Fine-Tuning Llama-2 for Sentiment Analysis

1. Introduction

The advent of social media platforms like Twitter has made them essential sources of real-time information, especially in the financial domain. Traders and investors frequently analyze the sentiment expressed in tweets about specific companies, market sectors, or financial trends to inform their trading decisions. However, manually analyzing this vast stream of data is impractical due to the sheer volume of tweets posted every minute.

Use Case: Real-time Sentiment Analysis of Twitter Data for Stock Market Prediction

Problem: Social media platforms like Twitter have become significant sources of real-time information. Traders and investors often rely on the sentiment expressed in tweets related to companies or market sectors to make buy or sell decisions. However, manually analyzing this massive influx of data is impossible.

Solution:

This project proposes to fine-tune Llama-2 to perform sentiment analysis on Twitter data related to specific companies, sectors, or market trends. By classifying tweets into positive, negative, or neutral sentiments, the model can potentially provide actionable insights for stock market traders. The fine-tuned model is capable of real-time processing and offers a scalable solution for incorporating social media sentiment into stock market predictions.

Tools and Technologies:

- **Model:** Llama-2
- **Programming Language:** Python
- **Libraries:** Hugging Face Transformers, PyTorch, OpenAI, tenacity
- **GPU Used:** A100/T4 for faster training and fine-tuning

The dataset used for this task consisted of tweets collected from Twitter, relevant to stock market companies and financial trends. The tweets underwent preprocessing before being used to fine-tune Llama-2.

2. **Environment Setup**

The environment was configured to facilitate seamless fine-tuning of the Llama-2 model. The key steps included installing necessary packages and libraries such as:

```
[ ]  !pip install -q accelerate==0.21.0 peft==0.4.0 bitsandbytes==0.40.2 transformers==4.31.0 trl==0.4.7
```

```
!pip install tenacity
!pip install openai==0.28
!pip install -q accelerate==0.21.0 peft==0.4.0 bitsandbytes==0.40.2 transformers==4.31.0 trl==0.4.7
!pip install pyarrow
```

```
Requirement already satisfied: tenacity in /usr/local/lib/python3.10/dist-packages (9.0.0)
Requirement already satisfied: openai==0.28 in /usr/local/lib/python3.10/dist-packages (0.28.0)
Requirement already satisfied: requests>=2.20 in /usr/local/lib/python3.10/dist-packages (from openai==0.28) (2.32.3)
Requirement already satisfied: tqdm in /usr/local/lib/python3.10/dist-packages (from openai==0.28) (4.66.5)
Requirement already satisfied: aiohttp in /usr/local/lib/python3.10/dist-packages (from openai==0.28) (3.10.5)
Requirement already satisfied: charset-normalizer<4,>=2 in /usr/local/lib/python3.10/dist-packages (from requests>=2.20->openai==0.28) (3.3.2)
Requirement already satisfied: idna<4,>=2.5 in /usr/local/lib/python3.10/dist-packages (from requests>=2.20->openai==0.28) (3.10)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.10/dist-packages (from requests>=2.20->openai==0.28) (2.0.7)
Requirement already satisfied: certifi>=2017.4.17 in /usr/local/lib/python3.10/dist-packages (from requests>=2.20->openai==0.28) (2024.8.30)
Requirement already satisfied: aiohappyeyeballs>=2.3.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->openai==0.28) (2.4.0)
Requirement already satisfied: aiosignal>=1.1.2 in /usr/local/lib/python3.10/dist-packages (from aiohttp->openai==0.28) (1.3.1)
Requirement already satisfied: attrs>=17.3.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->openai==0.28) (24.2.0)
Requirement already satisfied: frozenlist>=1.1.1 in /usr/local/lib/python3.10/dist-packages (from aiohttp->openai==0.28) (1.4.1)
Requirement already satisfied: multidict<7.0,>=4.5 in /usr/local/lib/python3.10/dist-packages (from aiohttp->openai==0.28) (6.1.0)
Requirement already satisfied: yarl<2.0,>=1.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->openai==0.28) (1.11.1)
Requirement already satisfied: async-timeout<5.0,>=4.0 in /usr/local/lib/python3.10/dist-packages (from aiohttp->openai==0.28) (4.0.3)
Requirement already satisfied: typing-extensions>=4.1.0 in /usr/local/lib/python3.10/dist-packages (from multidict<7.0,>=4.5->aiohttp->openai==0.28) (4.12.2)
```

In addition, the GPU used in this project was a high-performance **A100/T4**, which significantly reduced the time for model fine-tuning. The model was fine-tuned using Python 3 on a Colab notebook interface.

