FINE TUNING LLMs USING QLoRA:

What is fine tuning?
Fine tuning is a process where a pre-trained model is further trained on a custom dataset to adapt it for particular tasks like sentiment analysis, question answering, or text summarization.
Fine-tuning leverages the strengths of pre-trained models, offering a powerful, efficient, and flexible approach to adapting models for specific tasks. It significantly reduces the need for extensive computational resources and large datasets

Challenges:
Memory management: Large models consume significant amounts of memory, and storing the parameters of these models, especially when multiple versions are maintained, requires considerable storage capacity.
Resource constraints: Depending on the size of the model and the fine-tuning dataset, the process can take a significant amount of time and also high-performance GPUs or TPUs are often required to handle the computation load.
Catastrophic forgetting: During fine-tuning, the model may forget the general knowledge it acquired during pre-training, especially if the task-specific dataset is small or significantly different from the pre-training data.

To overcome these challenges, **Parameter-efficient fine-tuning (PEFT)** is a strategy designed to enhance the performance of large AI models while optimising resource usage such as time, energy, and computational power. This method targets a small subset of critical parameters for adjustment, maintaining the majority of the pre-trained model's structure intact.

- Only a subset of the model's parameters, typically those most relevant to the new task, are updated.
- This reduces the computational load and memory requirements compared to full fine-tuning.
- It enables the use of larger models on hardware with limited resources.
- By updating only key parameters, PEFT helps retain the valuable information learned during the initial pre-training phase, improving the model's generalization to new tasks without losing its original capabilities.

LoRA: Low Rank Adaptation of Large Language Models:

LoRA is a technique for parameter-efficient fine-tuning of large pre-trained models. It focuses on reducing the number of parameters that need to be updated during fine-tuning, thereby lowering the computational and memory requirements while maintaining or even improving model performance.

This technique introduces trainable rank decomposition matrices into each layer of transformer architecture and also reduces trainable parameters for downstream tasks while keeping the pre-trained weights frozen.

Working:

- The weight matrix W of a neural network layer is decomposed into two low-rank matrices A and B such that W ≈ A X B and A and B have significantly fewer parameters than W.
- During fine-tuning, only the low-rank matrices A and B are updated while the original weight matrix W is kept fixed. This allows the model to adapt to new tasks with minimal parameter updates.

This method can minimize the number of trainable parameters by up to 10,000 times and the GPU memory necessity by 3 times while still performing on par or better than fine-tuning model quality on various tasks.

QLoRA: It is the extended version of LoRA which works by quantizing the precision of the weight parameters in the pre-trained LLM to 4-bit precision. This method aims to reduce both the computational and memory overhead associated with fine-tuning.

- 4-Bit NormalFloat (NF4) Quantization: QLoRA introduces a new data type called 4-bit NormalFloat (NF4) that uses quantile quantization to estimate the quantiles in a 0 to 1 distribution, then normalizes the values into the [-1, 1] range. This allows quantizing neural network weights into 4-bit representations while preserving important information.
- Double Quantization: QLoRA employs double quantization, which involves quantizing not just the model parameters but also the quantization constants used in the NF4 quantization process. This further reduces memory usage with minimal impact on performance.
- Paged Optimizers: QLoRA leverages NVIDIA's unified memory feature to automatically page the optimizer states between CPU and GPU memory as needed. This allows processing large data sequences efficiently without running into GPU memory constraints.

In QLoRA, quantization is applied to reduce the memory footprint of the model. This technique involves converting the model’s weights from a float32 format to a smaller one, typically 4 or 8 bits. Then, freeze the quantized weights of the base model and perform backpropagation only on the weights of a lower-rank matrix that overlays the quantized base model.
Furthermore, the quantized model occupies much less RAM space than the original one (the google-t5/t5 3B model memory footprint reduces from approximately 11.4GB to just 4.29GB), allowing for development on a powerful local machine or a free Google Colab instance.

