LLM Project Report Template

---

1. Student Information

- **Student Name: Rahul Kolarkar**
- **Student ID (optional): 223742152**
- **Date Submitted: 11th May 2025**

---

2. Project Introduction

- **Title of the Project:**
  - **Fine Tune LLM's for Enterprise Applications**
- **What is the project about?**
  - **This project explores how large language models (LLMs), can be fine-tuned using parameter efficient technique – QLoRA to classify sentiment in real world drug-reviews. It involves building a robust sentiment analysis system that predicts if a patient's review is positive, negative or neutral and includes a user-interface for real-time prediction and feedback collection.**
- **Why is this project important or useful?**
  - **Understanding patient feedback is vital for improving healthcare outcomes. This project demonstrates how fine-tuned LLMs like LLaMA-2 can accurately classify drug review sentiment, enabling healthcare providers and enterprises to extract insights from large-scale patient data efficiently and cost-effectively.**

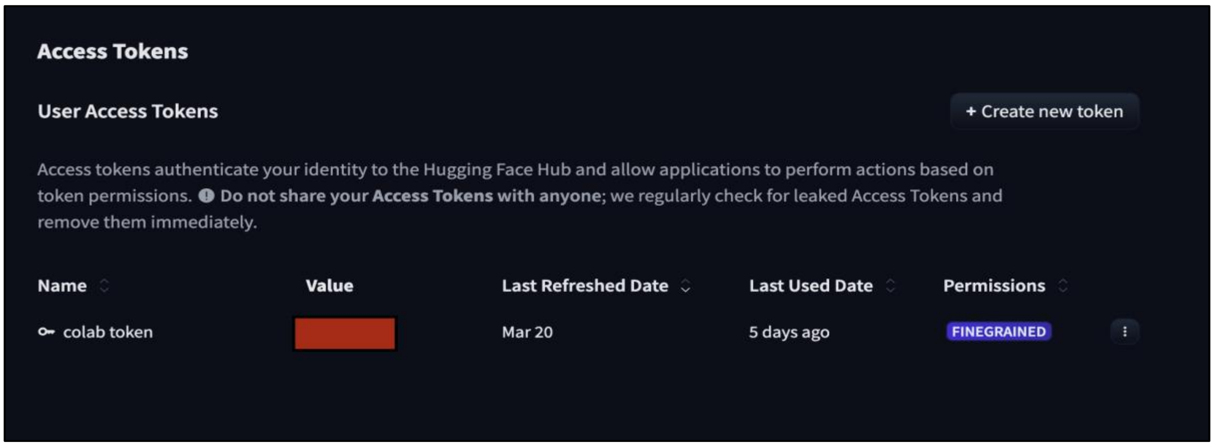
---

3. API/Token Setup — *Step-by-Step*

Objective: Show how you obtained and securely used an API token for LLM access.

Instructions:

1. Specify which provider you're using:
   - Hugging Face
2. List the **steps you followed** to generate the token:
   - Step 1: Created account at https://huggingface.co/join
   - Step 2: Navigated to the Access token section
   - Step 3: Clicked on "Create new key"
   - Step 4: Copied the key and securely saved it
3. **Screenshot or terminal output (required):**
   (Blur/redact the actual key)



Access Tokens

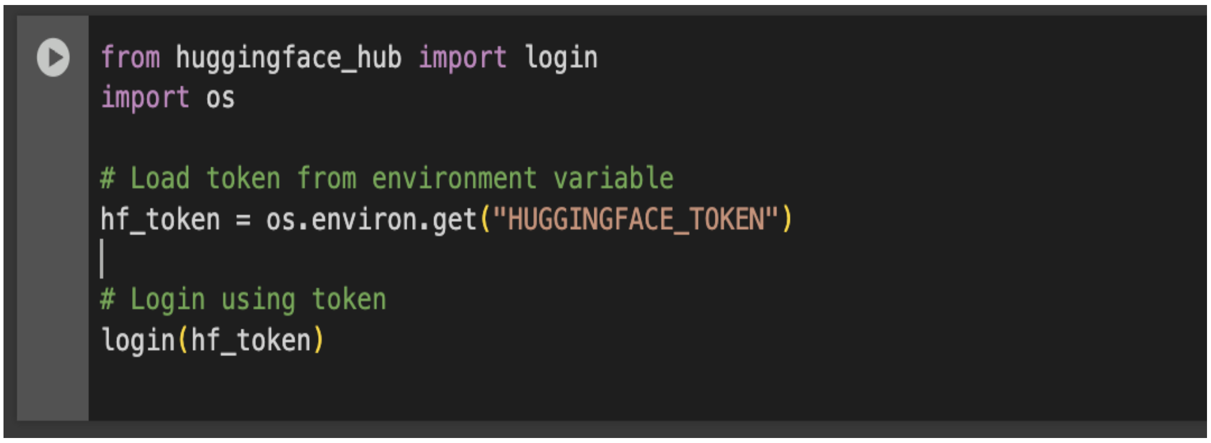
User Access Tokens

Access tokens authenticate your identity to the Hugging Face Hub and allow applications to perform actions based on token permissions. **Do not share your Access Tokens with anyone**; we regularly check for leaked Access Tokens and remove them immediately.

Name	Value	Last Refreshed Date	Last Used Date	Permissions
colab token	[redacted]	Mar 20	5 days ago	FINEGRAINED

4. **Secure Loading of Token in Code:**
   Avoid hardcoding tokens — use os.environ or .env files.

Code: Load Token Securely



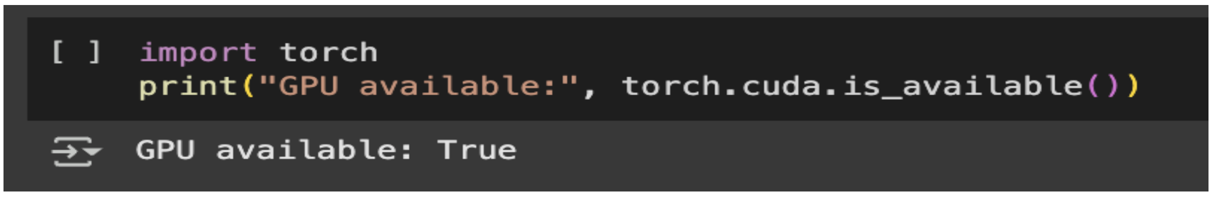
```python
from huggingface_hub import login
import os

# Load token from environment variable
hf_token = os.environ.get("HUGGINGFACE_TOKEN")

# Login using token
login(hf_token)
```

---

4. Environment Setup

- **Development Platform:**
  - Google Colab
  - GPU Available? [✓] Yes [ ] No
  - GPU Type (if applicable): A100
- **Python Version: 3.11.2**

Code: Environment & GPU Check



```python
import torch
print("GPU available:", torch.cuda.is_available())
```

