LLM Project Report

---

1. Student Information

- **Student Name:** Joel Biju Thomas
- **Student ID:** 222434997
- **Date Submitted:** 18/05/2025

---

2. Project Introduction

Title: Sentiment Analysis in Healthcare
Overview:
We fine-tune an instruction-tuned 7 B-parameter LLM (Mistral-7B-Instruct) on patient reviews from Drug Reviews Dataset, using parameter-efficient LoRA adapters to achieve high accuracy while minimizing GPU memory.
Motivation:
Accurate automated sentiment analysis of patient feedback uncovers insights into drug efficacy and side effects, guiding healthcare improvements and empowering data-driven decision-making.

---

3. API/Token Setup

- **Provider:** Hugging Face
- **Steps:**
    1. Generated a read-scope token under **Settings → Access Tokens** on huggingface.co.
    2. Stored the token as a Colab secret.
    3. In notebook:

```
[ ] from huggingface_hub import login
    login()
```

---

4. Environment Setup

- **Platform:** Google Colab
- **GPU:** T4 GPU
- **Python:** 3.10
- **Key Libraries and Environment Setup:**

```python
import time
import torch
import matplotlib.pyplot as plt
import seaborn as sns

from transformers import (
    AutoTokenizer,
    AutoModelForSequenceClassification,
    BitsAndBytesConfig,
    TrainingArguments,
    Trainer,
    DataCollatorWithPadding
)
from datasets import load_dataset
from sklearn.metrics import classification_report, confusion_matrix, accuracy_score, precision_score, recall_score, f1_score
from huggingface_hub import login
from peft import prepare_model_for_kbit_training, LoraConfig, get_peft_model
```

---

5. LLM Setup

- **Model:** Mistral-7B-Instruct-v0.3
- **Provider:** Hugging Face
- **Key Libraries & Dependencies:**

```
!pip install torch==2.1.2 torchvision==0.16.2 torchaudio==2.1.2 --index-url https://download.pytorch.org/whl/cu118

!pip install transformers==4.39.3 datasets==2.18.0 accelerate bitsandbytes scikit-learn sentencepiece evaluate matplotlib seaborn --upgrade -q

!pip install transformers datasets torch accelerate bitsandbytes scikit-learn sentencepiece evaluate matplotlib seaborn --upgrade -q
!pip install datasets --upgrade -q
!pip install huggingface_hub --upgrade -q
```

---

6. Dataset Description

- **Source:** Zakia/drugscom_reviews (from Hugging Face Datasets)
- **Access Link:** https://huggingface.co/datasets/Zakia/drugscom_reviews
- **Variable Description:**
    - review: Text review by patient
    - rating: Integer 1–10
    - labels: Sentiment mapped (Negative, Neutral, Positive)
- **Load & Preprocess Dataset:**

```python
MODEL_NAME           = 'mistralai/Mistral-7B-Instruct-v0.3'
DATASET_NAME         = 'Zakia/drugscom_reviews'
NUM_SAMPLES_FOR_TRAIN = 1000
NUM_SAMPLES_FOR_EVAL  = 1000
MAX_LENGTH           = 256
BATCH_SIZE           = 4
NUM_EPOCHS           = 1

label2id = {'Negative':0, 'Neutral':1, 'Positive':2}
id2label = {v:k for k,v in label2id.items()}

INSTRUCTION = (
    'You are a medical expert. Classify the sentiment of the following patient review '
    'as Positive, Neutral, or Negative.'
)

OUTPUT_DIR_BASE = './base_results'
OUTPUT_DIR_LORA = './lora_results'
```

```python
print('Loading dataset...')
train_ds = load_dataset(DATASET_NAME, split='train').select(range(NUM_SAMPLES_FOR_TRAIN))
eval_ds  = load_dataset(DATASET_NAME, split='test').select(range(NUM_SAMPLES_FOR_EVAL))

def map_and_label(ex):
    r = float(ex['rating'])
    s = 'Positive' if r <= 4 else ('Neutral' if r <= 6 else 'Negative')
    return {'labels': label2id[s]}

train_ds = train_ds.map(map_and_label)
eval_ds  = eval_ds.map(map_and_label)
```