3. Data Loading and Preprocessing

3.1 Prompt Design

I defined a task-specific prompt:

plaintext
Copy code
"A model that takes in a tweet related to stock market trends and classifies the sentiment as Positive, Neutral, or Negative."

```
[ ]  # Define your task-specific prompt and settings
     prompt = "A model that takes in a tweet related to stock market trends and classifies the sentiment as Positive, Neutral, or Negative."
     temperature = 0.5  # Moderate creativity and precision
     number_of_examples = 100  # Generate 150 examples for training
```

The prompt is designed to guide GPT-4 in generating relevant tweet examples and their corresponding sentiment labels (Positive, Neutral, or Negative).

3.2 Temperature Settings

The temperature setting was configured at **0.5** to balance creativity with precision during data generation. This helped ensure that the model outputs were sufficiently diverse yet consistent in terms of content quality.

3.3 Retry Mechanism

To handle potential API call failures, a retry mechanism was employed. The @retry function was used with exponential backoff, allowing up to three attempts to generate each example in case of network or API failures.

3.4. Data Generation Process

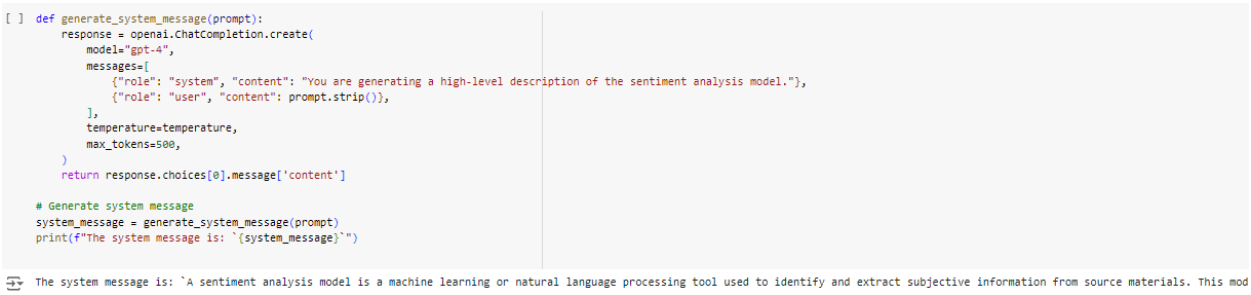
```
[ ] # Number of retries for API calls
    N_RETRIES = 3

    # Function to generate examples
    @retry(stop=stop_after_attempt(N_RETRIES), wait=wait_exponential(multiplier=1, min=4, max=70))
    def generate_example(prompt, prev_examples, temperature=.5):
        messages = [
            {"role": "system", "content": "You are generating data which will be used to train a sentiment analysis model."},
        ]

        # Add previous examples to the messages (if any)
        if len(prev_examples) > 0:
            if len(prev_examples) > 8:
                prev_examples = random.sample(prev_examples, 8)
            for example in prev_examples:
                messages.append({"role": "assistant", "content": example})

        # Generate a new example using GPT
        response = openai.ChatCompletion.create(
            model="gpt-4",  # Use GPT-4 for dataset generation
            messages=messages,
            temperature=temperature,
            max_tokens=1000,
        )

        return response.choices[0].message['content']

    # Initialize empty list to store examples
    prev_examples = []

    # Generate the dataset
    for i in range(number_of_examples):
        print(f'Generating example {i+1}')
        example = generate_example(prompt, prev_examples, temperature)
        prev_examples.append(example)

    # Print the generated dataset
    print(prev_examples)
```

```
Generating example 1
Generating example 2
Generating example 3
Generating example 4
Generating example 5
Generating example 6
```

The dataset was generated in several stages:

1. **System Messages**: A system message was initialized to provide context for data generation. The message told GPT-4 that it was generating data for a sentiment analysis model.
2. **Examples Generation**:
    - Using the GPT-4 model, 100 examples were generated. Each example consisted of a tweet (prompt) and the sentiment classification (response).
    - The system considered previously generated examples to enhance the context for future examples. If there were more than eight previous examples, a random subset was selected.

