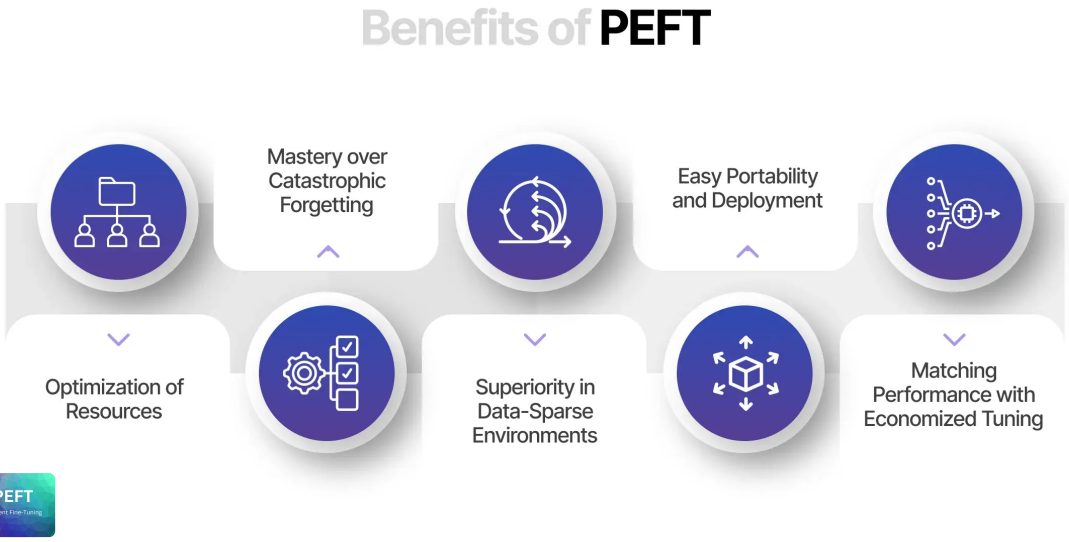


Parameter efficient fine-tuning (PEFT)

> Source: Coursera

▼ Parameter efficient fine-tuning (PEFT)

▼ Full fine-tuning of large LLMs is challenging

Training LLMs is computationally intensive. Full fine-tuning requires memory to store the model and various other parameters required during the training process.

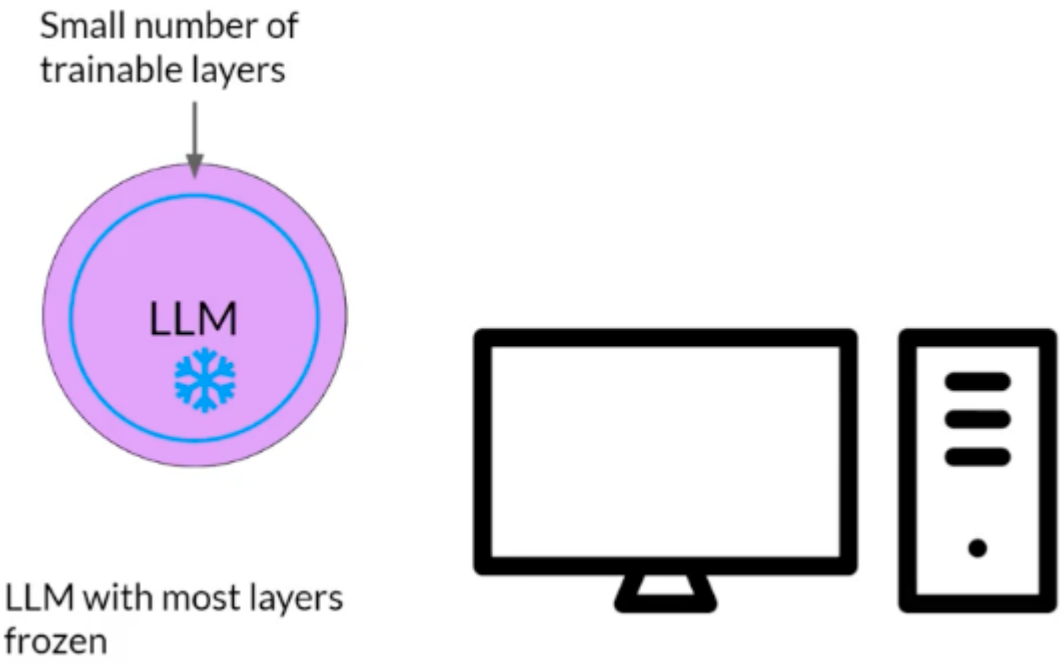
Even if the computer can hold the model weights, which are now hundreds of gigabytes for the largest models, memory for optimizer states, gradients, forward activations, and temporary memory throughout

the training process must also be allocated, these additional components can be many times larger than the model and quickly become too large to handle on consumer hardware.

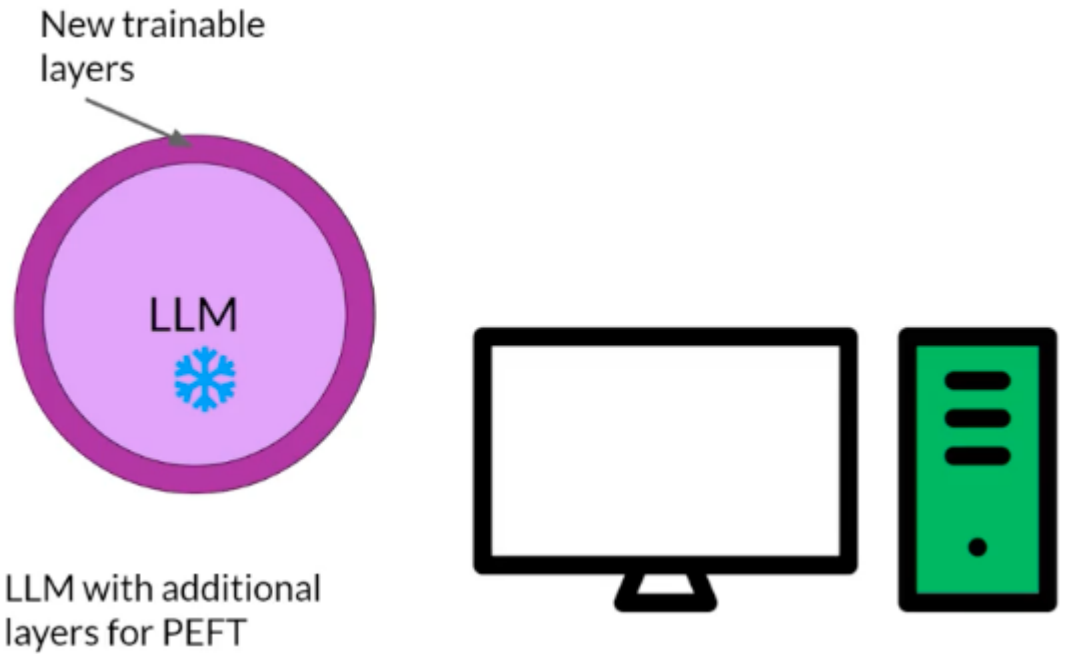
▼ Parameter efficient fine-tuning (PEFT)

In contrast to full fine-tuning, where every model weight is updated during supervised learning, parameter-efficient fine-tuning methods only update a small subset of parameters.

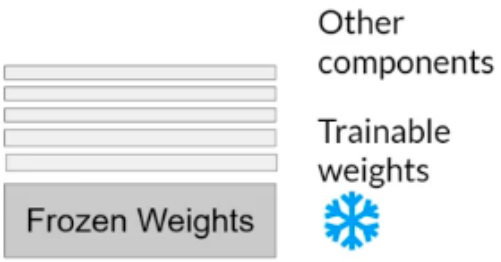
Some path techniques freeze most model weights and focus on fine-tuning a subset of existing model parameters, for example, particular layers or components.



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

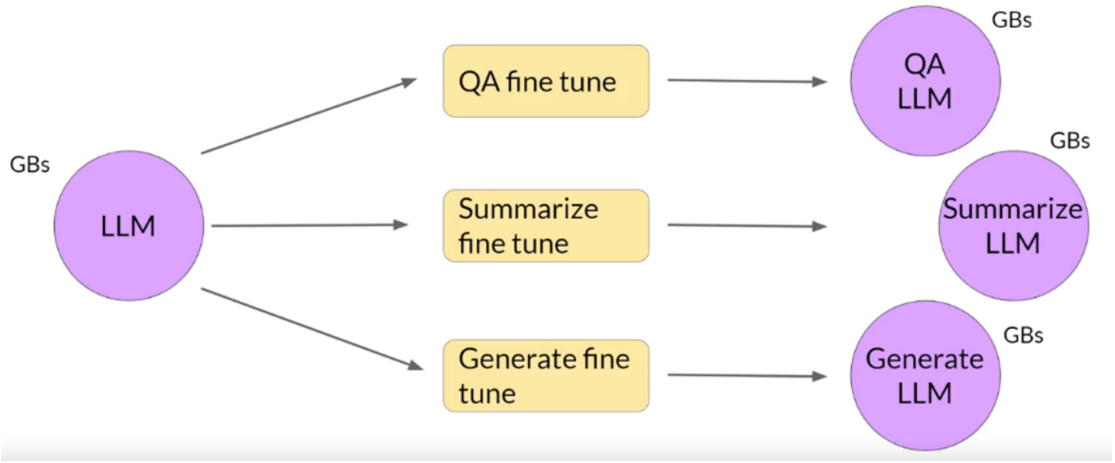


With PEFT, most, if not all, of the LLM weights are kept frozen. As a result, the number of trained parameters is much smaller than those in the original LLM. In some cases, just 15-20% of the original LLM weights. This makes the memory requirements for training much more manageable.