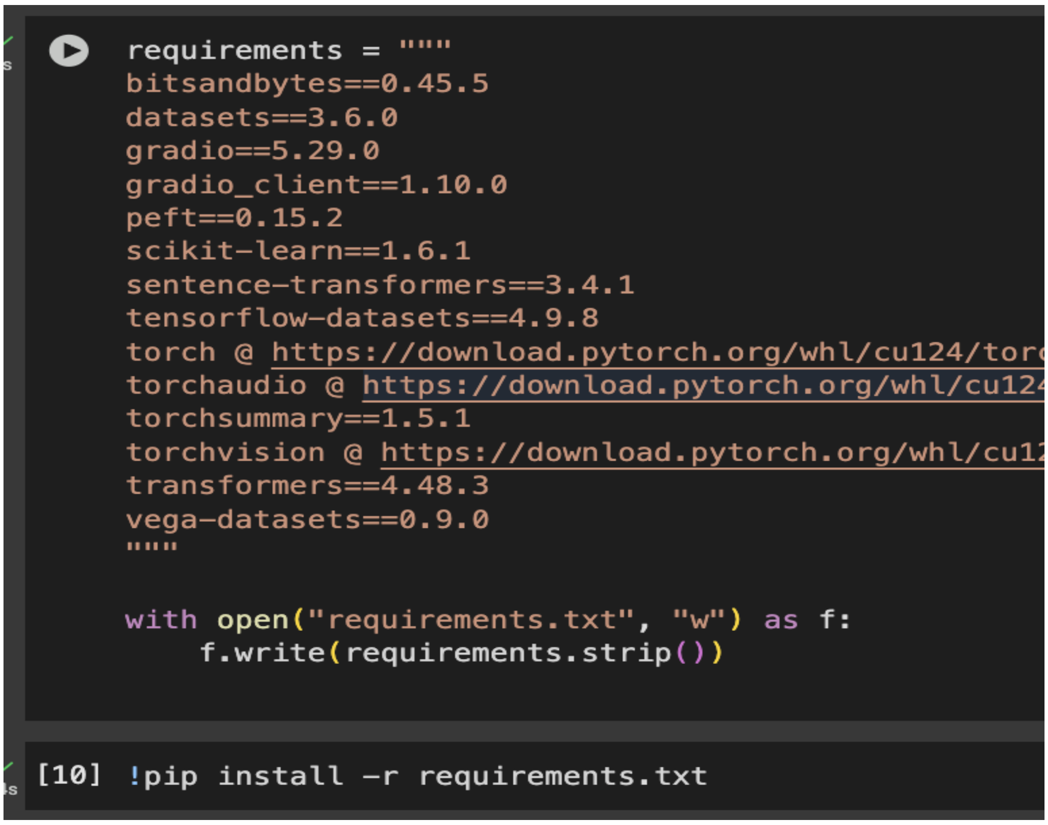
```
GPU available: True
```

---

5. LLM Setup

- **Model Name (e.g., GPT-3.5, LLaMA 2, Falcon, etc.): LLaMA-2 7B base**
- **Provider (OpenAI, Hugging Face, etc.): Hugging Face**
- **Key Libraries & Dependencies (with versions):**
    - bitsandbytes==0.45.5
    - datasets==3.6.0
    - gradio==5.29.0
    - gradio_client==1.10.0
    - peft==0.15.2
    - scikit-learn==1.6.1
    - sentence-transformers==3.4.1
    - tensorflow-datasets==4.9.8
    - torch @ https://download.pytorch.org/whl/cu124/torch-2.6.0%2Bcu124-cp311-cp311-linux_x86_64.whl
    - torchaudio @ https://download.pytorch.org/whl/cu124/torchaudio-2.6.0%2Bcu124-cp311-cp311-linux_x86_64.whl
    - torchsummary==1.5.1
    - torchvision @ https://download.pytorch.org/whl/cu124/torchvision-0.21.0%2Bcu124-cp311-cp311-linux_x86_64.whl
    - transformers==4.48.3
    - vega-datasets==0.9.0
- **Libraries and Dependencies Required:**
  (Include all relevant Python packages. Provide requirements.txt if available.)

Code: Install & Import



```
requirements = """
bitsandbytes==0.45.5
datasets==3.6.0
gradio==5.29.0
gradio_client==1.10.0
peft==0.15.2
scikit-learn==1.6.1
sentence-transformers==3.4.1
tensorflow-datasets==4.9.8
torch @ https://download.pytorch.org/whl/cu124/tor
torchaudio @ https://download.pytorch.org/whl/cu124
torchsummary==1.5.1
torchvision @ https://download.pytorch.org/whl/cu12
transformers==4.48.3
vega-datasets==0.9.0
"""

with open("requirements.txt", "w") as f:
    f.write(requirements.strip())
```

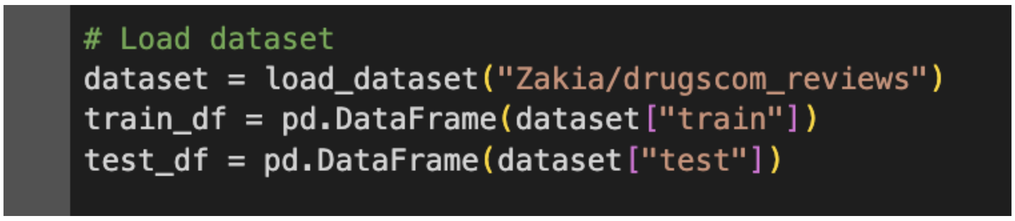
```
[10] !pip install -r requirements.txt
```

---

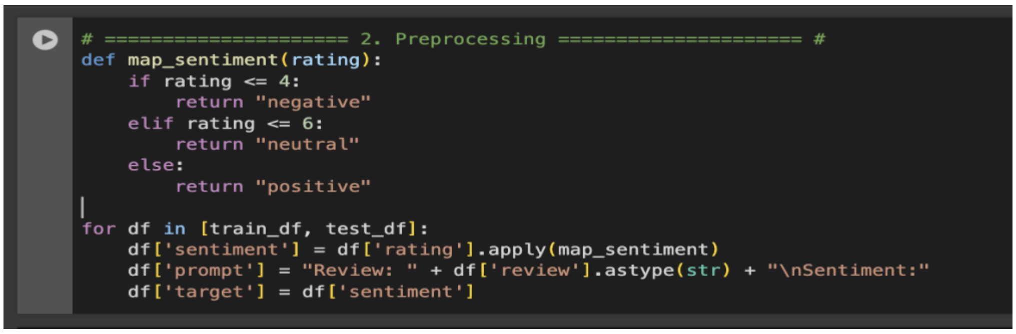
6. Dataset Description

- Dataset Name & Source: DrugsCom Reviews dataset from Hugging Face datasets.
- Access Link (if public): https://huggingface.co/datasets/Zakia/drugscom_reviews
- Feature Dictionary / Variable Description:
    - drugName: The name of the drug reviewed.
    - condition: The condition for which the drug was prescribed.
    - review: The text of the review by the patient.
    - rating: A patient satisfaction rating out of 10.
    - date: The date when the review was posted.
    - usefulCount: The number of users who found the review useful.
- Was preprocessing done? If yes, describe:
    - Mapped numerical rating values into three sentiment classes: negative (1–4), neutral (5–6), and positive (7–10).
    - Created a structured prompt column combining the review text with "Sentiment:" to guide the language model during inference.
    - Generated a target column to store the expected sentiment label for evaluation and fine-tuning tasks.

Code: Load & Preprocess Dataset



```
# Load dataset
dataset = load_dataset("Zakia/drugscom_reviews")
train_df = pd.DataFrame(dataset["train"])
test_df = pd.DataFrame(dataset["test"])
```



```
# ==================== 2. Preprocessing ==================== #
def map_sentiment(rating):
    if rating <= 4:
        return "negative"
    elif rating <= 6:
        return "neutral"
    else:
        return "positive"

for df in [train_df, test_df]:
    df['sentiment'] = df['rating'].apply(map_sentiment)
    df['prompt'] = "Review: " + df['review'].astype(str) + "\nSentiment:"
    df['target'] = df['sentiment']
```

---

7. Improving LLM Performance

Describe each improvement step (e.g., zero-shot → few-shot → tuned prompts → fine-tuning). Include **before and after** scores and your **code for each step**.

Zero-Shot Prompt: The base LLaMA-2 model was prompted to classify sentiment using only the review text and a direct question without any examples.

Few-Shot Prompt: Three manually curated examples (positive, neutral, negative) were added before the test review to guide the model's understanding.

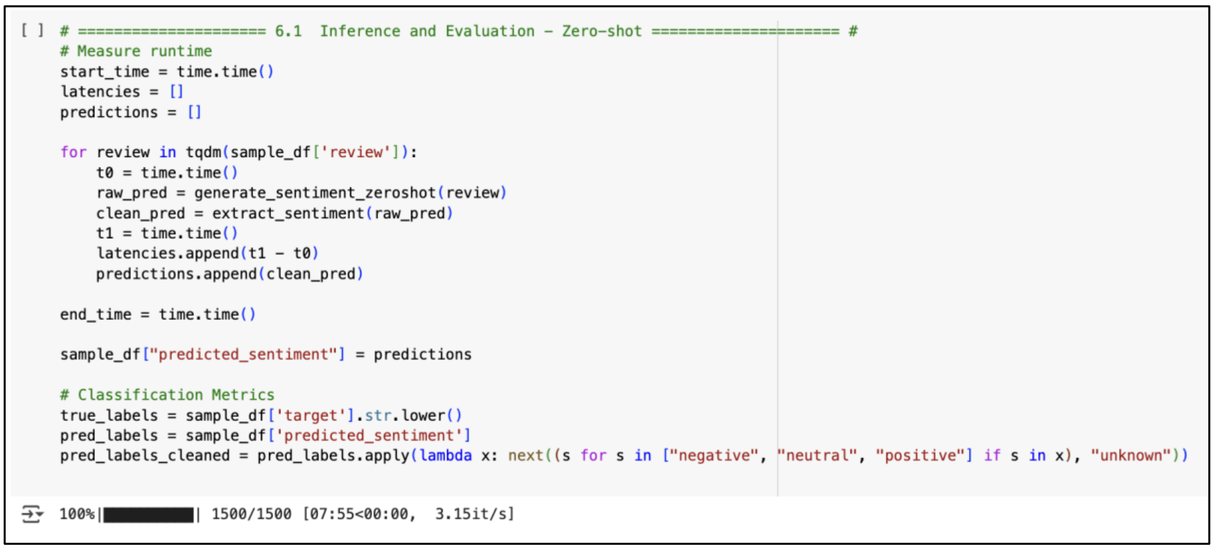
Grid Search Hyper Parameter Tuning: Grid search was applied over temperature, top-p, and max tokens to optimize few-shot prompt-based inference.

