Make your first LLM - Step-by-Step Guide

Mini Project: Fine-Tuning GPT-2

Objective: Fine-tune GPT-2 for text generation on a domain-specific corpus, such as product reviews.

Steps:

1. **Dataset**: Collect product reviews (or any specialized text).
   - You can use publicly available datasets or scrape text data from the web.
2. **Set Up Environment**:
   - Install Hugging Face's `transformers` and `datasets` libraries.
   - Set up Google Colab for free GPU access, or use a local environment.
3. **Load GPT-2**:
   - Load the GPT-2 model and tokenizer from Hugging Face.
   - Tokenize the dataset and prepare for model input.
4. **Training**:
   - Fine-tune the model with your dataset.
   - Monitor loss and adjust hyperparameters (e.g., learning rate, batch size).
5. **Test Model**:
   - Generate new text based on your fine-tuned model.
6. **Deploy Model** (Optional):
   - Deploy the model using a web app (e.g., Flask, FastAPI) or as an API for generating text on demand.

Step 1: Install dependencies

Install necessary libraries

```
pip install transformers datasets
```

Step 2: Load the pre-trained Model

Load the GPT-2 model and its tokenizer from Hugging Face's Transformers library

```
from transformers import GPT2Tokenizer, GPT2LMHeadModel

model = GPT2LMHeadModel.from_pretrained('gpt2')
tokenizer = GPT2Tokenizer.from_pretrained('gpt2')
```

Step 3: Fine-tune the model on a new dataset (e.g., a custom text corpus)

Here, we'll use a sample dataset like `wikitext-2` as an example, but you can replace it with your own domain-specific dataset (e.g., product reviews).

```
from datasets import load_dataset

dataset = load_dataset('wikitext', 'wikitext-2-raw-v1',
split='train')

# Tokenize dataset for input into GPT-2
def tokenize_function(examples):
        return tokenizer(examples['text'], truncation=True,
padding=True)

tokenized_dataset = dataset.map(tokenize_function, batched=True)
```

Step 4: Train the model

We'll use Hugging Face's `Trainer` API to fine-tune the model. Adjust hyper-parameters like learning rate and batch size.

```
from transformers import Trainer, TrainingArguments

# Define training arguments
training_args = TrainingArguments(
    output_dir='./results',          # Directory to save results
    overwrite_output_dir=True,       # Overwrite output directory
    num_train_epochs=3,              # Number of epochs
    per_device_train_batch_size=4,   # Batch size
    save_steps=10_000,               # Save every 10,000 steps
    save_total_limit=2,              # Limit to 2 checkpoints
)

# Define trainer
```

```
trainer = Trainer(
    model=model,
    args=training_args,
    train_dataset=tokenized_dataset,
)

# Fine-tune the model
trainer.train()
```

Save the fine-tuned model.

```
model.save_pretrained('fine_tuned_gpt2')
tokenizer.save_pretrained('fine_tuned_gpt2')
```

Step 5: Test the model

Generate new text using your fine-tuned GPT-2 model.

```
# Load the fine-tuned model and tokenizer
model = GPT2LMHeadModel.from_pretrained('fine_tuned_gpt2')
tokenizer = GPT2Tokenizer.from_pretrained('fine_tuned_gpt2')

# Generate text from the fine-tuned model
input_text = "Once upon a time"
input_ids = tokenizer.encode(input_text, return_tensors='pt')

# Generate output text
generated_outputs = model.generate(input_ids, max_length=50,
num_return_sequences=1)

# Decode and print the generated text
print(tokenizer.decode(generated_outputs[0],
skip_special_tokens=True))
```