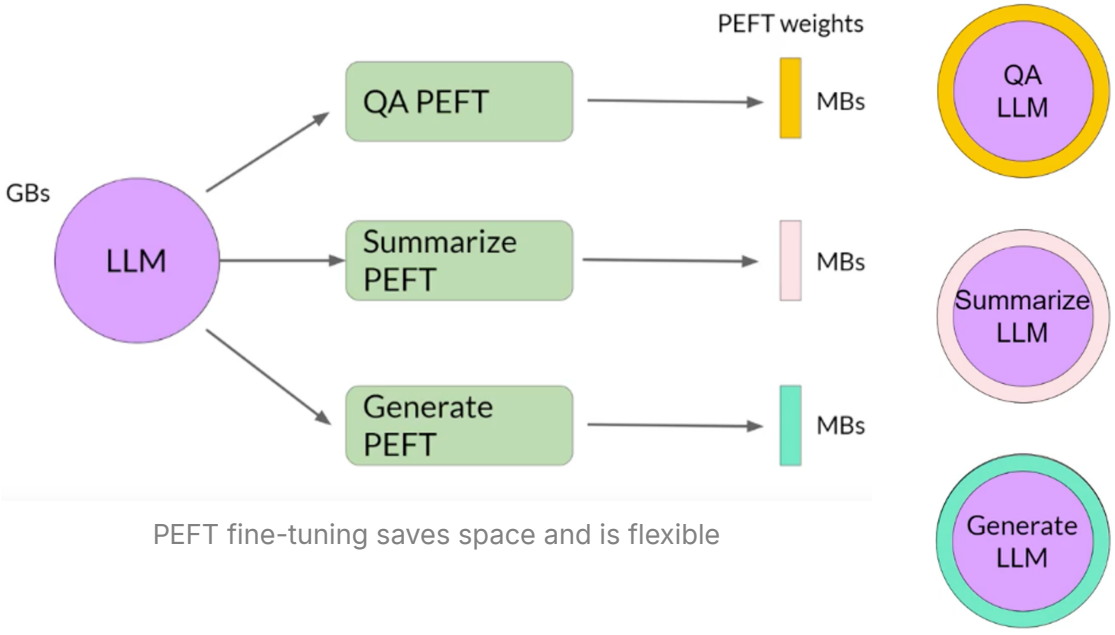
PEFT can often be performed on a single GPU. Because the original LLM is only slightly modified or left unchanged, PEFT is less prone to the catastrophic forgetting problems of full fine-tuning.

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



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

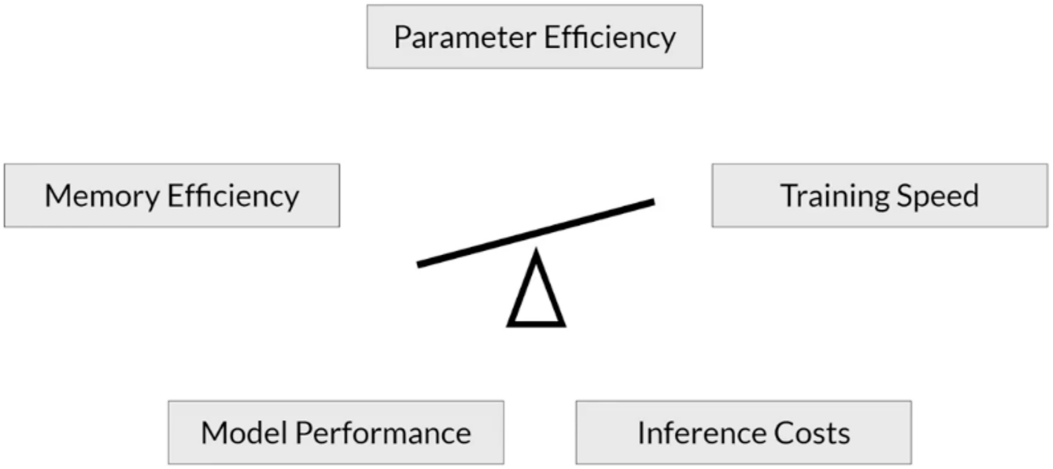
With parameter-efficient fine-tuning, only a small number of weights was trained, which results in a much smaller overall footprint—as small as megabytes, depending on the task.



PEFT fine-tuning saves space and is flexible

The new parameters are combined with the original LLM weights for inference. The PEFT weights are trained for each task and can be easily swapped out for inference, allowing efficient adaptation of the original model to multiple tasks.

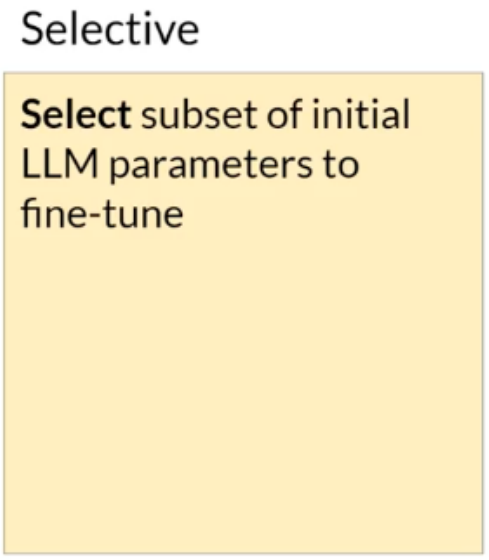
▼ PEFT Trade-offs



Several methods are available for parameter-efficient fine-tuning, each with trade-offs on parameter efficiency, memory efficiency, training speed, model quality, and inference costs.

▼ PEFT methods

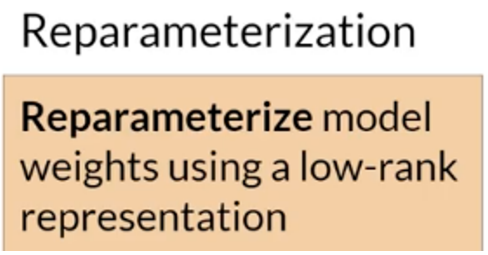
Let's examine the three main classes of PEFT methods.



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

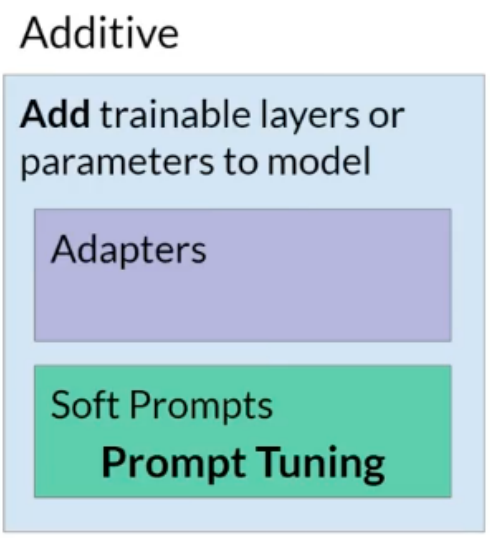
Selective methods fine-tune only a subset of the original LLM parameters. Several approaches exist for identifying which parameters to update. One option is to train only certain components of the model, specific layers, or even individual parameter types.

Researchers have found that the performance of these methods is mixed, and there are significant trade-offs between parameter and compute efficiency. We won't focus on them in this course.



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

Reparameterization methods also work with the original LLM parameters but reduce the number of parameters to train by creating new low-rank transformations of the original network weights. A commonly used technique of this type is LoRA.



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

Lastly, additive methods fine-tune by freezing the original LLM weights and introducing new trainable components with two main approaches.

Adapter methods add new trainable layers to the model's architecture, typically inside the encoder or decoder components after the attention or feed-forward layers.

Soft prompt methods keep the model architecture fixed and frozen, then focus on manipulating the input to achieve better performance. This can be done by adding trainable parameters to the prompt embeddings, keeping the input fixed, and then retraining the embedding weights.

There is a specific soft prompts technique called prompt tuning.

▼ PEFT Techniques 1: LoRA

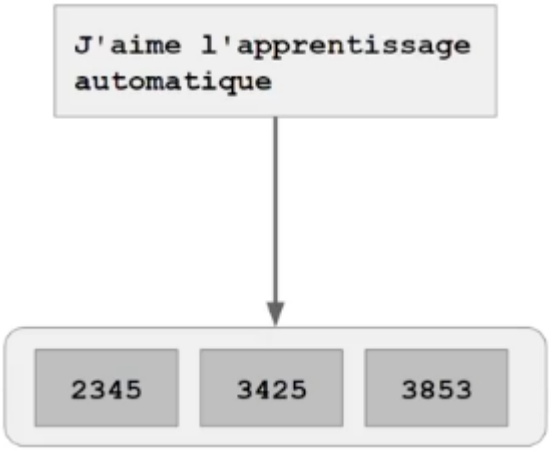
▼ LoRA: Low-Rank Adaption of LLMs

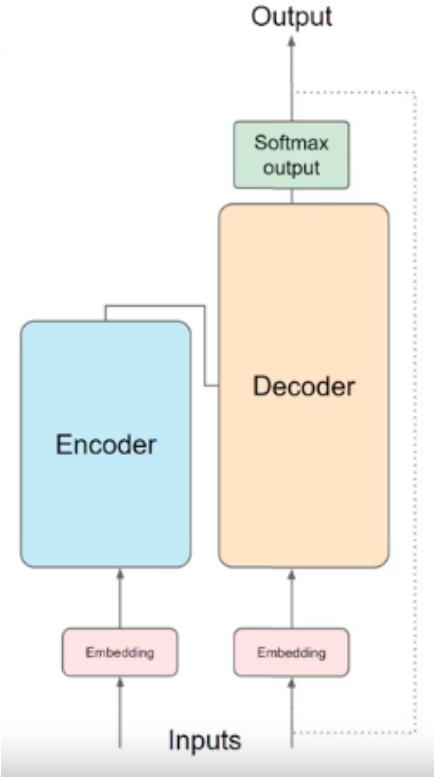
Low-rank Adaptation, or LoRA for short, is a parameter-efficient fine-tuning technique that falls into the re-parameterization category.