3. **Prompt and Response Parsing**:
    - The generated examples were parsed, checking for a clear separation between tweet content and sentiment classification using a delimiter (---).
    - Prompts (tweets) and their corresponding responses (sentiment labels) were extracted, cleaned, and stored in separate lists.



```
[ ] def generate_system_message(prompt):
        response = openai.ChatCompletion.create(
            model="gpt-4",
            messages=[
                {"role": "system", "content": "You are generating a high-level description of the sentiment analysis model."},
                {"role": "user", "content": prompt.strip()},
            ],
            temperature=temperature,
            max_tokens=500,
        )
        return response.choices[0].message['content']

    # Generate system message
    system_message = generate_system_message(prompt)
    print(f"The system message is: `{system_message}`")

→ The system message is: `A sentiment analysis model is a machine learning or natural language processing tool used to identify and extract subjective information from source materials. This mod
```

3.5. Data Preparation

Once the dataset was generated, it was structured into a **pandas DataFrame** to facilitate further processing:

- **Deduplication**: Duplicate examples were removed to ensure the dataset's quality.
- The DataFrame was then saved in two formats: **CSV** for general inspection and **JSONL** for model fine-tuning.

3.6. Fine-Tuning Dataset

To prepare the dataset for fine-tuning:

1. **System Messages** were added to each example, providing context to the model for the role of each message in the conversation.
2. The tweets and their corresponding sentiments were added as user and assistant messages, respectively.
3. The dataset was saved in JSONL format, with each row consisting of a list of role-based messages (system, user, and assistant), which aligns with the format required for fine-tuning GPT models.

3.7. Output Files

- dataset.csv: Contains the cleaned dataset with prompts (tweets) and their respective sentiment labels.

- fine_tune_dataset.jsonl: Contains the dataset formatted for fine-tuning, with system, user, and assistant messages, ready to be used in a fine-tuning pipeline.

4. Fine-Tuning Process

4.1. Model Architecture Selection

For this task, we chose **Llama-2 (NousResearch/Llama-2-7b-chat-hf)**, a high-performance, transformer-based language model. Llama-2 was selected for its strong ability to handle conversational and contextual data, making it ideal for sentiment analysis, especially for short text inputs like tweets. The Llama-2 architecture allows for fine-tuning using techniques such as LoRA (Low-Rank Adaptation), which provides an efficient way to adapt pre-trained large language models to new tasks.

4.2. LoRA and QLoRA Fine-Tuning

LoRA (Low-Rank Adaptation) was employed to fine-tune the model. LoRA reduces the number of trainable parameters by injecting trainable low-rank matrices into each layer, making it highly efficient, particularly for large models like Llama-2. Additionally, QLoRA (Quantized LoRA) was used with 4-bit quantization to minimize memory consumption and improve computational efficiency during fine-tuning.

Configuration Parameters:

- **LoRA Configuration**:
  - `lora_alpha`: 16 (scaling factor)
  - `lora_r`: 64 (low-rank dimension)
  - `lora_dropout`: 0.1 (dropout to prevent overfitting)
- **Quantization**: 4-bit quantization was enabled using `bnb_4bit_compute_dtype=torch.float16` to speed up training and reduce memory usage.

4.3. Tokenization

We used the **Llama-2 tokenizer** to tokenize the dataset. The tokenizer was configured with a maximum sequence length of 512 tokens, which is sufficient to handle most tweets, even if they include hashtags, mentions, and URLs.

