LLM Project Report

---

1. Student Information

- **Student Name:** Thi Hong Tham Nguyen (Shin)
- **Student ID (optional):** 224349871
- **Date Submitted:** 17/05/2025

---

2. Project Introduction

- **Title of the Project:** Fine-Tuning Large Language Models in Enterprise Healthcare Applications
- **What is the project about?**

  The rapid advancement of Large Language Models (LLMs) has led to their widespread adoption in enterprise applications, including customer support, content generation, and data analysis. However, general-purpose LLMs often lack domain-specific knowledge and may not meet the specialized needs of enterprises. This project aims to equip students with hands-on experience in finetuning LLMs to improve their performance for targeted enterprise use cases. Utilizing techniques such as Supervised Fine-Tuning (SFT) and Low-Rank Adaptation (LoRA), students will develop custom datasets and leverage opensource tools from GitHub (e.g., Unsloth) to fine-tune a model. By evaluating performance improvements post-fine-tuning, students will demonstrate the effectiveness of their approach and contribute to the growing field of AI customization for businesses.

- **Why is this project important or useful?**

  This project is important and useful because it allows students to develop practical skills in fine-tuning Large Language Models (LLMs) for enterprise-specific applications, reducing the gap between general-purpose AI and customized business solutions. By learning techniques like Supervised Fine-Tuning (SFT) and Low-Rank Adaptation (LoRA), students gain experience in adapting AI models to meet the unique needs of various industries, improving their performance, relevance, and efficiency.

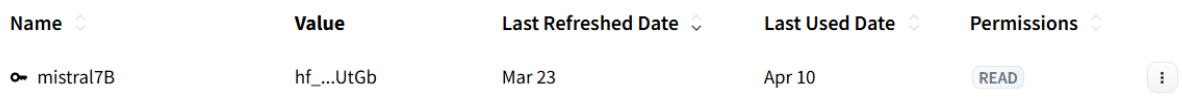
---

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://____ Hugging Face – The AI community building the future. _____
    - Step 2: Clicked on Settings -> Access Tokens
    - Step 3: Clicked on "Create new token" -> Type your Token name
    - Step 4: Copied the key and securely saved it
3. **Screenshot or terminal output (required):**
   (Blur/redact the actual key)



Name	Value	Last Refreshed Date	Last Used Date	Permissions
mistral7B	hf_...UtGb	Mar 23	Apr 10	READ

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

Code: Load Token Securely

```
[4] from huggingface_hub import login
    import os
    from dotenv import load_dotenv

    load_dotenv("/content/drive/MyDrive/shin_colab/tokens.env")
    HUGGINGFACE_API_KEY = os.getenv("HUGGINGFACE_API_KEY")
    # Replace 'your_token' with the token you copied from your Hugging Face account
    login(token=HUGGINGFACE_API_KEY)
```

---

4. Environment Setup

- **Development Platform:**
    - Google Colab
    - Local Machine (Windows)
    - GPU Available? [ ] Yes [**x**] No

- **Python Version: 3.11.11**

Code: Environment & GPU Check

```
[8]  import torch
     print("GPU Available: ", torch.cuda.is_available())
     print("GPU Name: ", torch.cuda.get_device_name())
```

```
GPU Available:  True
GPU Name:  Tesla T4
```

---

5. LLM Setup

- **Model Name (e.g., GPT-3.5, LLaMA 2, Falcon, etc.): Mistral_7B**
- **Provider (OpenAI, Hugging Face, etc.): Hugging Face**
- **Key Libraries & Dependencies (with versions)**
- **Libraries and Dependencies Required:**
  (Include all relevant Python packages. Provide requirements.txt if available.)

Code: Install & Import

Install

```
[2]  !pip -q install git+https://github.com/huggingface/transformers
     !pip -q install accelerate xformers einops
     !pip install -U bitsandbytes
     !pip install python-dotenv
```

Import

```
[6]  import torch
     import transformers
     from transformers import AutoTokenizer, AutoModelForCausalLM
     import pandas as pd
     import matplotlib.pyplot as plt
     import time
     import psutil
     from tqdm import tqdm # For progress tracking
     import numpy as np
     from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, classification_report
     import seaborn as sns
```

---

6. Dataset Description

- **Dataset Name & Source:** drugscom_reviews
- **Access Link (if public):** Zakia/drugscom_reviews · Datasets at Hugging Face
- **Feature Dictionary / Variable Description:**

Variable Name	Data Type	Description
drugname	string	The name of the drug reviewed.
condition	string	The condition for which the drug was prescribed.
review	string	The text of the review by the patient.
rating	Integer (0-10)	A patient satisfaction rating out of 10.
date	date	The date when the review was posted.
usefulCount	integer	The number of users who found the review useful.