▼ Transformer - Recap

Here's the diagram of the transformer architecture that we went through.

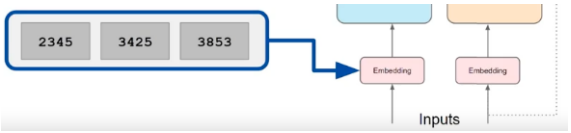
The input prompt is converted into tokens, converted to embedding vectors and passed into the transformer's encoder and/or decoder parts.

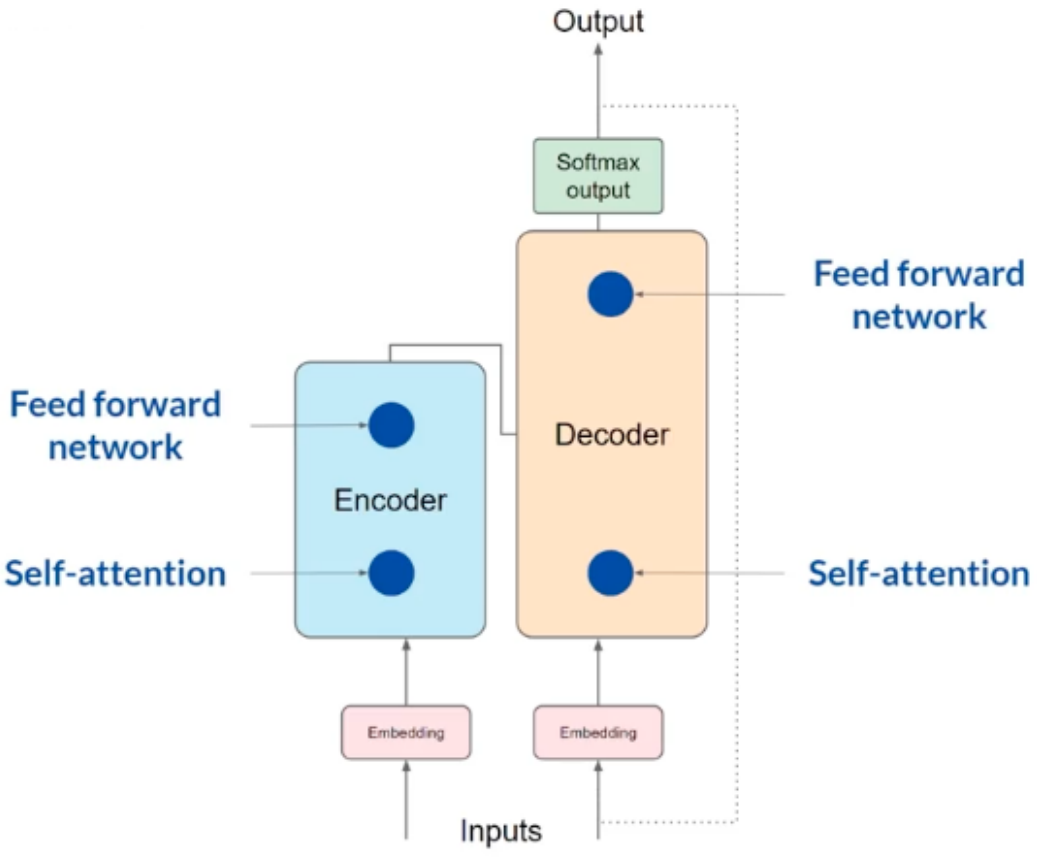




Transformer

Which are converted to embedding vectors and passed into the transformer's encoder and/or decoder parts.

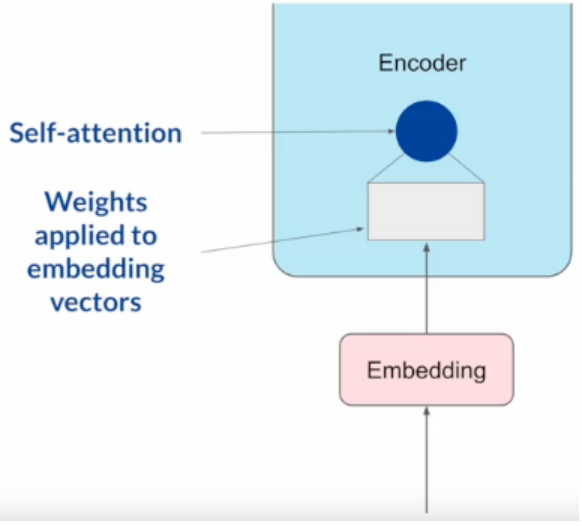


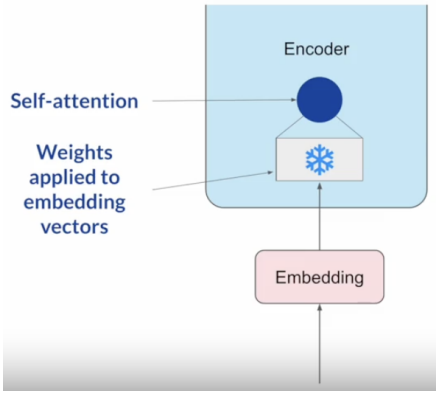


In both of these components, there are two kinds of neural networks: self-attention and feedforward networks. The weights of these networks are learned during pre-training.

▼ LoRA: Low-Rank Adaption of LLMs

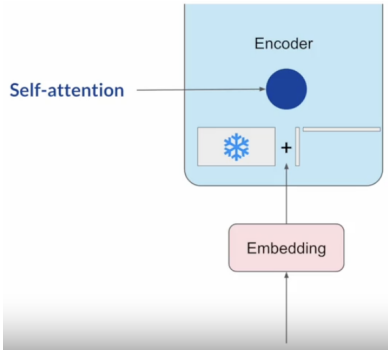
After the embedding vectors are created, they're fed into the self-attention layers, where weights are applied to calculate the attention scores. During full fine-tuning, every parameter in these layers is updated.

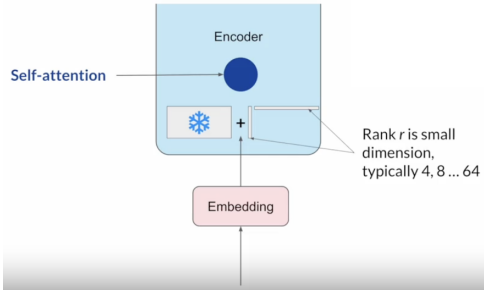




LoRA is a strategy that reduces the number of parameters to be trained during fine-tuning by freezing most of the original model parameters

and then injecting a pair of rank decomposition matrices alongside the original weights. The dimensions of the smaller matrices are set so that their product is a matrix with the same dimensions as the weights they're modifying.

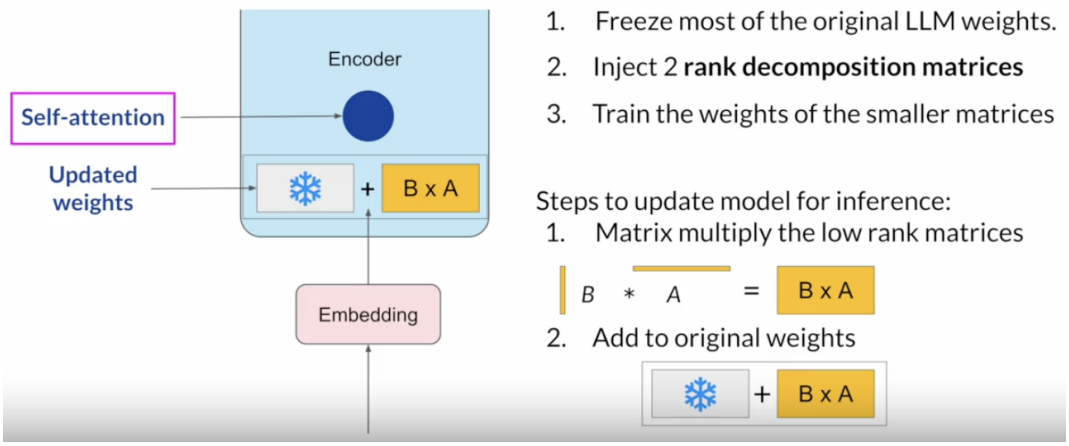




Then, keep the LLM's original weights frozen and train the smaller matrices using the supervised learning process.

For inference, the two low-rank matrices are multiplied to create a matrix with the same dimensions as the frozen weights. Then, add this to the original weights and replace them with these updated values in the model.

Now, we have a LoRA fine-tuned model that can carry out the specific task. Because this model has the same number of parameters as the original, inference latency has little to no impact.



Researchers have found that applying LoRA to just the self-attention layers of the model is often enough to fine-tune for a task and achieve performance gains. However, in principle, you can also use LoRA on other components like the feed-forward layers. However, since most of the parameters of LLMs are in the attention layers, you get the biggest savings in trainable parameters by applying LoRA to these weights matrices.

▼ Concrete example using base Transformer as reference

Let's look at a practical example using the transformer architecture described in the “Attention is All You Need” paper. The paper specifies that the transformer weights have dimensions of 512 by 64. This means that each weights matrix has 32,768 trainable parameters.