Objective: Fine tune Microsoft's phi-1_5 model using QLoRA.

phi-1_5: The language model Phi-1.5 is a Transformer with 1.3 billion parameters. Its training involved a variety of data sources, including subsets of Python codes from The Stack v1.2, Q&A content from StackOverflow, competition code from code_contests, and synthetic Python textbooks and exercises generated by gpt-3.5-turbo-0301, augmented with a new data source that consists of various NLP synthetic texts.
Phi-1.5 can write poems, draft emails, create stories, summarize texts and write Python code.

The model is fine-tuned on a dataset called DS-1000, which contains 1,000 data science-related questions covering seven different Python libraries. The questions in this dataset reflect diverse, realistic, and practical use cases in the field of data science.
The goal of fine-tuning this model is to improve its ability to answer questions related to data science and generate relevant code snippets to address these questions. The model is expected to become more knowledgeable and capable of providing helpful responses to users seeking information or assistance with data science tasks across a range of Python libraries and use cases.

Fine-tuning process:

- Library Installation and Imports: The necessary libraries, such as those for natural language processing, data preprocessing, and model training, have been installed and imported into the execution environment.
- Model and Tokenizer Loading: The pre-trained language model and its corresponding tokenizer have been directly loaded from their respective Hugging Face repositories onto the Colab environment.
- Dataset Preparation: The dataset, which is also hosted on the Hugging Face platform, has been loaded and all the data points have been preprocessed as required for the fine-tuning task.
- Model Quantization: The language model has been quantized using the bitsandbytes configuration library, which reduces the memory usage of the model without significantly impacting its performance.
- LoRA Configuration: The Low-Rank Adaptation (LoRA) configuration has been set up, specifying the parameters such as the rank, scaling factor, target modules, and dropout rate. This LoRA configuration has then been applied to specific layers of the pre-trained model.
- Training Setup and Execution: The training arguments, such as the number of epochs, learning rate, and batch size, have been defined. A Trainer object has been set up using these arguments, and the fine-tuning process has been executed for the specified number of epochs.

- Model Saving: The fine-tuned model has been saved to a designated location, likely for future use or deployment.
- Text Generation Pipeline: A text generation pipeline has been created using the fine-tuned model and its corresponding tokenizer. This pipeline is then used to generate a response to a given prompt, which is printed to the console.

Objective: Fine tune falcon-7b model using QLoRA.

The Falcon 7B model consists of 7 billion parameters. It utilizes the transformer architecture that includes self-attention mechanisms that enables the model to weigh the importance of different words in a sentence when generating responses.
It was trained on a large and diverse dataset sourced from a variety of text corpora. The data includes books, articles, websites, and other text forms to ensure the model can understand and generate human-like text across different contexts and topics. Features include: Text generation, text completion, summarization, translation, question answering.

This model is finetuned on the amod-mental health conversations dataset that contains questions and answers sourced from two online counselling and therapy platforms. The questions cover a wide range of mental health topics, and the answers are provided by qualified psychologists.

Fine-tuning process:

- Library Installation and Imports: The necessary libraries, such as those for natural language processing, data preprocessing, and model training, have been installed and imported into the execution environment.
- Model and Tokenizer Loading: The pre-trained language model and its corresponding tokenizer have been directly loaded from their respective Hugging Face repositories onto the Colab environment.
- Dataset Preparation: The dataset, which is also hosted on the Hugging Face platform, has been loaded and all the data points have been preprocessed as required for the fine-tuning task.
- Model Quantization: The language model has been quantized using the bitsandbytes configuration library, which reduces the memory usage of the model without significantly impacting its performance.
- LoRA Configuration: The Low-Rank Adaptation (LoRA) configuration has been set up, specifying the parameters such as the rank, scaling factor, target modules, and dropout rate. This LoRA configuration has then been applied to specific layers of the pre-trained model.
- Training Setup and Execution: The training arguments, such as the number of epochs, learning rate, and batch size, have been defined. A Trainer object has

been set up using these arguments, and the fine-tuning process has been executed for the specified number of epochs.

- Model Saving: The fine-tuned model has been saved to a designated location, likely for future use or deployment.
- Text Generation Pipeline: A text generation pipeline has been created using the fine-tuned model and its corresponding tokenizer. This pipeline is then used to generate a response to a given prompt, which is printed to the console.