- **Was preprocessing done? If yes, describe: Yes**
  - ✓ Loading the Dataset: The training and testing datasets were loaded into train_df and test_df DataFrames.
  - ✓ Sentiment Classification:
    - A new column named Sentiment was created in both training and testing datasets (train_df and test_df).
    - The sentiment classification was based on the values in the rating column, using a custom function named sentiment:
      - Ratings between 1 and 4 were classified as “Negative”.
      - Ratings between 5 and 6 were classified as “Neutral”.
      - Ratings between 7 and 10 were classified as “Positive”.
    - This function was applied using the apply() method on the rating column of both DataFrames.

Code: Load & Preprocess Dataset

```
[ ]  # - Create new column named “Sentiment” which based on the rating. The rule is: 1-4: Negative, 5-7: Neutral, 8-10: Positive

     # Firstly, define a function to classify sentiment based on rating
     def sentiment(rating):
       if rating >= 1 and rating <= 4:
         return 'Negative'
       elif rating >= 5 and rating <=6:
         return 'Neutral'
       elif rating >= 7 and rating <= 10:
         return 'Positive'

     # Apply the function to the 'rating' column to create a new 'Sentiment' column
     train_df['Sentiment'] = train_df['rating'].apply(sentiment)
     test_df['Sentiment'] = test_df['rating'].apply(sentiment)

     print("Train Dataset:")
     print(train_df[['rating', 'Sentiment']].head())

     print("Test_Dataset:")
     print(test_df[['rating', 'Sentiment']].head())
```

---

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**.

Step #	Method	Description	Result Metric (e.g., Accuracy)
1	Zero-shot Prompt	No examples	58%
2	Few-shot Prompt	3 examples added	59%
3	Temperature Tuning	Temp = 0.7	59%
4	Fine-tuning	2 epochs on 1500 samples (500 each class)	66%

Code Snippets for Each Step

- Function for experimenting:

```
# Process each row
def run_experiment(test_df, type='few_shot', temperature=0.5):
  test_df = test_df.copy()
  test_df['Predict_sentiment'] = None
```

- Step 1: Zero-shot Prompt

```
run_experiment(test_df, type='zero_shot')
```

```
100%|██████████| 500/500 [13:27<00:00,  1.62s/it]
```

- Step 2: Few-shot Prompt

```
run_experiment(test_df, type='few_shot')
```

```
100%|██████████| 500/500 [13:44<00:00,  1.65s/it]
```

- Step 3: Temperature Tuning – Temp = 0.7

```
[ ] run_experiment(test_df, type='few_shot', temperature=0.7)
```

```
100%|██████████| 500/500 [13:48<00:00,  1.66s/it]
```

- Step 4: Fine-Tuning - 2 epochs on 1500 samples (500 each class)

```
[ ] from trl import SFTTrainer
    from transformers import TrainingArguments
    from unsloth import is_bfloat16_supported

    # Setting up an SFTTrainer from the Hugging Face PEFT (Parameter-Efficient Fine-Tuning)
    trainer = SFTTrainer(
        model = model,
        tokenizer = tokenizer,
        train_dataset = dataset,
        dataset_text_field = "text",
        max_seq_length = max_seq_length,
        dataset_num_proc = 2,
        packing = False, # Can make training 5x faster for short sequences.
        # Training hyperparameters
        args = TrainingArguments(
            per_device_train_batch_size = 4,
            gradient_accumulation_steps = 4,
            warmup_steps = 5,
            # max_steps = 60,
            num_train_epochs = 2,
            learning_rate = 2e-4,
            fp16 = not is_bfloat16_supported(),
            bf16 = is_bfloat16_supported(),
            logging_steps = 1,
            optim = "adamw_8bit",
            weight_decay = 0.01,
            lr_scheduler_type = "linear",
            seed = 3407,
            output_dir = "/content/drive/MyDrive/shin_colab/ft_mistral_outputs",
            report_to = "none", # Use this for WandB etc
        ),
    )
```

---

8. Benchmarking & Evaluation

Required Components:

- **Metrics Used (e.g., Accuracy, F1, BLEU, etc.):** Accuracy, Precision, Recall, F1-score, Confusion Matrix
- **Why those metrics?**

  These metrics provide a comprehensive view of the model's performance:

  - Accuracy is easy to understand but can be misleading in imbalanced datasets.