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!**

Using LoRA as a fine-tuning method with a rank equal to eight, we will instead train two small rank decomposition matrices whose small dimension is eight. This means that Matrix A will have dimensions of 8

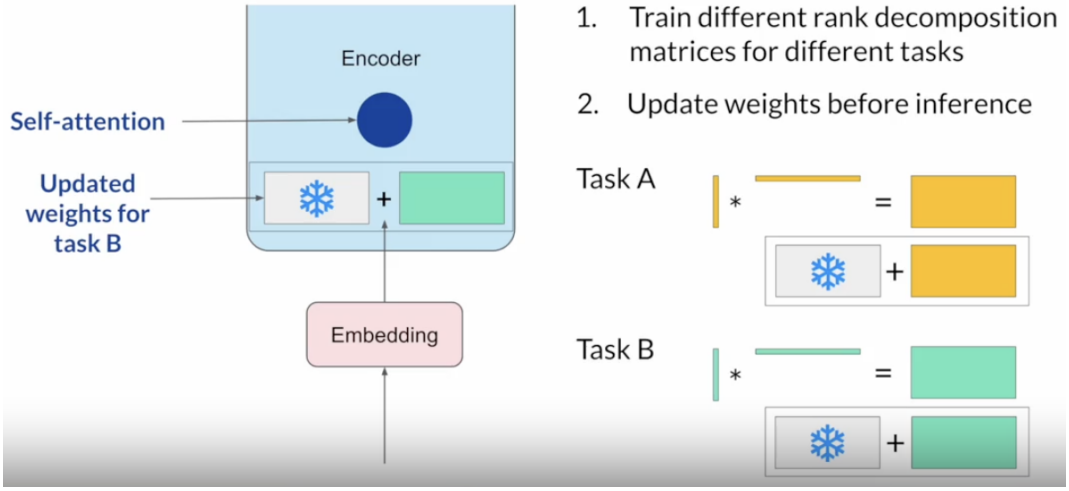
by 64, resulting in 512 total parameters. Matrix B will have dimensions of 512 by 8, or 4,096 trainable parameters.

By updating the weights of these new low-rank matrices instead of the original weights, we'll be training 4,608 parameters instead of 32,768 and 86% reduction. Because LoRA reduces the number of trainable parameters significantly, we can often perform this method of parameter-efficient fine-tuning with a single GPU and avoid the need for a distributed cluster of GPUs.

▼ LoRA

Since the rank-decomposition matrices are small, we can fine-tune a different set for each task and switch them out at inference time by updating the weights.

Suppose we train a pair of LoRA matrices for a specific task; let's call it Task A. To infer this task, multiply these matrices and add the resulting matrix to the original frozen weights.



Then, take this new summed weights matrix and replace the original weights where they appear in the model. This model will then be used to carry out inference on Task A.

If we want to carry out a different task, say Task B, we would take the LoRA matrices we trained for this task, calculate their product, add this matrix to the original weights, and update the model again.

The memory required to store these LoRA matrices is very small. So, in principle, LoRA can be used to train for many tasks. Switch out the

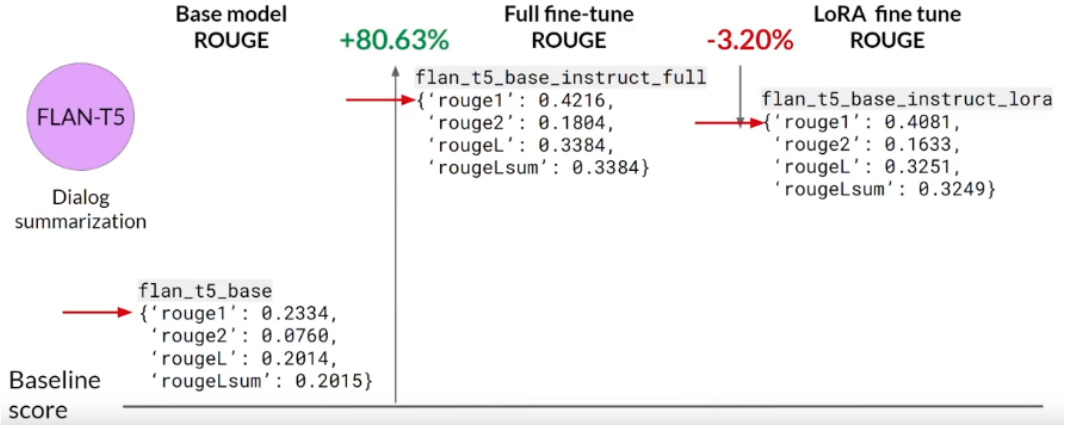
weights when we need to use them and avoid storing multiple full-size versions of the LLM.

How good are these models? Let's use the ROUGE metric to compare the performance of a LoRA fine-tuned model to both an original base model and a full fine-tuned version.

▼ Sample ROUGE metrics for full vs. LoRA fine-tuning

Let's focus on fine-tuning the FLAN-T5 for dialogue summarization. The FLAN-T5-base model has an initial set of full fine-tuning using a large instruction data set.

First, let's set a baseline score for the FLAN-T5 base model and the summarization data set we discussed earlier. Here, we have the ROUGE scores for the base model, where a higher number indicates better performance. We will focus on the ROUGE 1 score for this discussion, but we could use any of these scores for comparison. The scores are fairly low.



Sample ROUGE metrics for full vs. LoRA fine-tuning

Next, look at the scores for a model with additional full fine-tuning on dialogue summarization. Although FLAN-T5 is a capable model, it can still benefit from additional fine-tuning on specific tasks.

With full fine-tuning, we update the model in every way during supervised learning. This results in a much higher ROUGE 1 score, increasing by 0.19 over the base FLAN-T5 model. The additional round of fine-tuning has greatly improved the model's summarization task performance.

Now, let's look at the scores for the LoRA fine-tune model. This process also resulted in a big boost in performance. The ROUGE 1 score has increased from the baseline by 0.17. This is a little lower than full fine-tuning, but not much. However, using LoRA for fine-tuning trained a much smaller number of parameters than full fine-tuning using significantly less computing, so this small performance trade-off may be worth it.

The question is how to choose the rank of the LoRA matrices. In principle, the smaller the rank, the smaller the number of trainable parameters, and the bigger the savings on computing. However, there are some issues related to model performance to consider. In the paper that first proposed LoRA, researchers at Microsoft explored how different choices of rank impacted the model performance on language generation tasks.

Sample ROUGE metrics for full vs. LoRA fine-tuning

Here is the table that summarizes the results. The table shows the rank of the LoRA matrices in the first column, the final loss value of the model, and the scores for different metrics, including BLEU and ROUGE. The bold values indicate the best scores that were achieved for each metric.

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

Choosing the LoRA rank. Source: Hu et al. 2021, "LORA: Low-Rank Adaptation of Large Language Models"

The authors found a plateau in the loss value for ranks greater than 16. In other words, using larger LoRA matrices didn't improve performance. The takeaway is that ranks in the range of 4-32 can

provide a good trade-off between reducing trainable parameters and preserving performance.

Optimizing the choice of rank is an ongoing area of research, and best practices may evolve as more practitioners like you use LoRA. LoRA is a powerful fine-tuning method that achieves great performance. The principles behind the method are useful for training LLMs and models in other domains. The final path method you'll explore this week doesn't change the LLM but instead focuses on training your input text. Join me in the next video to learn more.

▼ PEFT techniques 2: Soft prompts

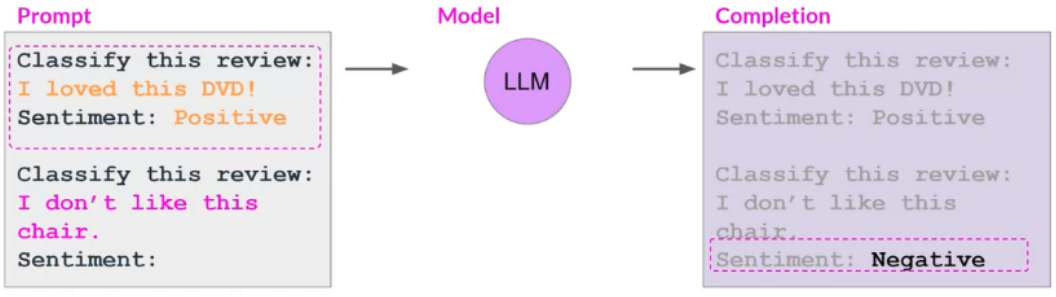
With LoRA, the goal was to find an efficient way to update the model's weights without training every parameter again. Additive methods within PEFT also aim to improve model performance without changing the weights.

▼ Prompt tuning is not prompt engineering!

Prompt tuning sounds like prompt engineering, but they are quite different.

▼ Prompt engineering

Prompt engineering works on the language of the prompt to get the desired completion. 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 task's nature and generate a better completion.

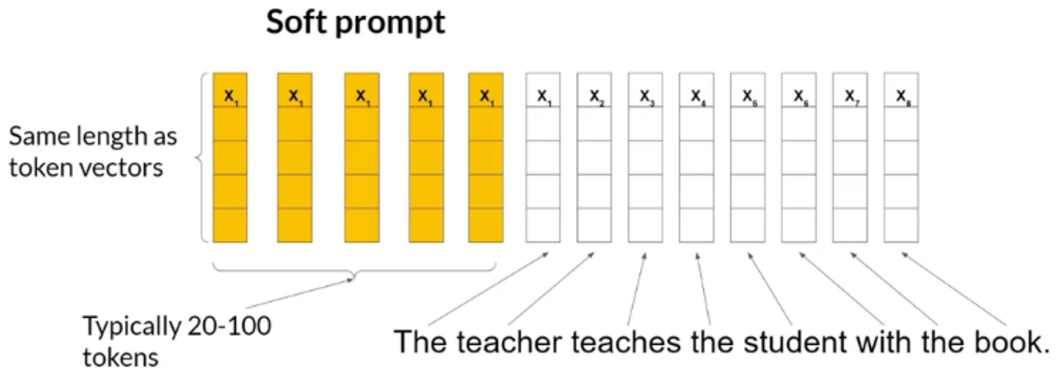


One-shot or Few-shot Inference

However, prompt engineering has some limitations, as writing and trying different prompts can require a lot of manual effort. It is also limited by the length of the context window, and at the end of the day, we may still not achieve the needed performance for the task.