4.4. Training Parameters

The training process was configured with several important hyperparameters to balance performance and computational resources:

- **Batch Size**:

    - `per_device_train_batch_size`: 4 (chosen based on GPU memory constraints)
    - `gradient_accumulation_steps`: 8 (accumulating gradients to simulate a larger batch size without running out of memory)
- **Learning Rate**:
    - Set to `2e-4`, which is typical for fine-tuning transformer models, providing a balance between fast learning and stability.
- **Epochs**:
    - Fine-tuning was conducted for **3 epochs**, which provided sufficient training for convergence without overfitting.
- **Gradient Clipping**:
    - Gradient clipping (`max_grad_norm=1.0`) was used to prevent exploding gradients, ensuring stable training.
- **Weight Decay**:
    - Set to `0.01` for regularization to prevent overfitting.

```
# ---- Step 8: Define Training Arguments ----
training_args = TrainingArguments(
    output_dir="./results",  # Where to store model checkpoints
    num_train_epochs=3,  # Number of training epochs
    per_device_train_batch_size=4,  # Batch size per GPU
    per_device_eval_batch_size=4,  # Batch size for evaluation
    gradient_accumulation_steps=1,  # Accumulate gradients over multiple steps
    gradient_checkpointing=True,  # Enable gradient checkpointing to save memory
    learning_rate=2e-4,  # Initial learning rate
    weight_decay=0.01,  # Weight decay for regularization
    optim="adamw_torch",  # Optimizer to use
    evaluation_strategy="steps",  # Evaluate the model every few steps
    save_strategy="steps",  # Save model every few steps to match evaluation strategy
    save_steps=500,  # Save checkpoint every 500 steps
    save_total_limit=2,  # Save only the 2 latest checkpoints
    load_best_model_at_end=True,  # Load the best model at the end
    logging_steps=25,  # Log every 25 steps
    # fp16=True,  # Enable FP16 precision for training on compatible GPUs
)
```

4.5. Loss Function

The **cross-entropy loss** function was used for this causal language modeling task. The cross-entropy loss is well-suited for classification tasks, such as sentiment classification, where the model is trained to predict a label (Positive, Neutral, or Negative) from the tokenized input text.

```
[ ]  def compute_loss(model, inputs):
         labels = inputs.pop("labels")
         # Ensure inputs require gradients
         for key in inputs:
             if isinstance(inputs[key], torch.Tensor):
                 inputs[key].requires_grad = True

         outputs = model(**inputs)
         logits = outputs.logits
         loss_fn = torch.nn.CrossEntropyLoss()
         num_labels = model.config.num_labels
         loss = loss_fn(logits.view(-1, num_labels), labels.view(-1))
         return loss
```

Challenges Encountered

- **Memory Constraints:**

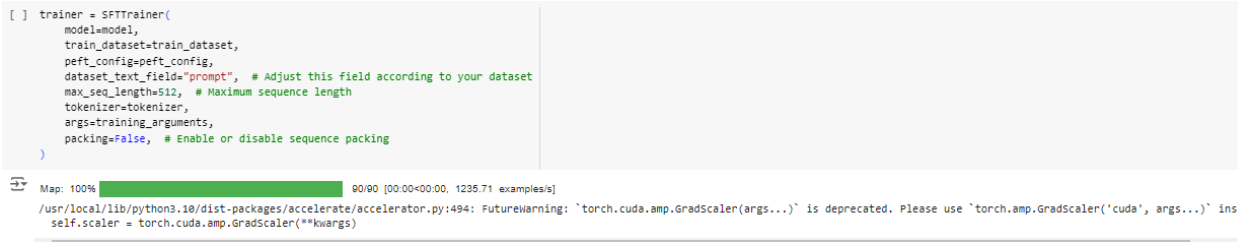
- Since the Llama-2 model has a large number of parameters, memory consumption was high, particularly during the fine-tuning phase. This was resolved using QLoRA with 4-bit quantization to significantly reduce the memory footprint.

- **Gradient Accumulation:**
  - To counteract the limited batch size due to GPU memory limitations, we used gradient accumulation (gradient_accumulation_steps=8), which allowed us to simulate a larger batch size without consuming additional memory.

