoRA is broadly used in practice because of the comparable performance to full fine tuning for many tasks and data sets, and you'll get to try it out for yourself in this week's lab

Selective

Select subset of initial LLM parameters to fine-tune

Reparameterization

Reparameterize model weights using a low-rank representation

LoRA

Additive

Add trainable layers or parameters to model

Adapters

Soft Prompts

Prompt Tuning