▼ Prompt tuning

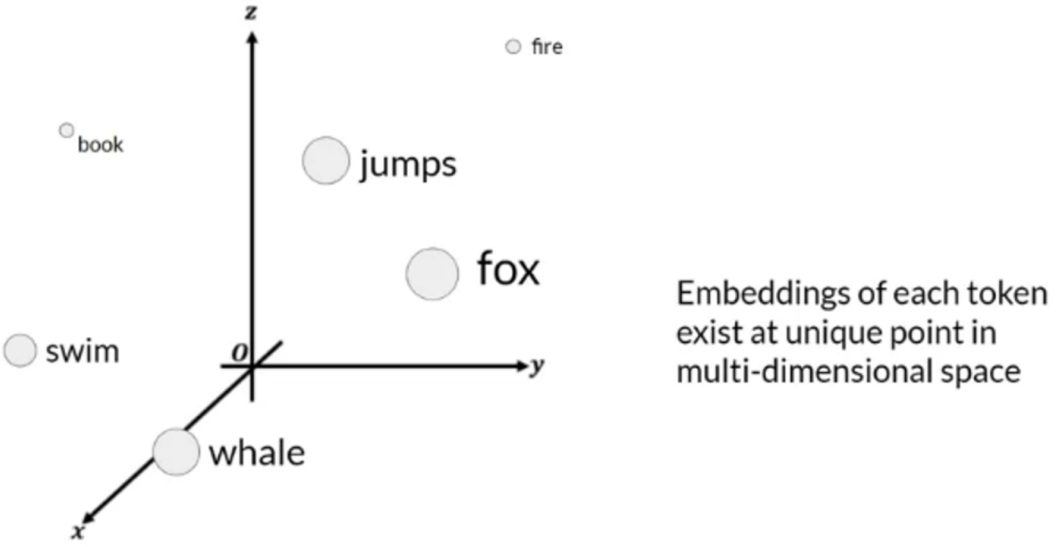
Prompt tuning adds trainable tokens to the prompt and leaves it up to the supervised learning process to determine their optimal values. The set of trainable tokens is called a soft prompt, and it gets prepended to embedding vectors that represent your input text.



Prompt tuning adds a trainable "soft prompt" to inputs

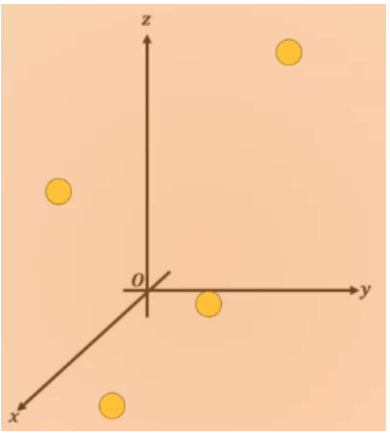
The soft prompt vectors have the same length as the embedding vectors of the language tokens. Including between 20 and 100 virtual tokens can be sufficient for good performance. The tokens representing natural language are hard because they correspond to a fixed location in the embedding vector space.

However, the soft prompts are not fixed discrete words of natural language.



Soft prompts

Instead, they can be considered virtual tokens that can take on any value within the continuous multidimensional embedding space. Through supervised learning, the model learns the values for these virtual tokens that maximize performance for a given task.

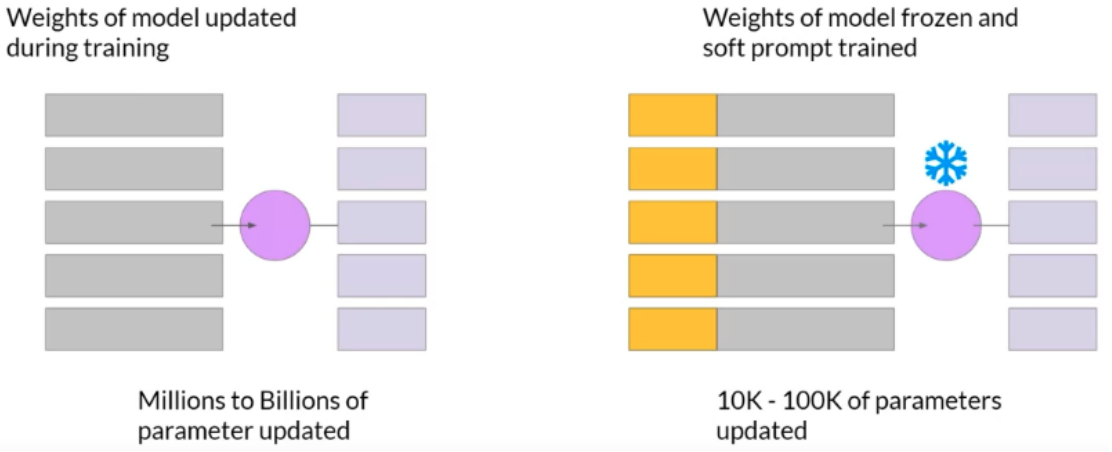


Soft prompts

Full Fine-tuning vs prompt tuning

The training data set consists of input prompts and output completions or labels in full fine-tuning. The weights of the large language model are updated during supervised learning.

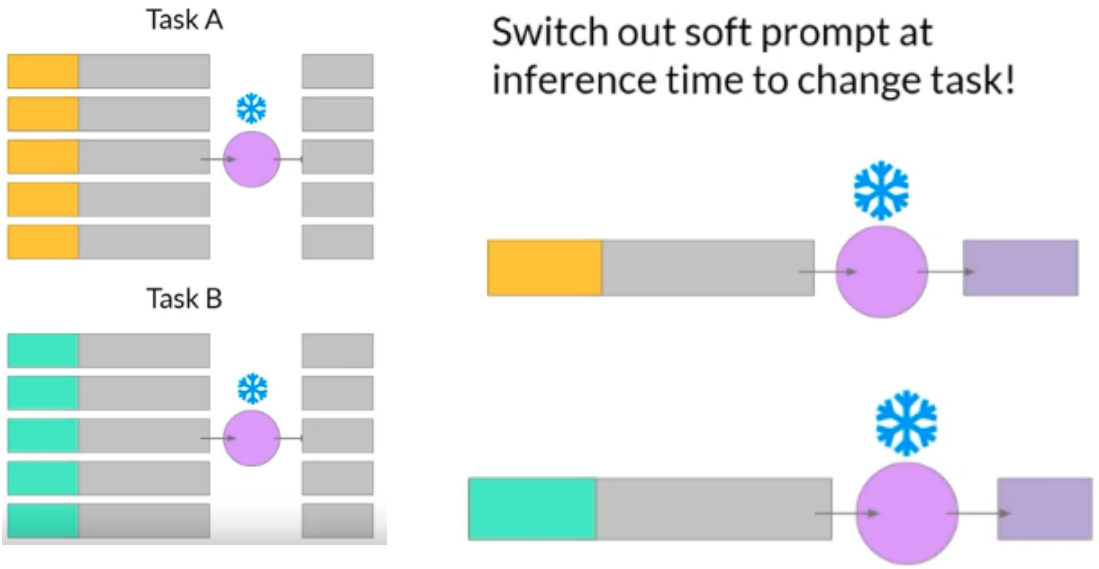
In contrast with prompt tuning, the weights of the large language model are frozen, and the underlying model is not updated. Instead, the embedding vectors of the soft prompt get updated over time to optimize the model's completion of the prompt. Prompt tuning is a very parameter-efficient strategy because only a few parameters are being trained.



This contrasts with the millions to billions of parameters in full fine-tuning, similar to LoRA.

Prompt tuning for multiple tasks

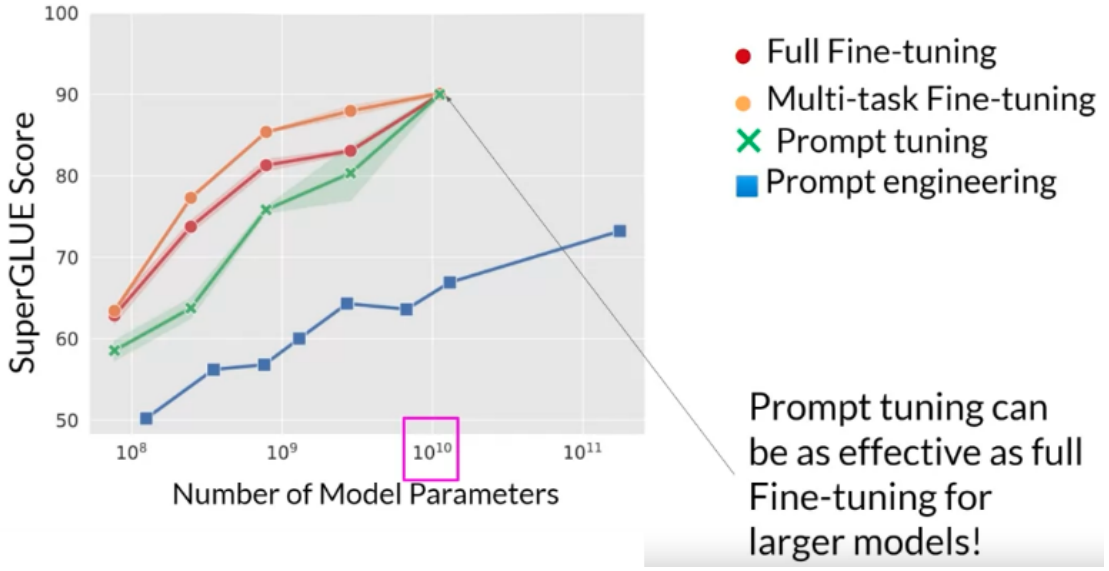
Different soft prompts can be trained for each task and easily swapped out at inference time. A set of soft prompts can be trained for one task and a different set for another.