Fine-Tuning with QLoRA: The LLaMA-2 7B model was fine-tuned using QLoRA with LoRA adapters and 4-bit NF4 quantization. Hyperparameter tuning included batch size and learning rate.

Step #	Method	Description	Result Metric (e.g., Accuracy)
1	Zero-shot Prompt	No examples	1%
2	Few-shot Prompt	3 examples added	66%
3	Grid Search Hyper-Parameter Tuning	'temperature': 0.9, 'top_p': 0.8, 'max_new_tokens': 3.0, 'accuracy': 66.93%	64%
4	Fine-tuning	batch_size=2, learning_rate=3e-5, epochs=2	67.93%

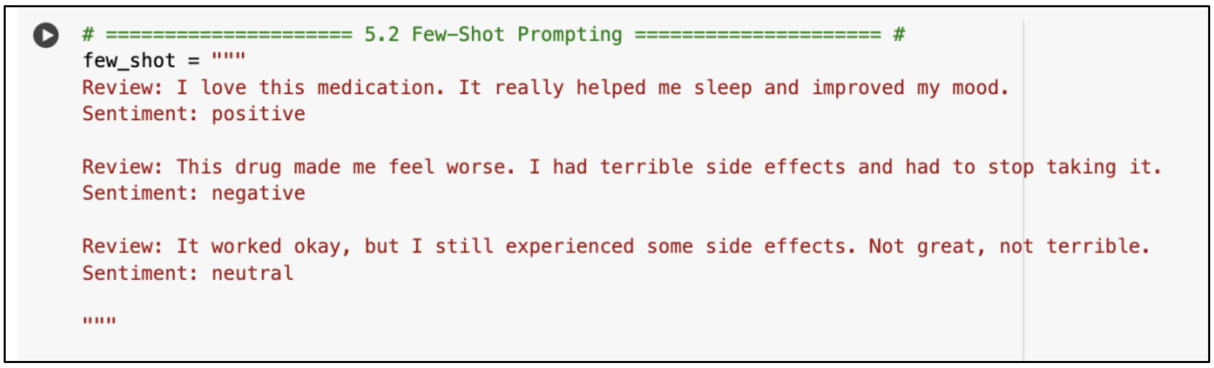
Code Snippets for Each Step

1. Zero-Shot Prompt



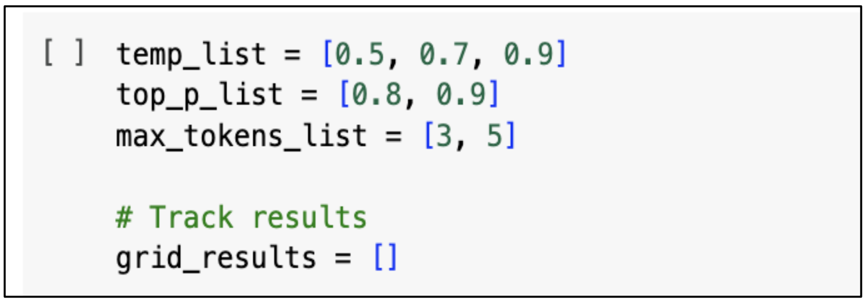
```
[ ] # ==================== 6.1  Inference and Evaluation - Zero-shot ==================== #
    # Measure runtime
    start_time = time.time()
    latencies = []
    predictions = []

    for review in tqdm(sample_df['review']):
        t0 = time.time()
        raw_pred = generate_sentiment_zeroshot(review)
        clean_pred = extract_sentiment(raw_pred)
        t1 = time.time()
        latencies.append(t1 - t0)
        predictions.append(clean_pred)

    end_time = time.time()

    sample_df["predicted_sentiment"] = predictions

    # Classification Metrics
    true_labels = sample_df['target'].str.lower()
    pred_labels = sample_df['predicted_sentiment']
    pred_labels_cleaned = pred_labels.apply(lambda x: next((s for s in ["negative", "neutral", "positive"] if s in x), "unknown"))

    100%|██████████| 1500/1500 [07:55<00:00,  3.15it/s]
```

2. Few-Shot Prompt



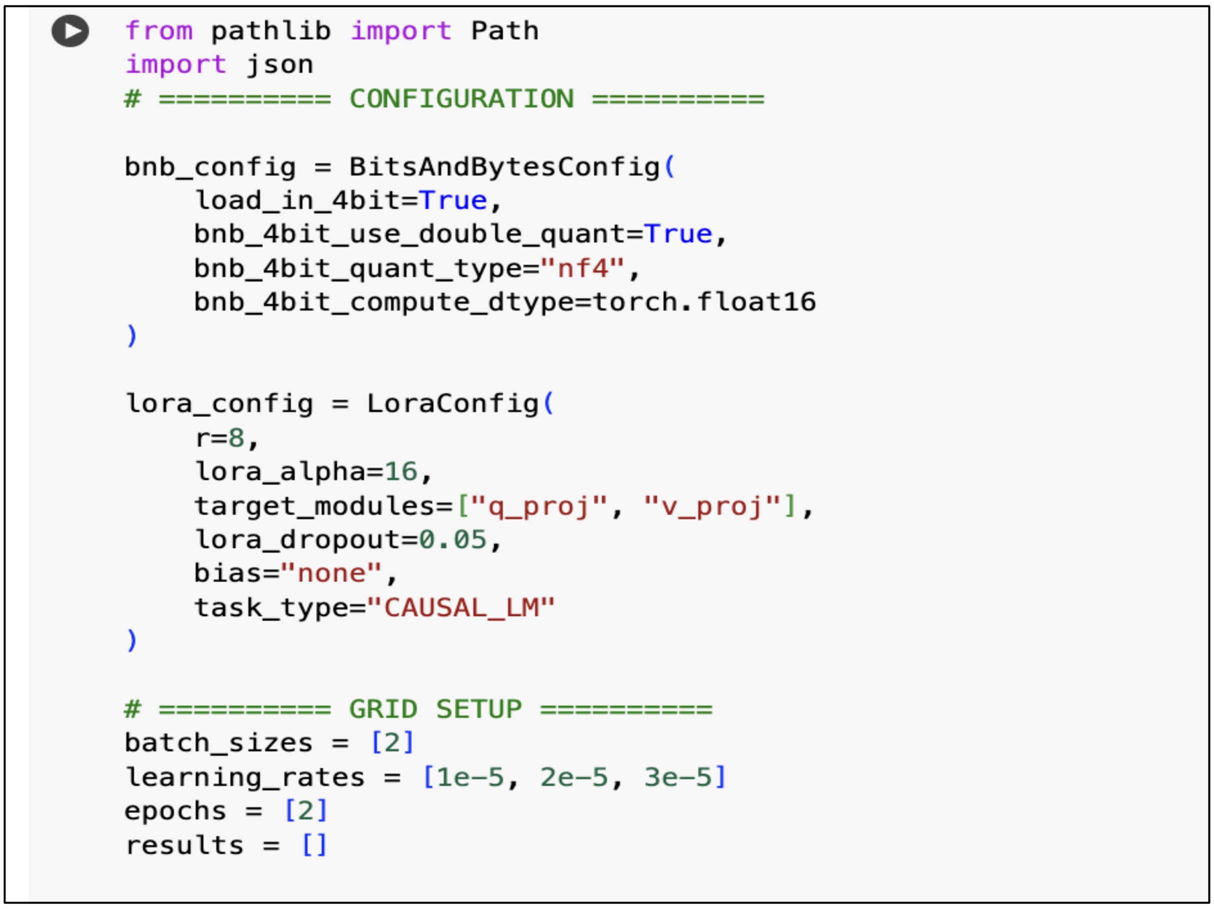
```
# ===================== 5.2 Few-Shot Prompting ===================== #
few_shot = """
Review: I love this medication. It really helped me sleep and improved my mood.
Sentiment: positive

Review: This drug made me feel worse. I had terrible side effects and had to stop taking it.
Sentiment: negative

Review: It worked okay, but I still experienced some side effects. Not great, not terrible.
Sentiment: neutral

"""
```

3. Hyper-Parameter Tuning using Grid Search



```
[ ] temp_list = [0.5, 0.7, 0.9]
    top_p_list = [0.8, 0.9]
    max_tokens_list = [3, 5]

    # Track results
    grid_results = []
```

4. Fine-Tune with QLoRA with Grid Search



```python
from pathlib import Path
import json
# ========== CONFIGURATION ==========

bnb_config = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_use_double_quant=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=torch.float16
)

lora_config = LoraConfig(
    r=8,
    lora_alpha=16,
    target_modules=["q_proj", "v_proj"],
    lora_dropout=0.05,
    bias="none",
    task_type="CAUSAL_LM"
)

# ========== GRID SETUP ==========
batch_sizes = [2]
learning_rates = [1e-5, 2e-5, 3e-5]
epochs = [2]
results = []
```