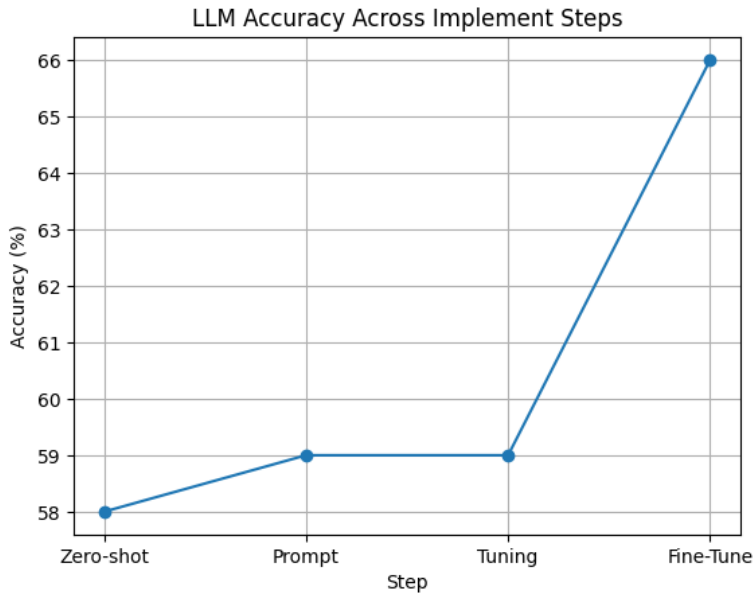
- Precision and Recall help understand the balance between correctly predicted positives and actual positives.
- F1-Score is useful when you want a single metric that balances Precision and Recall.
- Confusion Matrix offers a detailed view of model performance for each class.

- **Benchmark Dataset & Sample Size:**
  - **Dataset Name:** drugscom_reviews
  - **Sample Size:** 500 test samples

Code: Metric Calculation

```
import numpy as np
from sklearn.metrics import accuracy_score, precision_score, recall_score, f1_score, confusion_matrix, classification_report
import seaborn as sns

new_test_df = pd.read_csv("/content/drive/MyDrive/shin_colab/2_epochs_processed_ft.csv")

# Select the first 500 rows
new_test_df_copy = new_test_df.iloc[:500].copy()  # Use .copy() to avoid SettingWithCopyWarning
new_test_df_copy['Predict_sentiment'] = new_test_df_copy['Predict_sentiment'].apply(str.strip)


y_pred = list(new_test_df_copy['Predict_sentiment'])

y_true = list(new_test_df_copy['Sentiment'])


# Compute classification metrics
accuracy = accuracy_score(y_true, y_pred)
precision = precision_score(y_true, y_pred, average='macro')
recall = recall_score(y_true, y_pred, average='macro')
f1 = f1_score(y_true, y_pred, average='macro')

print("\n=== Model Performance Metrics ===")
print(f"Accuracy: {accuracy:.2f}")
print(f"Precision (Macro): {precision:.2f}")
print(f"Recall (Macro): {recall:.2f}")
print(f"F1-Score (Macro): {f1:.2f}")
print(classification_report(y_true, y_pred))

conf_matrix = confusion_matrix(y_true, y_pred)

plt.figure(figsize=(6, 5))
sns.heatmap(conf_matrix, annot=True, fmt="d", cmap="Blues",
            xticklabels=['Negative', 'Neutral', 'Positive'],
            yticklabels=['Negative', 'Neutral', 'Positive'])
plt.xlabel("Predicted Label")
plt.ylabel("True Label")
plt.title("Confusion Matrix")
plt.show()
```

Plot Results



Interpretation:

Explain what the graph tells us. Where is the biggest gain? Where does improvement taper off?

The graph illustrates the accuracy of a Large Language Model (LLM) across four implementation steps: **Zero-shot, Prompt, Tuning, and Fine-Tune**.

Key Observations:

1. **Initial Accuracy (Zero-shot)**: The model begins with an accuracy of **58%** without any task-specific instructions or optimization.
2. **Slight Improvement (Prompt)**: A slight increase to **59%** is observed when prompts are used, indicating a minimal impact from prompt engineering.
3. **Plateau (Tuning)**: The accuracy remains at **59%**, showing no significant improvement with basic tuning techniques.
4. **Significant Gain (Fine-Tune)**: The model's accuracy jumps to **66%** in the fine-tuning step, which is a substantial improvement.

Biggest Gain:

- The most significant gain occurs in the **Fine-Tune step**, with a **7% increase** in accuracy (from 59% to 66%). This demonstrates the high impact of fine-tuning in optimizing model performance.

Where Improvement Tapers Off:

- The improvement tapers off during the **Prompt and Tuning steps**, where the model's accuracy only slightly improves or stays stagnant.