To use them for inference, we prepend the input prompt with the learned tokens to switch to another task, simply changing the soft prompt. Soft prompts are very small on disk, so this kind of fine-tuning is extremely efficient and flexible. The same LLM is used for all tasks; we switch out the soft prompts at inference time.

▼ Performance of prompt tuning

So, how well does prompt tuning perform? In the original paper, Exploring the Method, by Brian Lester and collaborators at Google. The authors compared prompt tuning to several other methods for various model sizes.

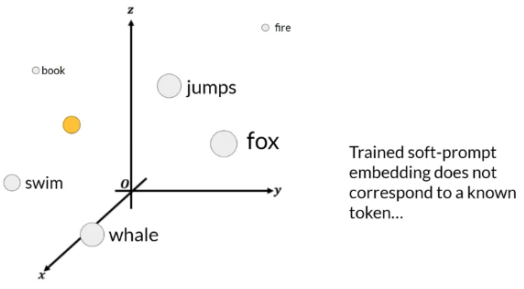


Source: Lester et al. 2021, "The Power of Scale for Parameter-Efficient Prompt Tuning"

In this figure from the paper, the Model size is on the X-axis, and the SuperGLUE score is on the Y-axis. This is the evaluation benchmark you learned about earlier this week that grades model performance on many different language tasks. The red line shows the scores for models created through full fine-tuning on a single task. The orange line shows the score for models created using multitask fine-tuning. The green line shows the performance of prompt tuning, and finally, the blue line shows scores for prompt engineering only.

Prompt tuning doesn't perform as well as full fine-tuning for smaller LLMs. However, as the model size increases, prompt tuning is performed. Once models have around 10 billion parameters, prompt tuning can be as effective as full fine tuning and offers a significant boost in performance over prompt engineering alone.

▼ Interpretability of soft prompts

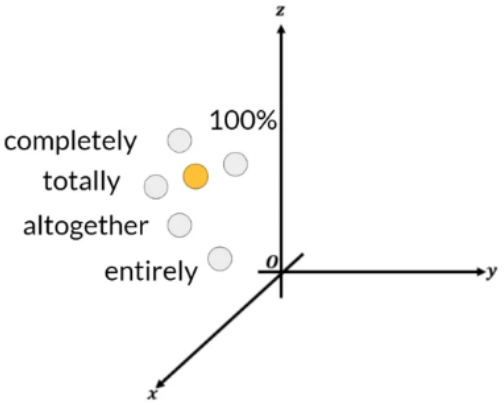


One potential issue to consider is the interpretability of learned virtual tokens. The soft prompt tokens can take any value within the continuous embedding vector space. The trained tokens don't correspond to any known token,

word, or phrase in the LLM's vocabulary.

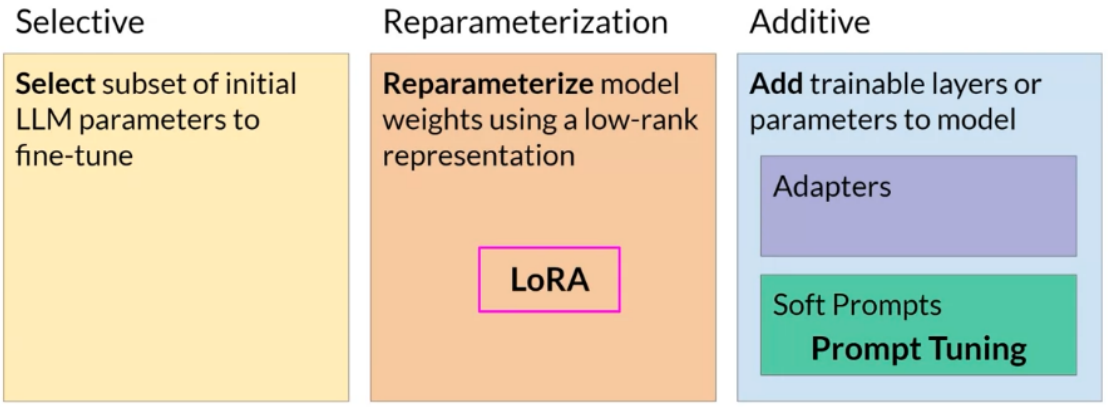
However, analysing the nearest neighbour tokens to the soft prompt location shows they form tight semantic clusters.

In other words, the words closest to the soft prompt tokens have similar meanings. The words identified usually have some meaning related to the task, suggesting that the prompts are learning word-like representations.



▼ PEFT Methods summary

So far, this lesson has two PEFT methods: LoRA, which uses rank decomposition matrices to update the model parameters efficiently, and Prompt Tuning, in which trainable tokens are added to the prompt while the model weights are untouched.



Both methods fine-tune models, potentially improving task performance, using much less computing than full fine-tuning methods. LoRA is broadly used in practice because it offers comparable performance to full fine-tuning for many tasks and data sets.

▼ Keys takeaway

- The method to adapt a foundation model through a process called instruction fine-tuning.
- The prompt templates and data sets were used to train the FLAN-T5 model.
- Evaluation metrics and benchmarks such as ROUGE and HELM measure success during model fine-tuning.
- Instruction finetuning has proven effective and useful across various natural language use cases and tasks. Amazingly, we can fine-tune a model to a specific task with just a few hundred examples.
- How can parameter-efficient fine-tuning, or PEFT, reduce the computing required to fine-tune a model?
- Two methods can be used for this LoRA and Prompt Tuning.
- We can also combine LoRA with the quantization techniques we learned in week 1 to further reduce the memory footprint. This is known as QLoRA.
- In practice, PEFT is used heavily to minimize computing and memory resources, ultimately reducing the cost of fine-tuning. This allows us to maximise our computing budget and speed up development.

▼ Question

Parameter-efficient fine-tuning (PEFT) methods specifically address some of the challenges of performing full fine training. Which of the following options describes challenges that PEFT tries to overcome?

☐ **Model performance**

☑ **Storage requirements**

> 💡 With PEFT, we can change just a small amount of parameters when fine-tuning, so during inference, you can combine the original model with the new parameters instead of duplicating the entire model for each new task you want to perform fine-tuning.

☑ **Catastrophic forgetting**

> 💡 With PEFT, most of the LLM's parameters are unchanged, which helps to make it less prone to catastrophic forgetting.

☑️ Computational constraints

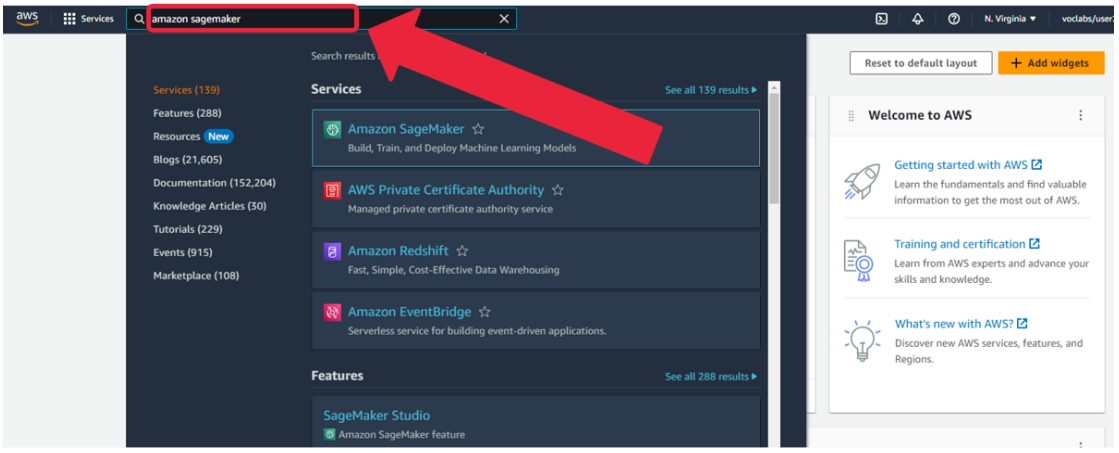
> 💡 Because most parameters are frozen, we typically only need to train 15%-20% of the original LLM weights, making the training process less expensive (less memory required)

▼ Lab 2 - Practice

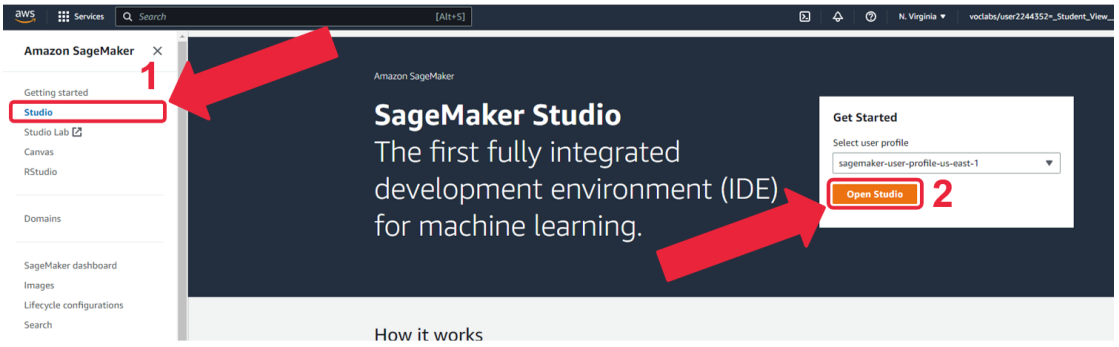
▼ AWS Sagemaker

This section is for those with AWS Sagemaker service; if not, using Google Colab or Jupyter on your local host is fine.