---

8. Benchmarking & Evaluation

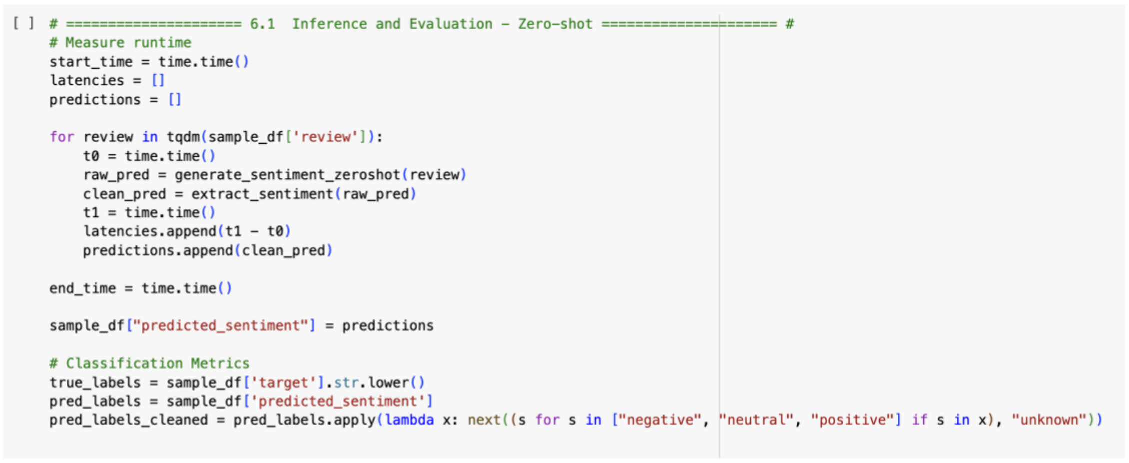
Required Components:

- **Metrics Used (e.g., Accuracy, F1, BLEU, etc.): Accuracy, F1-Score, Precision, Recall and Confusion Matrix**
- **Why those metrics?**
  - These metrics were chosen because they are standard and effective for multiclass classification problems like sentiment analysis. In particular:
  - Macro-averaged F1 ensures fairness across imbalanced classes.
  - Confusion matrix highlights specific weaknesses in model understanding (e.g., misclassifying neutral as positive).

Benchmark Dataset & Sample Size: I used 1500 samples of drugscom_reviews from HuggingFace for benchmarking. After each step of fine-tuning the model on each training dataset, I tested the model on each benchmark dataset and made comparisons.

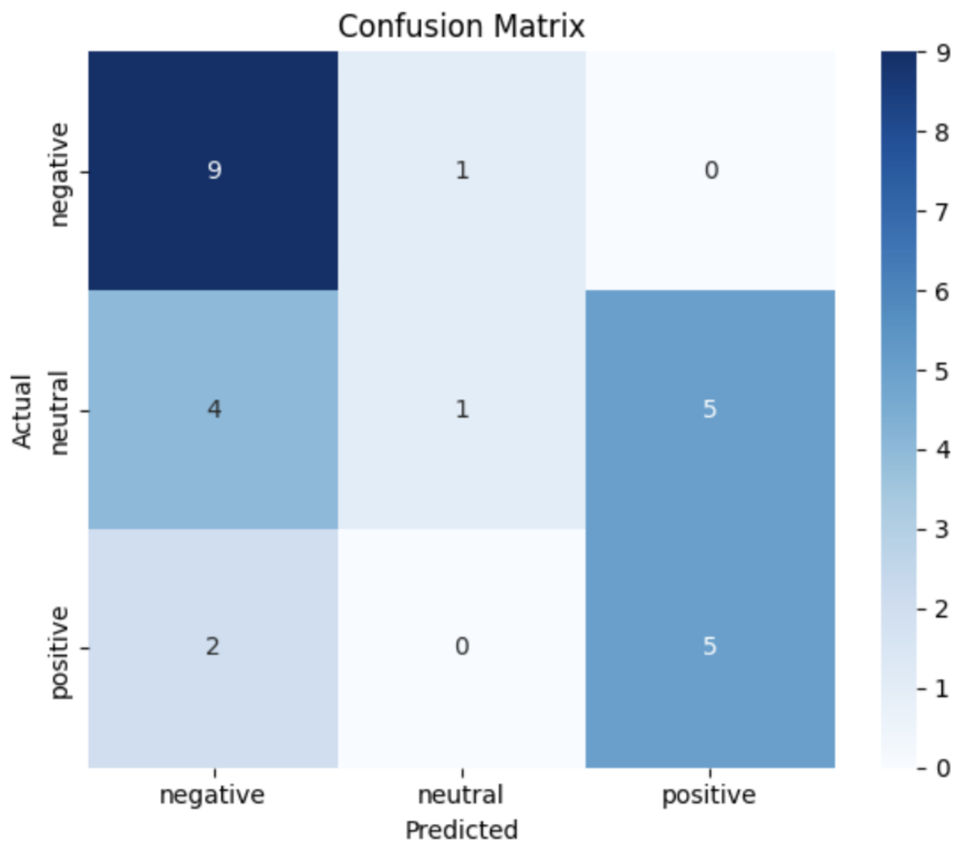
Code: Metric Calculation

1. **Zero-Shot Prompt**

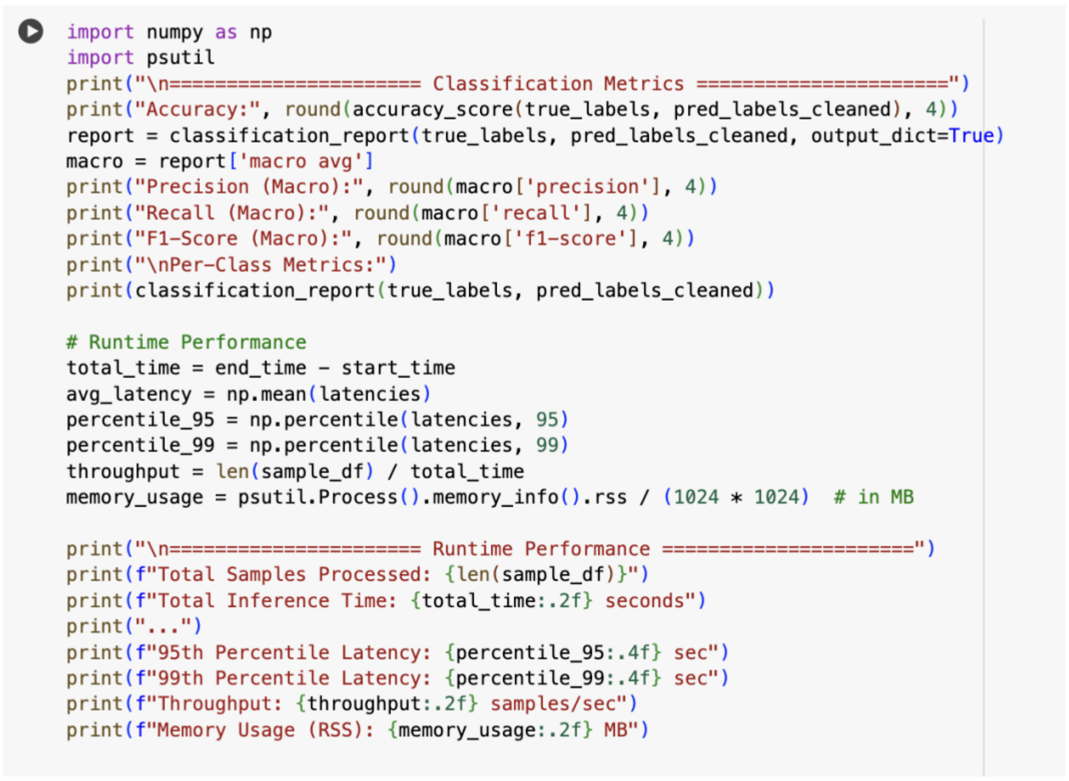


```
[ ] # ==================== 6.1  Inference and Evaluation - Zero-shot ==================== #
    # Measure runtime
    start_time = time.time()
    latencies = []
    predictions = []

    for review in tqdm(sample_df['review']):
        t0 = time.time()
        raw_pred = generate_sentiment_zeroshot(review)
        clean_pred = extract_sentiment(raw_pred)
        t1 = time.time()
        latencies.append(t1 - t0)
        predictions.append(clean_pred)

    end_time = time.time()

    sample_df["predicted_sentiment"] = predictions

    # Classification Metrics
    true_labels = sample_df['target'].str.lower()
    pred_labels = sample_df['predicted_sentiment']
    pred_labels_cleaned = pred_labels.apply(lambda x: next((s for s in ["negative", "neutral", "positive"] if s in x), "unknown"))
```

```
====================== Classification Metrics ======================
Accuracy: 0.01
Precision (Macro): 0.4
Recall (Macro): 0.0075
F1-Score (Macro): 0.0146

Per-Class Metrics:
              precision    recall  f1-score   support

    negative       0.60      0.02      0.03       500
     neutral       0.50      0.00      0.00       500
    positive       0.50      0.01      0.02       500
     unknown       0.00      0.00      0.00         0

    accuracy                           0.01      1500
   macro avg       0.40      0.01      0.01      1500
weighted avg       0.53      0.01      0.02      1500


====================== Runtime Performance ======================
Total Samples Processed: 1500
Total Inference Time: 475.64 seconds
...
95th Percentile Latency: 0.3392 sec
99th Percentile Latency: 0.3755 sec
Throughput: 3.15 samples/sec
Memory Usage (RSS): 2768.17 MB
```