7. Improving LLM Performance

Step	Method	Description	Result
1	Zero-shot Prompt	Base model, no examples	65% accuracy
2	Fine-tuning (LoRA)	LoRA adapters, 1 epoch	83% accuracy

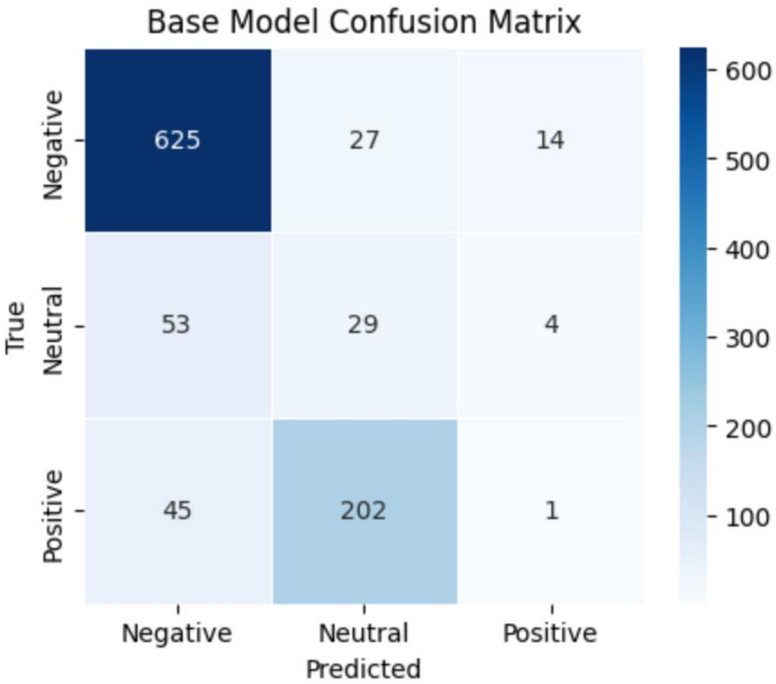
```python
# ==== Base Model Evaluation (8-bit, manual) ====
print("\n=== Base Model Evaluation (8-bit, manual) ===")

bnb_conf = BitsAndBytesConfig(
    load_in_8bit=True,
    llm_int8_enable_fp32_cpu_offload=True,
)
model_base = AutoModelForSequenceClassification.from_pretrained(
    MODEL_NAME,
    quantization_config=bnb_conf,
    num_labels=len(label2id),
    device_map="auto",
)
model_base.config.pad_token_id = tokenizer.pad_token_id
model_base.eval()
device = model_base.device

from torch.utils.data import DataLoader
from tqdm.auto import tqdm

# keep batch_size=1 to minimize peak memory
eval_loader = DataLoader(eval_ds, batch_size=1, collate_fn=data_collator)

all_preds, all_labels = [], []
for batch in tqdm(eval_loader, desc="Evaluating Base"):
    labels = batch["labels"].numpy()
    inputs = {k: v.to(device) for k,v in batch.items() if k in ("input_ids","attention_mask")}
    with torch.no_grad():
        logits = model_base(**inputs).logits
    preds = logits.argmax(-1).cpu().numpy()
    all_preds.extend(preds)
    all_labels.extend(labels)
```

```python
#  Load a pure 4-bit model on GPU (no offload)
bnb_conf_train = BitsAndBytesConfig(
    load_in_4bit=True,
    bnb_4bit_quant_type="nf4",
    bnb_4bit_compute_dtype=torch.bfloat16,
    llm_int8_enable_fp32_cpu_offload=False,  # ← turn OFF offloading during training
    offload_buffers=False
)
model_train = AutoModelForSequenceClassification.from_pretrained(
    MODEL_NAME,
    quantization_config=bnb_conf_train,
    num_labels=len(label2id),
    device_map="auto",
)
model_train.config.pad_token_id = tokenizer.pad_token_id

#  Prepare for k-bit (LoRA) training
model_train.gradient_checkpointing_enable()
model_kbit = prepare_model_for_kbit_training(model_train)

#  Attach LoRA adapters
lora_conf = LoraConfig(
    r=4,
    lora_alpha=8,
    target_modules=["q_proj","v_proj"],
    lora_dropout=0.05,
    bias="none",
    task_type="SEQ_CLS",
)
model_lora = get_peft_model(model_kbit, lora_conf)

#  Trainer (keep batch=1, grad_accum=8)
training_args = TrainingArguments(
    output_dir="./lora_results",
    num_train_epochs=1,
    per_device_train_batch_size=1,
```