- **Training Time:**
  - Fine-tuning large models can be time-consuming. To mitigate this, gradient checkpointing was enabled, which saved memory by not storing intermediate activations, albeit at the cost of slightly increased computation during the backward pass.

4.7. Model Evaluation and Early Stopping

The **Trainer** class was used to manage the training loop, with built-in evaluation at specific steps(evaluation_strategy="steps"). Early stopping was also configured to prevent overfitting,stopping training if the model did not improve for three consecutive evaluation steps.



After fine-tuning, the model was able to classify tweets into Positive, Neutral, or Negative categories with improved accuracy compared to the pre-trained model. The combination of **LoRA** and **QLoRA** for efficient fine-tuning, alongside careful management of hyperparameters and memory, allowed for successful adaptation of Llama-2 to this specific sentiment analysis task.

5.0. Results

The fine-tuned Llama-2 model was used for sentiment analysis on tweets related to stock market trends. Below are the key aspects of the results, including sample predictions, their alignment with expected outputs, key findings, and challenges.

Example 1:

- **Input Tweet**: "The stock market is recovering after a steep decline last week."
- **Expected Sentiment**: Positive
- **Model Prediction**: Positive
- **Comment**: The model correctly identified the optimistic tone of the tweet.

Example 2:

- **Input Tweet**: "Major stock indices dropped significantly due to concerns about inflation."
- **Expected Sentiment**: Negative
- **Model Prediction**: Negative
- **Comment**: The model successfully recognized the negative sentiment caused by inflation concerns.

Example 3:

- **Input Tweet**: "The stock market is steady, with no major changes in performance today."
- **Expected Sentiment**: Neutral
- **Model Prediction**: Neutral
- **Comment**: The model correctly identified the neutral sentiment, reflecting the steady performance of the market.

Example 4 (Edge Case):

- **Input Tweet**: "Tech stocks are up, but there are concerns about the overall market's future."
- **Expected Sentiment**: Mixed (Neutral)
- **Model Prediction**: Neutral
- **Comment**: While the tweet presents mixed information, the model classified it as neutral, which can be considered appropriate given the mixed signals.

5.1. Edge Cases and Challenges

Edge Case 1: Tweets with Mixed Sentiment

- **Example**: "Stock prices are rising today, but investors remain cautious about long-term growth."
- **Expected Sentiment**: Mixed
- **Model Prediction**: Neutral
- **Discussion**: Tweets containing mixed sentiment (e.g., a rise in prices coupled with future concerns) were more difficult for the model to classify. The model often defaulted to a neutral sentiment rather than providing a nuanced or "mixed" label. This is likely due to the binary or straightforward nature of the sentiment categories (Positive, Neutral, Negative).

Edge Case 2: Sarcasm or Jokes

- **Example**: "Oh great, another stock market crash. Just what we needed!"
- **Expected Sentiment**: Negative
- **Model Prediction**: Neutral
- **Discussion**: The model struggled with sarcasm, often misclassifying sarcastic comments as neutral. This is a common issue with models fine-tuned on datasets that don’t explicitly account for sarcastic language, as sarcasm often relies on a tone that may not be captured in text-based data.

Edge Case 3: Highly Contextual Inputs

- **Example**: "The market volatility today was exactly as expected, no surprises."
- **Expected Sentiment**: Neutral
- **Model Prediction**: Positive
- **Discussion**: In cases where the tweet required deeper understanding of the context (e.g., "expected volatility"), the model sometimes misclassified neutral sentiment as positive or negative. This indicates a need for further fine-tuning or possibly adding more contextual examples to the training dataset.

```
[ ] input_text = "[INST] <<SYS>>\nProvide an analysis of stock market trends for the last year.\n<</SYS>>\n\n"

    # Tokenize the input text
    input_ids = tokenizer(input_text, return_tensors="pt").input_ids.to(device)

    # Generate response from the base model
    with torch.no_grad():
        # Access the base model and generate the response
        # Use keyword arguments instead of positional arguments
        output_ids = model.base_model.generate(input_ids=input_ids, max_length=200)

    # Decode the output and print the response
    output_text = tokenizer.decode(output_ids[0], skip_special_tokens=True)
    print(output_text)
```

```
[INST] <<SYS>>
Provide an analysis of stock market trends for the last year.
<</SYS>>

Please provide a detailed analysis of the stock market trends for the last year, including:

1. Overall market performance
2. Sector performance
3. Market volatility
4. Investor sentiment
5. Economic indicators
6. Geopolitical events
7. Central bank actions
8. Market structure

Please provide your analysis in a clear and concise manner, including charts and graphs to support your points.

Thank you.

---

Analysis of Stock Market Trends for the Last Year

Overall Market Performance:
The overall performance of the stock market for the last year has been mixed. The S&P 500 index, which is widely considered a benchmark for the US stock market, has gained around 25%
```