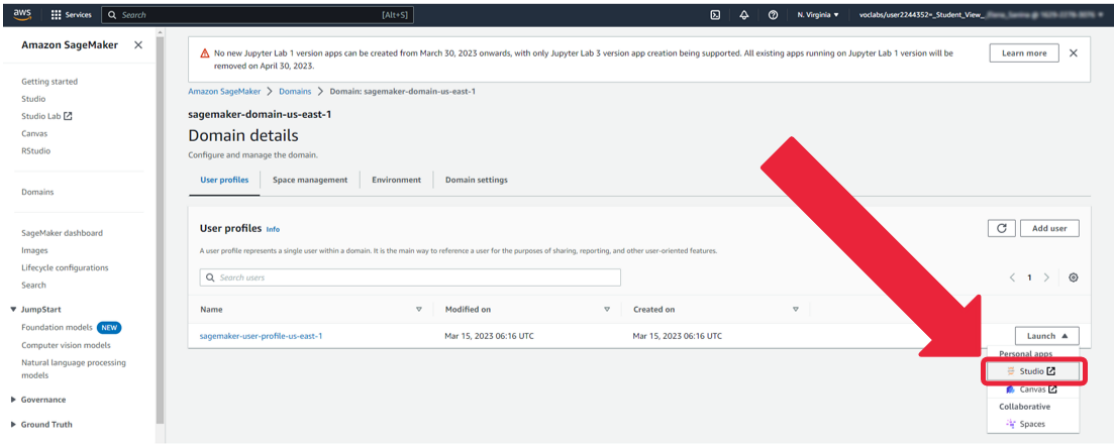
Go to **Amazon SageMaker**.



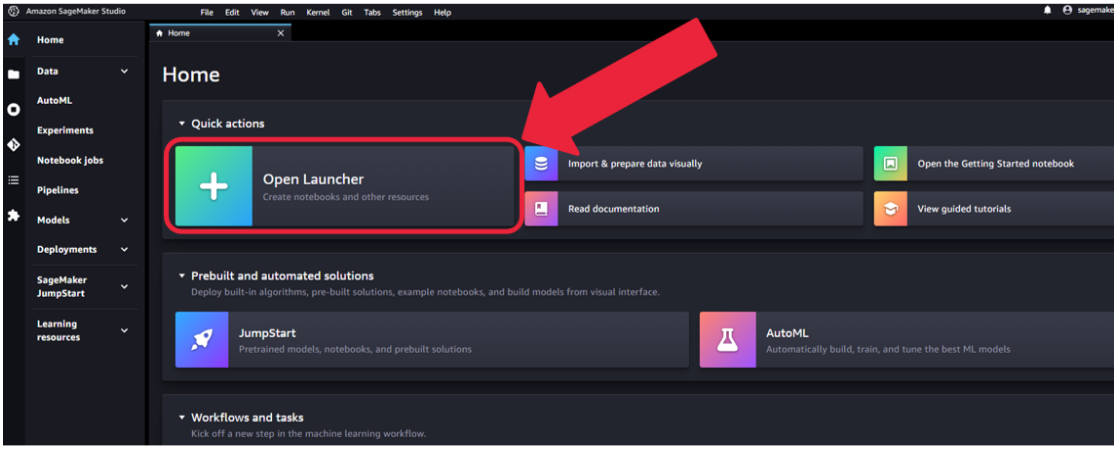
Studio → Open Studio.



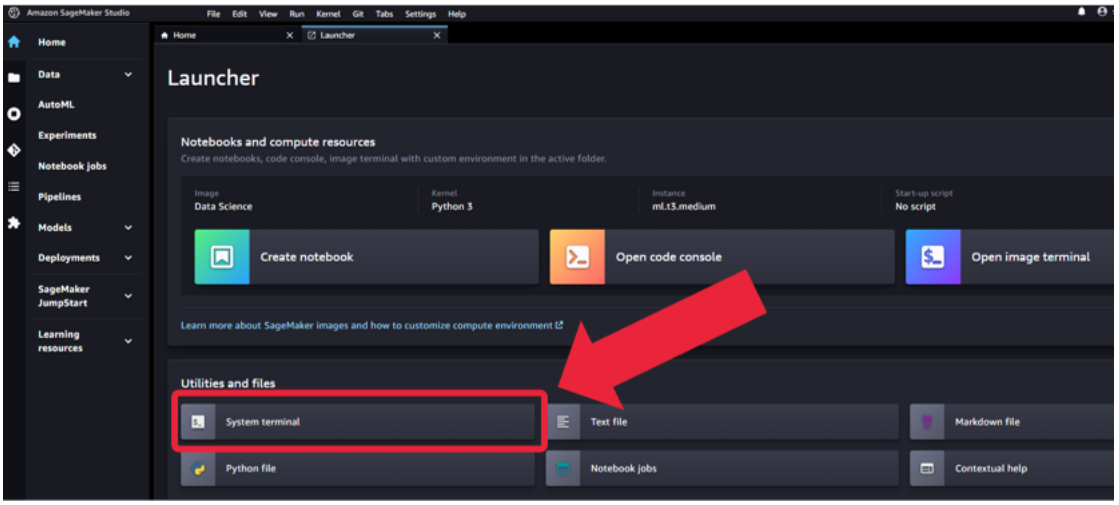
Launch → Studio.



Open Launcher.



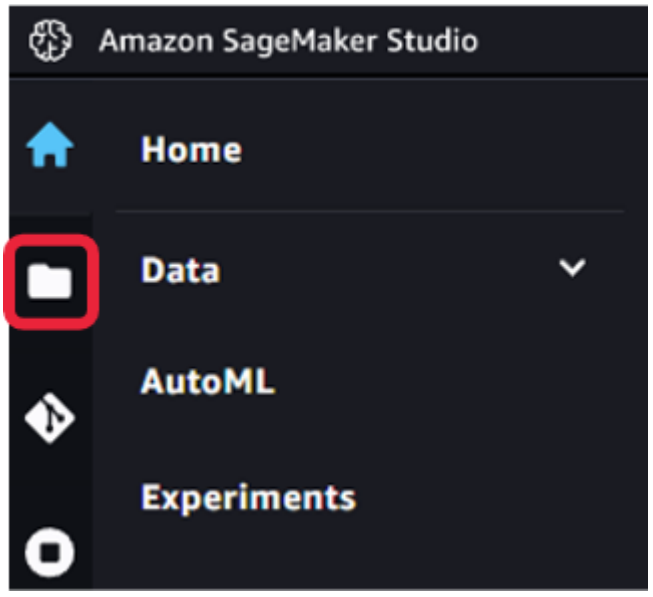
Open System terminal



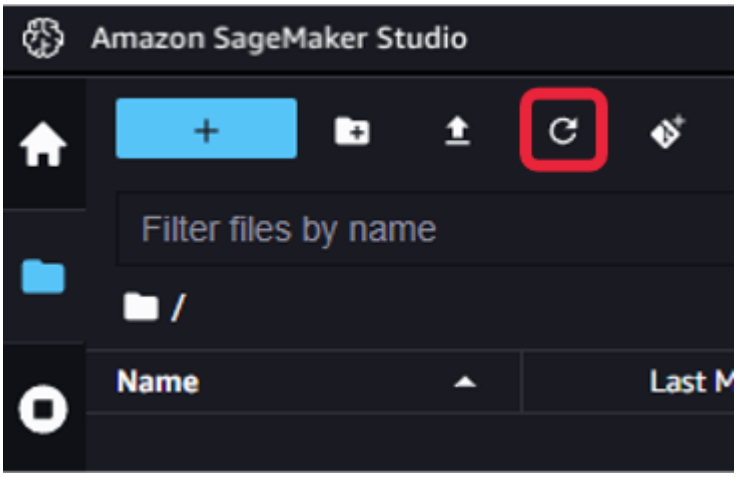
Use the following command (you can copy and paste it) in the System terminal to download the lab:

aws s3 cp --recursive s3://dlai-generative-ai/labs/w2-170864/ ./

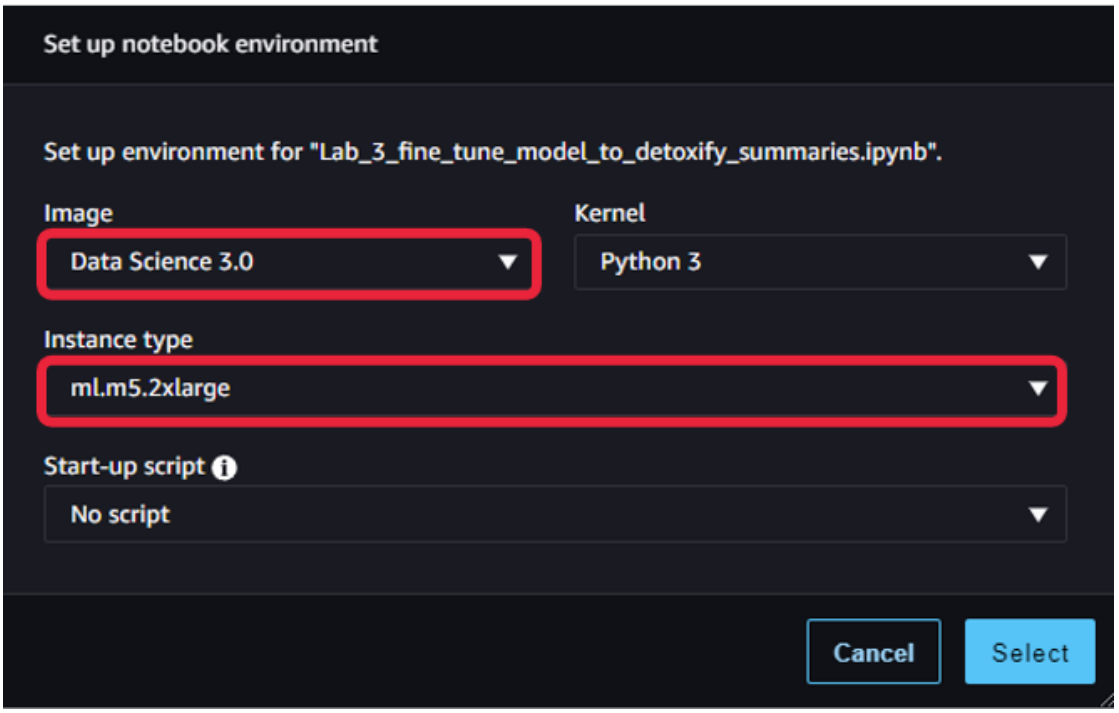
Click on the folder icon on the left to find the downloaded notebook:



Note: You might need to update the environment to see the downloaded notebook.