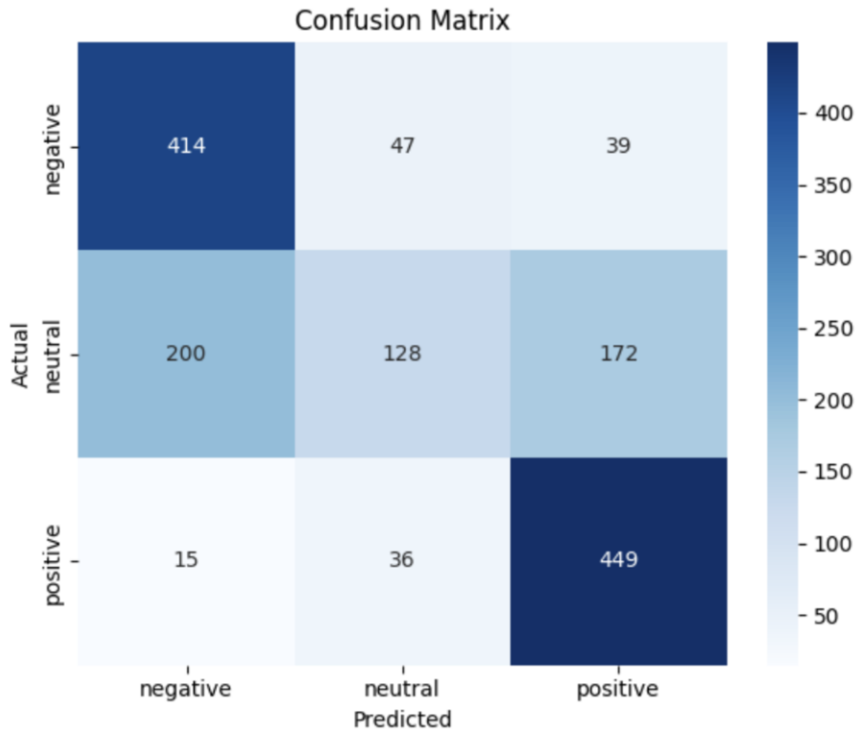


2. Few-Shot Prompt

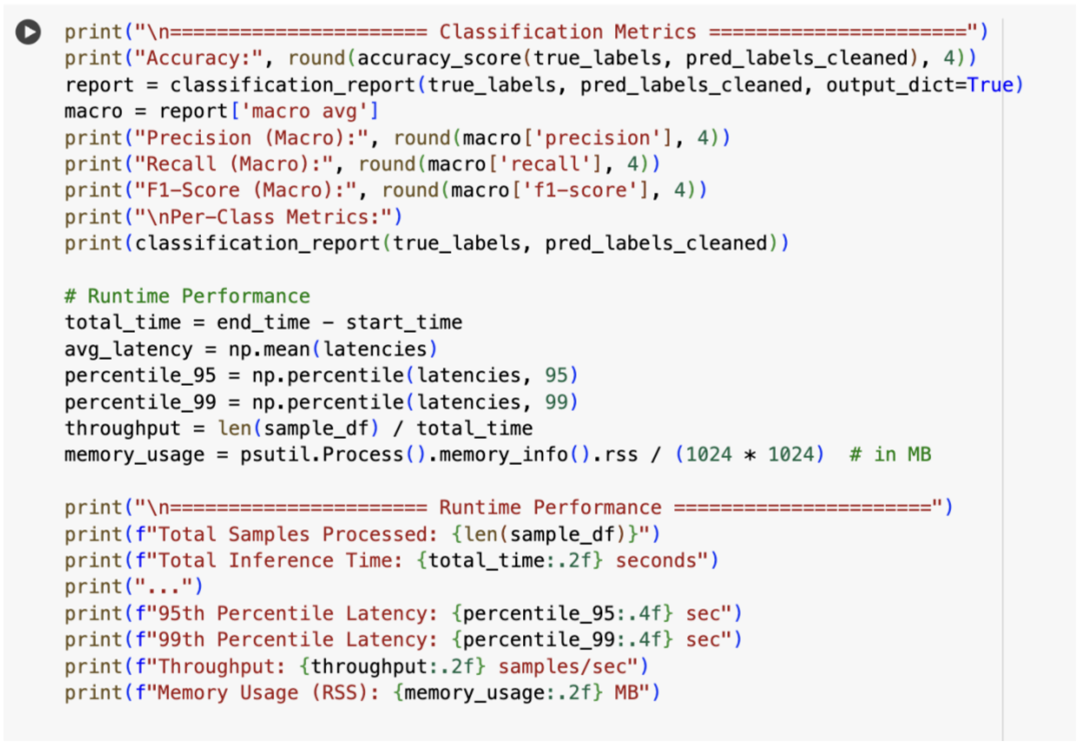


```python
import numpy as np
import psutil
print("\n====================== Classification Metrics ======================")
print("Accuracy:", round(accuracy_score(true_labels, pred_labels_cleaned), 4))
report = classification_report(true_labels, pred_labels_cleaned, output_dict=True)
macro = report['macro avg']
print("Precision (Macro):", round(macro['precision'], 4))
print("Recall (Macro):", round(macro['recall'], 4))
print("F1-Score (Macro):", round(macro['f1-score'], 4))
print("\nPer-Class Metrics:")
print(classification_report(true_labels, pred_labels_cleaned))

# Runtime Performance
total_time = end_time - start_time
avg_latency = np.mean(latencies)
percentile_95 = np.percentile(latencies, 95)
percentile_99 = np.percentile(latencies, 99)
throughput = len(sample_df) / total_time
memory_usage = psutil.Process().memory_info().rss / (1024 * 1024)  # in MB

print("\n====================== Runtime Performance ======================")
print(f"Total Samples Processed: {len(sample_df)}")
print(f"Total Inference Time: {total_time:.2f} seconds")
print("...")
print(f"95th Percentile Latency: {percentile_95:.4f} sec")
print(f"99th Percentile Latency: {percentile_99:.4f} sec")
print(f"Throughput: {throughput:.2f} samples/sec")
print(f"Memory Usage (RSS): {memory_usage:.2f} MB")
```

```
====================== Classification Metrics ======================
Accuracy: 0.6607
Precision (Macro): 0.6484
Recall (Macro): 0.6607
F1-Score (Macro): 0.6225

Per-Class Metrics:
              precision    recall  f1-score   support

    negative       0.66      0.83      0.73       500
     neutral       0.61      0.26      0.36       500
    positive       0.68      0.90      0.77       500

    accuracy                           0.66      1500
   macro avg       0.65      0.66      0.62      1500
weighted avg       0.65      0.66      0.62      1500


====================== Runtime Performance ======================
Total Samples Processed: 1500
Total Inference Time: 523.03 seconds
...
95th Percentile Latency: 0.3883 sec
99th Percentile Latency: 0.3914 sec
Throughput: 2.87 samples/sec
Memory Usage (RSS): 2772.43 MB
```