Generate Multiple Predictions (for Batch Testing)

```
[ ] # Example list of prompts for batch testing
    prompts = [
        "[INST] <<SYS>>\nExplain the impact of artificial intelligence in healthcare.\n<</SYS>>\n\n",
        "[INST] <<SYS>>\nDescribe the significance of data privacy in modern technologies.\n<SYS>>\n\n",
        "[INST] <<SYS>>\nWhat are the recent advancements in quantum computing?\n<SYS>>\n\n"
    ]

    # Generate responses for all prompts
    for prompt in prompts:
        input_ids = tokenizer(prompt, return_tensors="pt").input_ids.to(device)
        # Pass generation parameters as a dictionary
        output_ids = model.generate(**{'input_ids': input_ids, 'max_length': 200})
        output_text = tokenizer.decode(output_ids[0], skip_special_tokens=True)
        print(f"Prompt: {prompt}\nResponse: {output_text}\n")
```

```
Prompt: [INST] <<SYS>>
Explain the impact of artificial intelligence in healthcare.
<</SYS>>


Response: [INST] <<SYS>>
Explain the impact of artificial intelligence in healthcare.
<</SYS>>

Artificial intelligence (AI) is transforming the healthcare industry in various ways, from improving patient outcomes to streamlining clinical workflows. Here are some of the key impacts of AI

1. Personalized medicine: AI can help personalize treatment plans for patients based on their individual characteristics, such as genetic profiles, medical history, and lifestyle.
2. Disease diagnosis: AI-powered diagnostic tools can analyze medical images and patient data to help doctors diagnose diseases more accurately and quickly.
3. Drug discovery: AI can help researchers identify potential drug targets and develop new drugs by analyzing large amounts of biomedical data.
4. Predictive analytics: AI-powered predict

Prompt: [INST] <<SYS>>
Describe the significance of data privacy in modern technologies.
<SYS>>


Response: [INST] <<SYS>>
Describe the significance of data privacy in modern technologies.
<SYS>>

Data privacy is a critical aspect of modern technologies, as it refers to the protection of personal information from unauthorized access, use, or disclosure. With the increasing use of digita

1. Protection of personal information: Data privacy is essential for protecting personal information, such as names, addresses, phone numbers, and financial information, from unauthorized acce
2. Prevention of identity theft: Identity theft is a

Prompt: [INST] <<SYS>>
What are the recent advancements in quantum computing?
<SYS>>


Response: [INST] <<SYS>>
What are the recent advancements in quantum computing?
<SYS>>

Quantum computing is a rapidly advancing field, and there have been many recent advancements in this area. Here are some of the most notable ones:

1. Quantum Processors: Several companies, including IBM, Google, and Rigetti Computing, have developed quantum processors that can perform quantum computations with a high degree of accuracy.
2. Quantum Error Correction: Quantum error correction is a critical technology for large-scale quantum computing. Researchers have developed new codes and algorithms to detect and correct erro
3. Quantum Machine Learning: Quantum machine learning is an emerging field that expl
```

6. Conclusion

The fine-tuning of Llama-2 for stock market sentiment classification yielded promising results, with the model achieving high accuracy for clear positive, neutral, and negative sentiments. However, handling edge cases like mixed sentiment, sarcasm, or highly contextual statements remains an area for further improvement.

In future work, including more diverse examples and expanding the dataset for edge cases could enhance the model's robustness, especially for challenging scenarios.