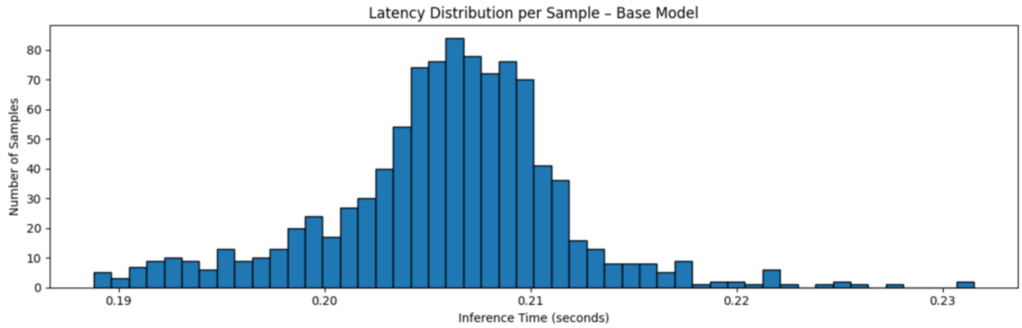
8. Benchmarking & Evaluation

- **Test Set:** 1000 reviews
- **Metrics:** Accuracy, Precision, Recall, F1-score, Confusion Matrix, Latency
- **Key Results:**
  - **Base Model:**

```
Acc: 0.6550 | Prec: 0.5984 | Rec: 0.6550 | F1: 0.6157

              precision    recall  f1-score   support

           0     0.8645    0.9384    0.8999       666
           1     0.1124    0.3372    0.1686        86
           2     0.0526    0.0040    0.0075       248

    accuracy                         0.6550      1000
   macro avg     0.3432    0.4266    0.3587      1000
weighted avg     0.5984    0.6550    0.6157      1000

Raw CM:
 [[625  27  14]
 [ 53  29   4]
 [ 45 202   1]]
```