---

9. UI Integration

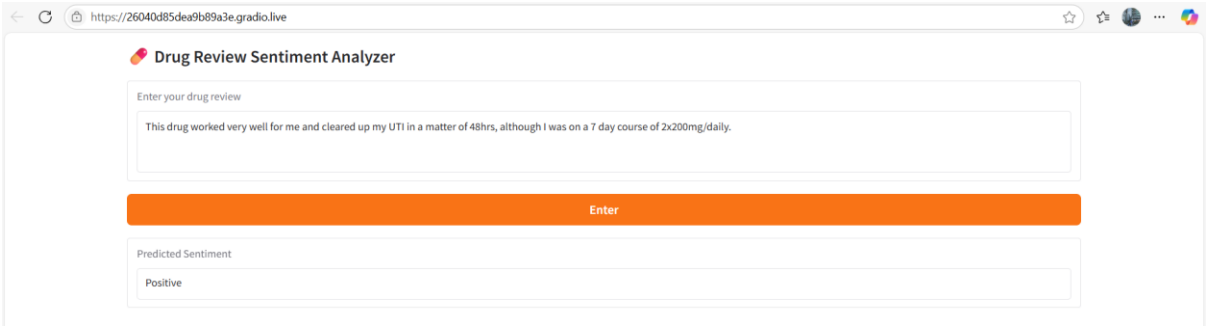
- **Tool Used (e.g., Gradio, Streamlit):** Gradio
- **Key Features of the Interface:**

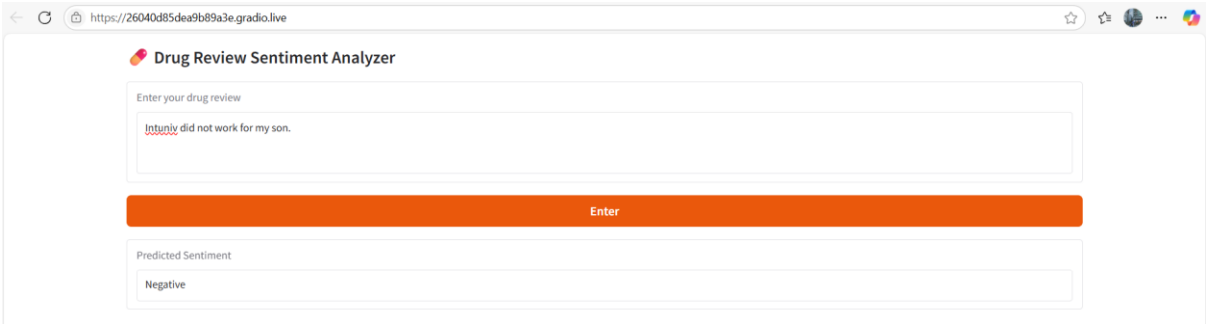
    1. **Title and Icon:** The interface displays a clear title (" 💊 Drug Review Sentiment Analyzer") with a relevant emoji icon, making the purpose of the app immediately clear.
    2. **Input Section:**
        - **Textbox for Review:** A multi-line text box labeled "Enter your drug review," allowing users to type their review.
        - **Placeholder Text:** The placeholder text guides users with "Type your review and press Enter..." for better user experience.
    3. **Interactive Button:**
        - **Enter Button:** A prominently displayed orange "Enter" button allows users to manually trigger the sentiment analysis.
    4. **Keyboard Shortcut:**
        - **Enter Key Binding:** Users can directly press "Enter" on their keyboard to submit the review, improving accessibility.
    5. **Output Display:**
        - **Predicted Sentiment Textbox:** A separate textbox labeled "Predicted Sentiment" dynamically displays the predicted sentiment (e.g., "Negative").

- **Include 2+ Screenshots of Working UI:**

Code: UI Implementation (Gradio Example)

```
# Function to send input to the API
def get_sentiment(input_text):
    payload = {"input_text": input_text}
    try:
        response = requests.post(api_url, json=payload)
        if response.status_code == 200:
            return response.json().get("result", "No result returned")
        else:
            return f"API Error: {response.status_code}"
    except Exception as e:
        return f"Request failed: {str(e)}"
```


https://26040d85dea9b89a3e.gradio.live
Drug Review Sentiment Analyzer
Enter your drug review
This drug worked very well for me and cleared up my UTI in a matter of 48hrs, although I was on a 7 day course of 2x200mg/daily.
Enter
Predicted Sentiment
Positive


https://26040d85dea9b89a3e.gradio.live
Drug Review Sentiment Analyzer
Enter your drug review
Intuniv did not work for my son.
Enter
Predicted Sentiment
Negative