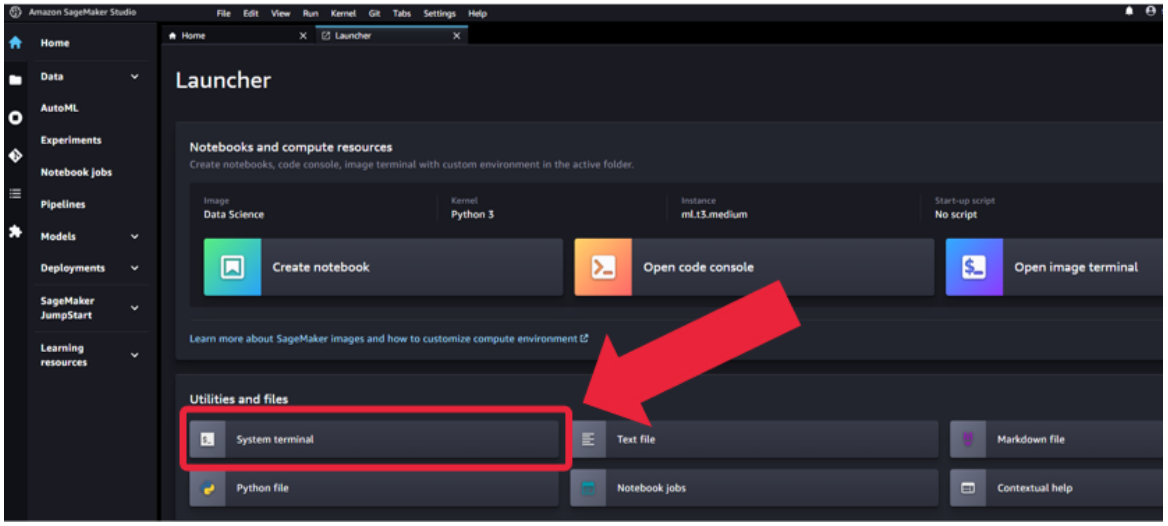
Open the notebook (if you need to set the kernel, please choose "Python 3 (Data Science 3.0)" with instance type `ml.m5.2xlarge`).



Finish the lab

If you wish to download the notebook, right-click on the notebook file in the File Browser, then select **Download**. Or, in the main menu, choose **File**, then **Download**.

Open System terminal



Use the following command (you can copy and paste it) in the System terminal to download the lab:

aws s3 cp --recursive s3://dlai-generative-ai/labs/w2-170864/ ./

▼ Week 2 quiz

1. Fill in the blanks: _____ involves using many prompt-completion examples as the labelled training dataset to continue training the model by updating its weights. This is different from _____, where you provide prompt-completion examples during inference.
   ◉ **Instruction fine-tuning, In-context learning**
   ○ **Prompt engineering, Pre-training**
   ○ **In-context learning, Instruction fine-tuning**
   ○ **Pre-training, Instruction fine-tuning**

2. Fine-tuning a model on a single task can improve model performance specifically on that task; however, it can also degrade the performance of other tasks as a side effect. This phenomenon is known as:

   ○ **Model toxicity**
   ○ **Catastrophic loss**
   ○ **Instruction bias**
   ◉ **Catastrophic forgetting**

3. Which evaluation metric below focuses on precision in matching generated output to the reference text and is used for text translation?
   ○ **ROUGE-1**
   ○ **HELM**
   ○ **ROUGE-2**
   ◉ **BLEU**

4. Which of the following statements about multi-task finetuning is correct? Select all that apply:
   ☑ **Multi-task finetuning can help prevent catastrophic forgetting.**

   ☐ **Performing multi-task finetuning may lead to slower inference.**

   ☑ **FLAN-T5 was trained with multi-task finetuning.**

   ☐ **Multi-task finetuning requires separate models for each task being performed.**

5. "Smaller LLMs can struggle with one-shot and few-shot inference:"

Is this true or false?

◉ **True**
○ **False**

6. Which of the following are Parameter Efficient Fine-Tuning (PEFT) methods? Select all that apply.
   ☑
   Reparameterization

   ☑ **Additive**

   ☐ **Subtractive**

   ☑ **Selective**

7. Which of the following best describes how LoRA works?
   ○ **LoRA continues the original pre-training objective on new data to update the weights of the original model.**
   ◉ **LoRA decomposes weights into two smaller rank matrices and trains those instead of the full model weights.**
   ○ **LoRA freezes all weights in the original model layers and introduces new components which are trained on new data.**
   ○ **LoRA trains a smaller, distilled version of the pre-trained LLM to reduce model size**

8. What is a soft prompt in the context of LLMs (Large Language Models)?
   ◉ **A set of trainable tokens that are added to a prompt and whose values are updated during additional training to improve performance on specific tasks.**
   ○ **A strict and explicit input text that serves as a starting point for the model's generation.**
   ○ **A technique to limit the creativity of the model and enforce specific output patterns.**
   ○ **A method to control the model's behaviour by adjusting the learning rate during training.**

9. "Prompt Tuning is a technique used to adjust all hyperparameters of a language model."

   Is this true or false?

○ **True**
◉ **False**

10. "PEFT methods can reduce the memory needed for fine-tuning dramatically, sometimes to just 12-20% of the memory needed for full fine-tuning."

    Is this true or false?

    ◉ **True**
    ○ **False**

▼ Reading: Week 2 Resources

Generative AI Lifecycle

- **Generative AI on AWS: Building Context-Aware, Multimodal Reasoning Applications** - This O'Reilly book dives deep into all phases of the generative AI lifecycle including model selection, fine-tuning, adapting, evaluation, deployment, and runtime optimizations.

Multi-task, instruction fine-tuning

- **Scaling Instruction-Finetuned Language Models** - Scaling fine-tuning with a focus on task, model size and chain-of-thought data.
- **Introducing FLAN: More Generalizable Language Models with Instruction Fine-Tuning—**This blog (and article) explores instruction fine-tuning, which aims to improve language models' performance at performing NLP tasks with zero-shot inference.

Model Evaluation Metrics

- **HELM - Holistic Evaluation of Language Models** - HELM is a living benchmark for evaluating language models more transparently.
- **General Language Understanding Evaluation (GLUE) benchmark** - This paper introduces GLUE, a benchmark for evaluating models on diverse natural language understanding (NLU) tasks and emphasizing the importance of improved general NLU systems.

- **SuperGLUE** - This paper introduces SuperGLUE, a benchmark designed to evaluate the performance of various NLP models on a range of challenging language understanding tasks.
- **ROUGE: A Package for Automatic Evaluation of Summaries** - This paper introduces and evaluates four different measures (ROUGE-N, ROUGE-L, ROUGE-W, and ROUGE-S) in the ROUGE summarization evaluation package, which assess the quality of summaries by comparing them to ideal human-generated summaries.
- **Measuring Massive Multitask Language Understanding (MMLU)** - This paper presents a new test to measure multitask accuracy in text models, highlighting the need for substantial improvements in achieving expert-level accuracy and addressing lopsided performance and low accuracy on socially important subjects.
- **BigBench-Hard - Beyond the Imitation Game: Quantifying and Extrapolating the Capabilities of Language Models** - The paper introduces BIG-bench, a benchmark for evaluating language models on challenging tasks, providing insights on scale, calibration, and social bias.

Parameter- efficient fine-tuning (PEFT)

- **Scaling Down to Scale Up: A Guide to Parameter-Efficient Fine-Tuning** - This paper provides a systematic overview of Parameter-Efficient Fine-tuning (PEFT) Methods in all three categories discussed in the lecture videos.
- **On the Effectiveness of Parameter-Efficient Fine-Tuning** - The paper analyzes sparse fine-tuning methods for pre-trained models in NLP.

LoRA

- **LoRA Low-Rank Adaptation of Large Language Models** This paper proposes a parameter-efficient fine-tuning method that uses low-rank decomposition matrices to reduce the trainable parameters needed for fine-tuning language models.
- **QLoRA: Efficient Finetuning of Quantized LLMs** - This paper introduces an efficient method for fine-tuning large language models on a single

GPU based on quantization, achieving impressive results on benchmark tests.

Prompt tuning with soft prompts

- **The Power of Scale for Parameter-Efficient Prompt Tuning—**The paper explores "prompt tuning," a method for conditioning language models with learned soft prompts. This method achieves competitive performance compared to full fine-tuning and enables model reuse for many tasks.

▼ Copyright Notice