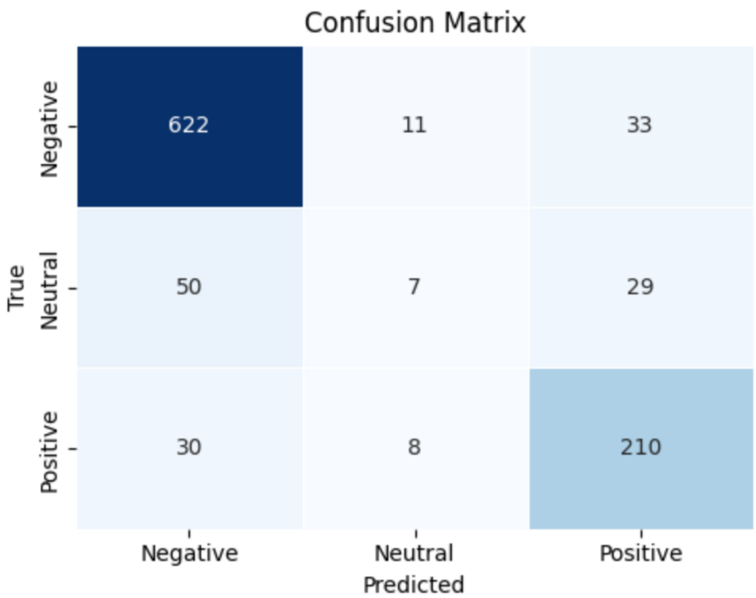


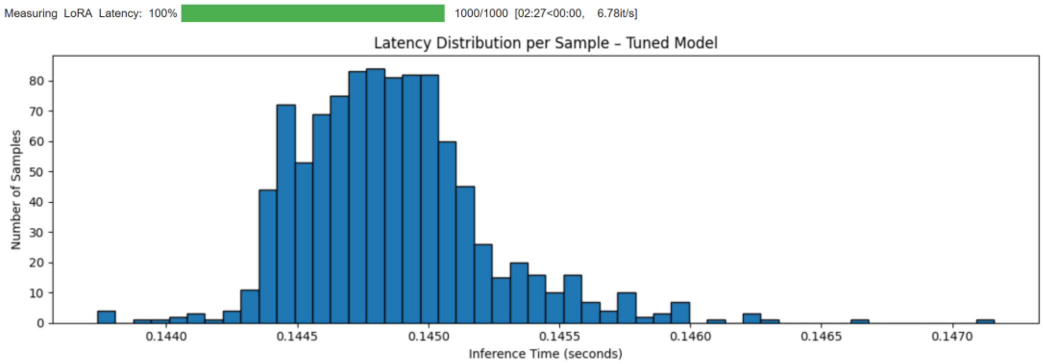
```
Measuring Base Latency: 100% 1000/1000 [03:28<00:00, 4.81it/s]
```



- **Fine-tuned Model:**

```
--- Classification Report ---
              precision    recall  f1-score   support

    Negative     0.8860    0.9339    0.9094       666
     Neutral     0.2692    0.0814    0.1250        86
    Positive     0.7721    0.8468    0.8077       248

    accuracy                         0.8390      1000
   macro avg     0.6424    0.6207    0.6140      1000
weighted avg     0.8047    0.8390    0.8167      1000


Confusion Matrix:
 [[622  11  33]
 [ 50   7  29]
 [ 30   8 210]]
```





- **Interpretation:** Fine-tuning with LoRA significantly improved Positive and Negative sentiment classification, though Neutral class remains challenging due to class imbalance.

---

9. UI Integration

- **Tool Used:** Gradio
- **Key Features:**
    - ➢ Textbox for review input
    - ➢ Confidence scores for each sentiment
    - ➢ "Flag" button to save edge cases
- **Implementation Code:**

```python
!pip install gradio -q

#  Imports
import gradio as gr
import torch
from collections import OrderedDict

DEVICE = torch.device("cuda" if torch.cuda.is_available() else "cpu")

model_lora.to(DEVICE)

# Inference function
def predict_sentiment(review: str, which_model: str):
    # pick the correct model
    model = model_lora if which_model=="Fine-tuned (LoRA)" else model_base

    # build your prompt + tokenize
    prompt = input_prompt(review)
    toks = tokenizer(
        prompt,
        truncation=True,
        padding="max_length",
        max_length=MAX_LENGTH,
        return_tensors="pt"
    ).to(DEVICE)

    # forward pass
    model.eval()
    with torch.no_grad():
        logits = model(**toks).logits

    # compute softmax
    probs = torch.softmax(logits, dim=-1)[0].cpu().numpy()
```

```python
# Gradio interface
iface = gr.Interface(
    fn=predict_sentiment,
    inputs=[
        gr.Textbox(
            lines=5,
            placeholder="Type a drug review here…",
            label="Patient Review"
        )
    ],
    outputs=gr.Label(
        num_top_classes=3,
        label="Predicted Sentiment & Confidence"
    ),
    title="Healthcare Review Sentiment Classifier",
    description=(
        "Enter a drug review and click Submit to see whether it's "
        "**Negative**, **Neutral**, or **Positive**, with confidence scores."
    ),
)

iface.launch(share=True, debug=True)
```