Assignment-2: Fine-Tuning a LLM for Domain-Specific Applications

Problem Statement: Develop a domain-specific language model by fine-tuning a pre-trained LLM. This model should effectively understand and generate text pertinent to the chosen domain, enhancing performance on specialized tasks.

Requirements:

1. Domain Selection:

Domain: Medical

Justification: The medical domain was selected due to the abundance of available data, its high relevance to real-world applications, and the significant potential impact on healthcare. Medical literature, clinical guidelines, and patient records offer a rich source of information for training the model.

2. Dataset Preparation:

A substantial dataset relevant to the medical domain was gathered from Hugging Face. This dataset includes medical articles, clinical reports, and patient records. The data was pre-processed and structured to ensure compatibility with the fine-tuning process.

3.Model Selection:

DeepSeek, a state-of-the-art LLM known for its robust performance and efficiency, was chosen as the base model for fine-tuning. DeepSeek's ability to understand the complexities of the medical domain made it a suitable choice.

4. Methodology

The fine-tuning process involved training the pre-trained DeepSeek model on the domain-specific medical dataset. Various hyperparameters, such as learning rate, batch size, and epochs, were adjusted to optimize the model's performance. LoRA (Low-Rank Adaptation) was used to efficiently fine tune the model.

Packages to be used for this project are:-

- unsloth: Efficient fine-tuning and inference for LLMs — Specifically we will be using:
    - FastLanguageModel module to optimize inference & fine-tuning
    - get_peft_model to enable LoRa (Low-Rank Adaptation) fine-tuning
- peft: Supports LoRA-based fine-tuning for large models.
- Different Hugging Face modules:
    - transformers from HuggingFace to work with our fine-tuning data and handle different model tasks
    - trl Transformer Reinforcement Learning from HuggingFace which allows for supervised fine-tuning of the model — we will use the SFFTrainer wrapper
    - datasets to fetch reasoning datasets from the Hugging Face Hub

- torch: Deep learning framework used for training
- wandb: Provides access to weights and biases for tracking our fine-tuning experiment.

Before starting we accessed the Hugging Face and Weights & Biases API
And Set the GPU accelerator.

Then Installed all the relevant packages.

Then Created API keys and login to Hugging Face and Weights and Biases.

Loading DeepSeek R1 and the Tokenizer

I loaded the DeepSeek R1 model and its tokenizer using FastLanguageModel.from_pretrained(). I also configured key parameters for efficient inference and fine-tuning. I'll be using a distilled 8B version of R1 for faster computation.

Intuition behind 4-bit quantization
Imagine compressing a **high-resolution image** to a smaller size—**it takes up less space but still looks good enough**. Similarly, **4-bit quantization reduces the precision of model weights**, making the model **smaller and faster while keeping most of its accuracy**. Instead of storing precise **32-bit or 16-bit numbers**, we compress them into **4-bit values**. This allows **large language models to run efficiently on consumer GPUs** without needing massive amounts of memory.

Testing DeepSeek R1 on a medical use-case before fine-tuning
Defining a system prompt
To create a prompt style for the model, I'll define a system prompt and include placeholders for the question and response generation. The prompt will guide the model to think step-by-step and provide a logical, accurate response.

Running inference on the model
In this step, I will **test the DeepSeek R1 model** by providing a **medical question** and generating a response.
The process involves the following steps:

1. **Define a test question** related to a medical case.
2. **Format the question using the structured prompt (prompt_style)** to ensure the model follows a logical reasoning process.
3. **Tokenize the input and move it to the GPU (cuda)** for faster inference.
4. **Generate a response using the model**, specifying key parameters like max_new_tokens=1200 (limits response length).
5. **Decode the output tokens back into text** to obtain the final readable answer.

Fine-tuning step by step

Step 1 — Update the system prompt
I will slightly change the prompt style for processing the dataset by adding the third placeholder for the complex chain of thought column. </think>

Step 2 — Download the fine-tuning dataset and format it for fine-tuning
I will use the Medical O1 Reasoninng SFT found here on [Hugging Face](). From the authors: This dataset is used to fine-tune HuatuoGPT-o1, a medical LLM designed for advanced medical reasoning. This dataset is constructed using GPT-4o, which searches for solutions to verifiable medical problems and validates them through a medical verifier

Next step is to structure the fine-tuning dataset according to train prompt style—why?

- Each question is paired with chain-of-thought reasoning and the final response.
- Ensures every training example follows a consistent pattern.
- Prevents the model from continuing beyond the expected response lengt by adding the EOS token.

Step 3 — Setting up the model using LoRA
An intuitive explanation of LoRA
Large language models (LLMs) have **millions or even billions of weights** that determine how they process and generate text. When fine-tuning a model, we usually update all these weights, which **requires massive computational resources and memory**.
LoRA (**Low-Rank Adaptation**) allows to fine-tune efficiently by:

- Instead of modifying all weights, **LoRA adds small, trainable adapters** to specific layers.
- These adapters **capture task-specific knowledge** while leaving the original model unchanged.
- This reduces the number of trainable parameters **by more than 90%**, making fine-tuning **faster and more memory-efficient**.

Think of an LLM as a **complex factory**. Instead of rebuilding the entire factory to produce a new product, LoRA **adds small, specialized tools** to existing machines. This allows the factory to adapt quickly **without disrupting its core structure**.

Step 4 — Model training!
This took me around 30 to 40 minutes — then check out our training results on Weights and Biases

Step 5 — Run model inference after fine-tuning