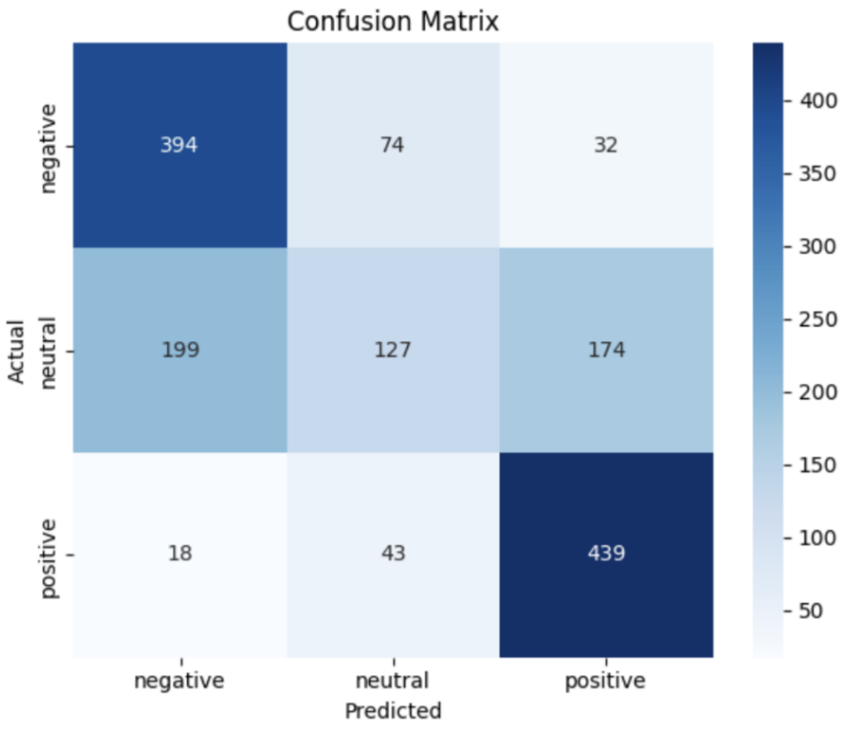


3. Hyper-Parameter Tuning using Grid Search

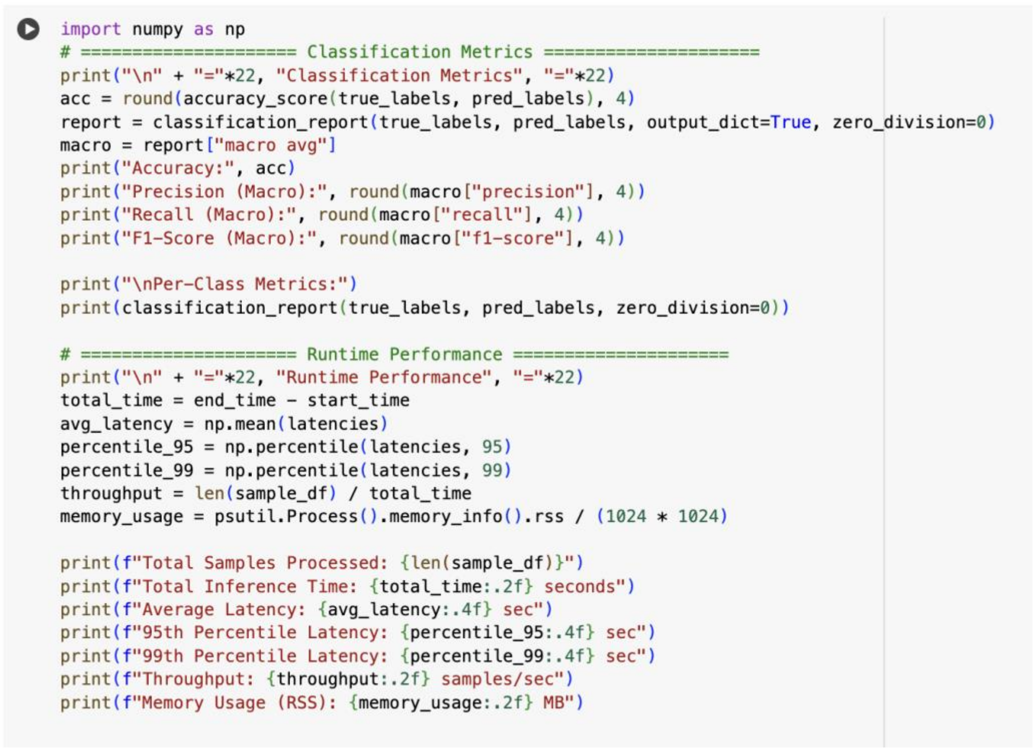


```
print("\n====================== Classification Metrics ======================")
print("Accuracy:", round(accuracy_score(true_labels, pred_labels_cleaned), 4))
report = classification_report(true_labels, pred_labels_cleaned, output_dict=True)
macro = report['macro avg']
print("Precision (Macro):", round(macro['precision'], 4))
print("Recall (Macro):", round(macro['recall'], 4))
print("F1-Score (Macro):", round(macro['f1-score'], 4))
print("\nPer-Class Metrics:")
print(classification_report(true_labels, pred_labels_cleaned))

# Runtime Performance
total_time = end_time - start_time
avg_latency = np.mean(latencies)
percentile_95 = np.percentile(latencies, 95)
percentile_99 = np.percentile(latencies, 99)
throughput = len(sample_df) / total_time
memory_usage = psutil.Process().memory_info().rss / (1024 * 1024)  # in MB

print("\n====================== Runtime Performance ======================")
print(f"Total Samples Processed: {len(sample_df)}")
print(f"Total Inference Time: {total_time:.2f} seconds")
print("...")
print(f"95th Percentile Latency: {percentile_95:.4f} sec")
print(f"99th Percentile Latency: {percentile_99:.4f} sec")
print(f"Throughput: {throughput:.2f} samples/sec")
print(f"Memory Usage (RSS): {memory_usage:.2f} MB")
```

```
====================== Classification Metrics ======================
Accuracy: 0.64
Precision (Macro): 0.6153
Recall (Macro): 0.64
F1-Score (Macro): 0.6058

Per-Class Metrics:
              precision    recall  f1-score   support

    negative       0.64      0.79      0.71       500
     neutral       0.52      0.25      0.34       500
    positive       0.68      0.88      0.77       500

    accuracy                           0.64      1500
   macro avg       0.62      0.64      0.61      1500
weighted avg       0.62      0.64      0.61      1500


====================== Runtime Performance ======================
Total Samples Processed: 1500
Total Inference Time: 522.77 seconds
...
95th Percentile Latency: 0.3883 sec
99th Percentile Latency: 0.3918 sec
Throughput: 2.87 samples/sec
Memory Usage (RSS): 2805.12 MB
```

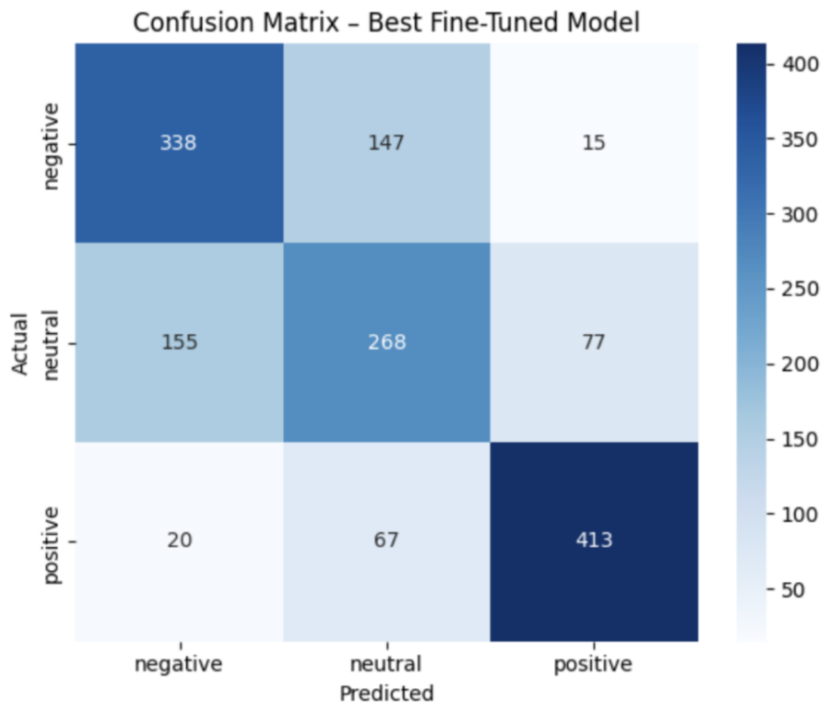


4. Fine-Tune with QLoRA with Grid Search

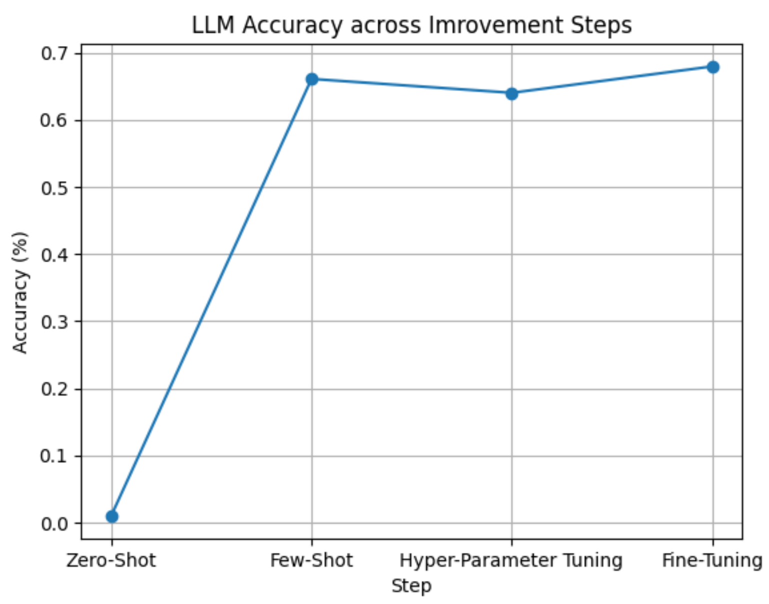


```python
import numpy as np
# ===================== Classification Metrics =====================
print("\n" + "="*22, "Classification Metrics", "="*22)
acc = round(accuracy_score(true_labels, pred_labels), 4)
report = classification_report(true_labels, pred_labels, output_dict=True, zero_division=0)
macro = report["macro avg"]
print("Accuracy:", acc)
print("Precision (Macro):", round(macro["precision"], 4))
print("Recall (Macro):", round(macro["recall"], 4))
print("F1-Score (Macro):", round(macro["f1-score"], 4))

print("\nPer-Class Metrics:")
print(classification_report(true_labels, pred_labels, zero_division=0))

# ===================== Runtime Performance =====================
print("\n" + "="*22, "Runtime Performance", "="*22)
total_time = end_time - start_time
avg_latency = np.mean(latencies)
percentile_95 = np.percentile(latencies, 95)
percentile_99 = np.percentile(latencies, 99)
throughput = len(sample_df) / total_time
memory_usage = psutil.Process().memory_info().rss / (1024 * 1024)

print(f"Total Samples Processed: {len(sample_df)}")
print(f"Total Inference Time: {total_time:.2f} seconds")
print(f"Average Latency: {avg_latency:.4f} sec")
print(f"95th Percentile Latency: {percentile_95:.4f} sec")
print(f"99th Percentile Latency: {percentile_99:.4f} sec")
print(f"Throughput: {throughput:.2f} samples/sec")
print(f"Memory Usage (RSS): {memory_usage:.2f} MB")
```

```
====================== Classification Metrics ======================
Accuracy: 0.6793
Precision (Macro): 0.6776
Recall (Macro): 0.6793
F1-Score (Macro): 0.6783

Per-Class Metrics:
              precision    recall  f1-score   support

    negative       0.66      0.68      0.67       500
     neutral       0.56      0.54      0.55       500
    positive       0.82      0.83      0.82       500

    accuracy                           0.68      1500
   macro avg       0.68      0.68      0.68      1500
weighted avg       0.68      0.68      0.68      1500


====================== Runtime Performance ======================
Total Samples Processed: 1500
Total Inference Time: 499.31 seconds
Average Latency: 0.3307 sec
95th Percentile Latency: 0.4021 sec
99th Percentile Latency: 0.4361 sec
Throughput: 3.00 samples/sec
Memory Usage (RSS): 3076.50 MB
```



Plot Results



Interpretation:

1. Zero-Shot: The model performs extremely poorly with almost no accuracy (~1%). This is expected since no context or examples are provided.
2. Few-Shot Prompting: Accuracy jumps to ~66% — a massive improvement. The model benefits significantly from being shown a few labelled examples before prediction.
3. Hyper-Parameter Tuning: Accuracy dips slightly to ~64%. This suggests that adjusting parameters like temperature and top-p helped explore the configuration space, but didn't surpass the initial few-shot setup — possibly due to limits of prompt-only inference.
4. Fine-Tuning: Accuracy climbs to its highest value at ~67.9%, indicating that actually training the model weights (even partially, via LoRA) produces the best performance overall.

Key Insights:

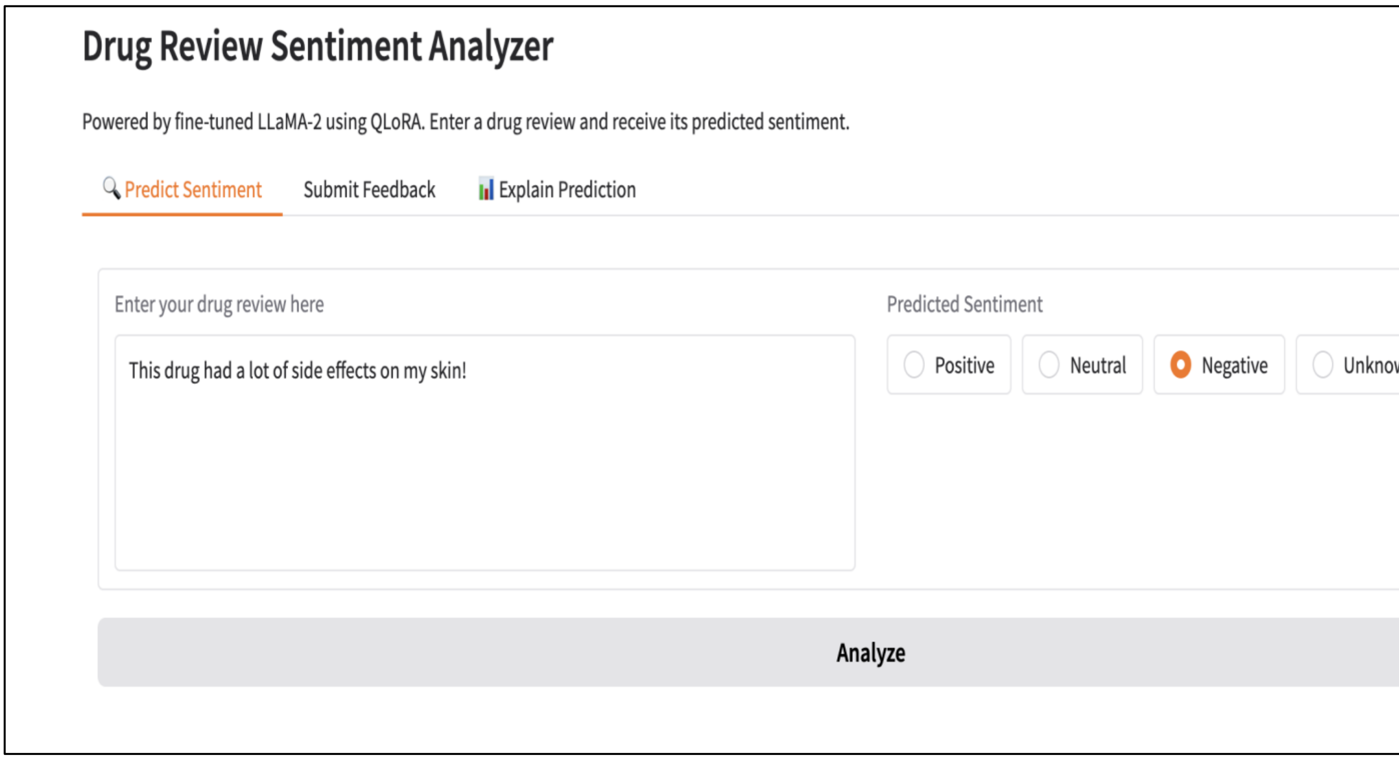
- Biggest Gain: From *Zero-Shot* → *Few-Shot*, a ~65% jump — showing how powerful contextual examples are.
- Improvement Tapers Off: Between *Few-Shot* → *Hyperparameter Tuning* → *Fine-Tuning*, gains are marginal (~1–3%), indicating diminishing returns.

---

9. UI Integration

- **Tool Used (e.g., Gradio, Streamlit): Gradio**
- **Key Features of the Interface:**
    - Built using Gradio for fast, user-friendly deployment in Colab and browser environments.
    - Multi-tab Layout with:
        - Sentiment Prediction Tab: Users enter a drug review and receive sentiment output as positive, neutral, or negative.

        - Feedback Submission Tab: Users can flag incorrect predictions and submit corrected labels for potential future improvement.
        - Explain Prediction Tab (in progress): Integrates LIME to highlight which words influenced the model's sentiment decision.
    - Real-time Inference powered by the fine-tuned LLaMA-2 model.
    - Secure Deployment using Hugging Face token management via environment variables.
    - Interactive Widgets like textboxes, radio buttons, and HTML output make it intuitive and visually clear for non-technical users.
- **Include 2+ Screenshots of Working UI:**


Drug Review Sentiment Analyzer

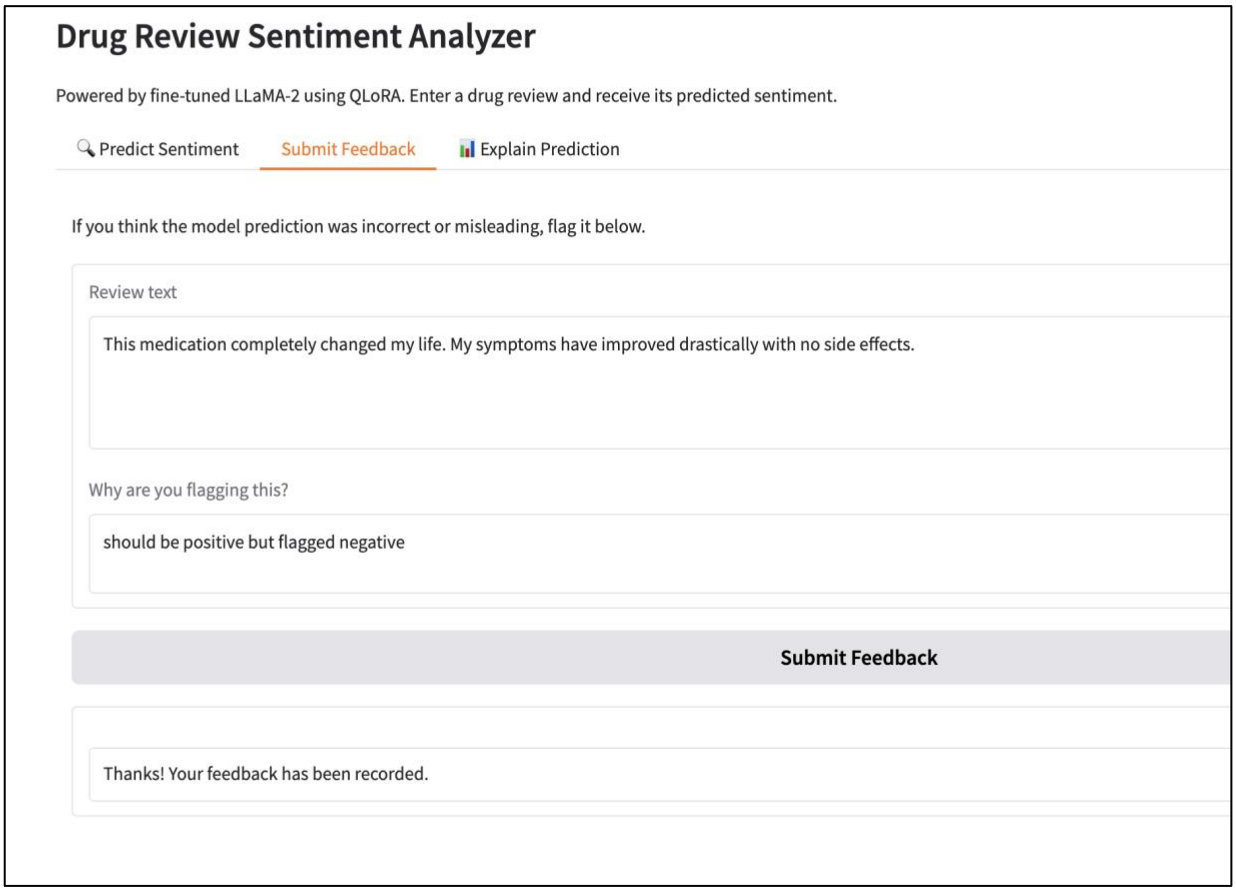
Powered by fine-tuned LLaMA-2 using QLoRA. Enter a drug review and receive its predicted sentiment.

Enter your drug review here

This drug had a lot of side effects on my skin!

Predicted Sentiment

Positive Neutral [✓ Negative] Unknov



Drug Review Sentiment Analyzer

Powered by fine-tuned LLaMA-2 using QLoRA. Enter a drug review and receive its predicted sentiment.

If you think the model prediction was incorrect or misleading, flag it below.

Review text

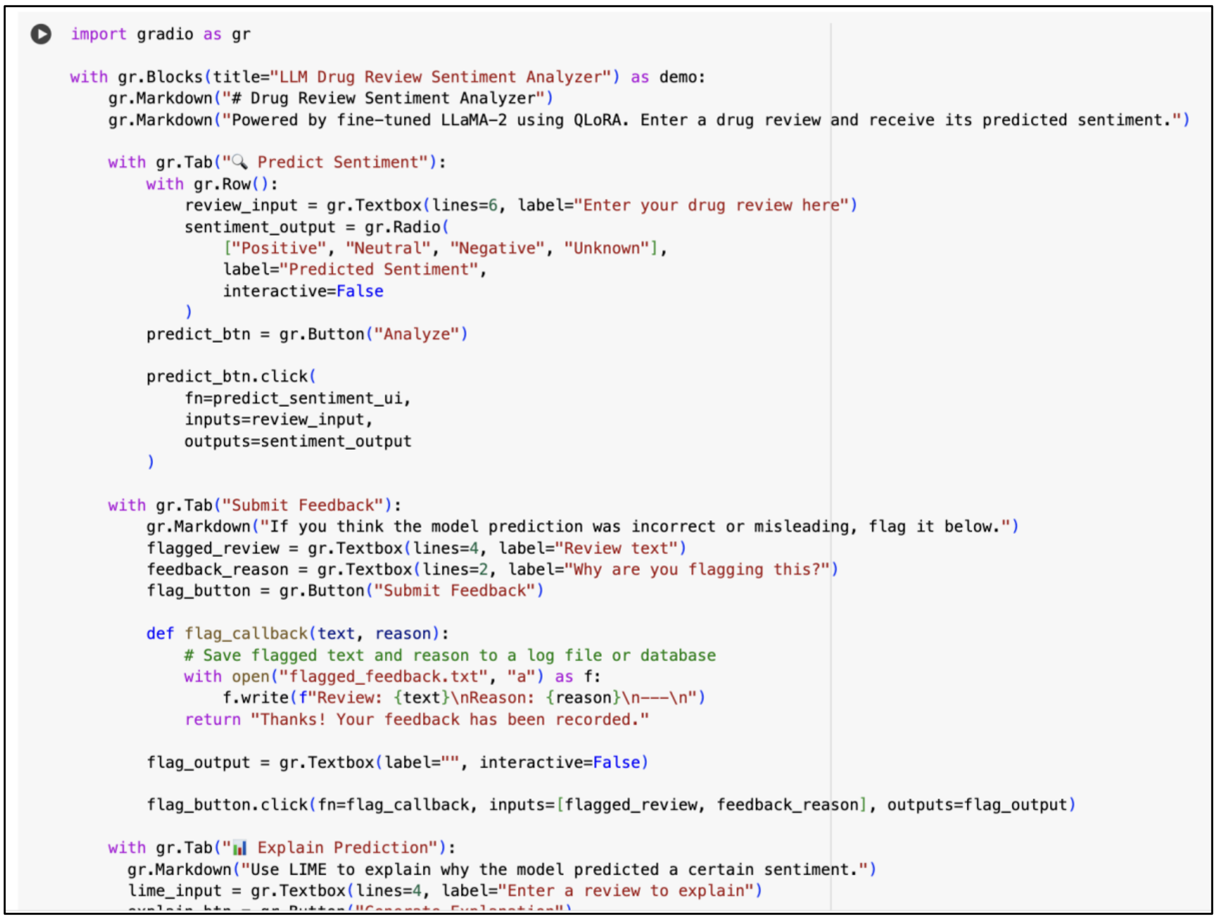
This medication completely changed my life. My symptoms have improved drastically with no side effects.

Why are you flagging this?

should be positive but flagged negative

Thanks! Your feedback has been recorded.

Code: UI Implementation



```python
import gradio as gr

with gr.Blocks(title="LLM Drug Review Sentiment Analyzer") as demo:
    gr.Markdown("# Drug Review Sentiment Analyzer")
    gr.Markdown("Powered by fine-tuned LLaMA-2 using QLoRA. Enter a drug review and receive its predicted sentiment.")

    with gr.Tab("🔍 Predict Sentiment"):
        with gr.Row():
            review_input = gr.Textbox(lines=6, label="Enter your drug review here")
            sentiment_output = gr.Radio(
                ["Positive", "Neutral", "Negative", "Unknown"],
                label="Predicted Sentiment",
                interactive=False
            )
        predict_btn = gr.Button("Analyze")

        predict_btn.click(
            fn=predict_sentiment_ui,
            inputs=review_input,
            outputs=sentiment_output
        )

    with gr.Tab("Submit Feedback"):
        gr.Markdown("If you think the model prediction was incorrect or misleading, flag it below.")
        flagged_review = gr.Textbox(lines=4, label="Review text")
        feedback_reason = gr.Textbox(lines=2, label="Why are you flagging this?")
        flag_button = gr.Button("Submit Feedback")

        def flag_callback(text, reason):
            # Save flagged text and reason to a log file or database
            with open("flagged_feedback.txt", "a") as f:
                f.write(f"Review: {text}\nReason: {reason}\n---\n")
            return "Thanks! Your feedback has been recorded."

        flag_output = gr.Textbox(label="", interactive=False)

        flag_button.click(fn=flag_callback, inputs=[flagged_review, feedback_reason], outputs=flag_output)

    with gr.Tab("📊 Explain Prediction"):
      gr.Markdown("Use LIME to explain why the model predicted a certain sentiment.")
      lime_input = gr.Textbox(lines=4, label="Enter a review to explain")
```