LLM PROJECT REPORT

---

1. Student Information

- **Student Name:** Dac Cong Nguyen
- **Student ID (optional):** 224263452
- **Date Submitted:** 18/05/2025

---

2. Project Introduction

- **Title of the Project:** Fine-tuning LLMs for Enterprise Applications
- **Use case:** Sentiment Analysis in Healthcare
- **What is the project about?**

  This project focuses on adapting large language models (LLMs) to perform sentiment analysis specifically within the healthcare domain. It involves refining pre-trained LLMs using domain-specific data to improve their understanding of medical language, patient feedback, and healthcare-related sentiment. The goal is to enable more accurate analysis of sentiments in healthcare communications such as patient reviews, clinical notes, or survey responses.

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

  This project is important because sentiment analysis in healthcare can uncover valuable insights into patient experiences, identify areas for improvement, and support decision-making. Generic language models, such as pre-trained LLMs often lack the nuance required to interpret healthcare-specific language, so fine-tuning them ensures better performance and relevance in enterprise applications.

---

3. Environment Setup

- **Development Platform:**
  - Google Colab
  - Local Machine: Linux
  - GPU Available? Yes
  - GPU Type: NVIDIA Tesla T4
- **Python Version:** 3.11

Code: GPU Check

```
print('GPU Available:', torch.cuda.is_available())
```

```
GPU Available: True
```

---

4. LLM Setup

- **Model Name:** Qwen2.5-7B-Instruct
- **Provider:** HuggingFace
- **Key Libraries & Dependencies (with versions):**
    - bitsandbytes: 0.45.5
    - peft: 0.15.2
    - trl: 0.15.2
    - datasets: 3.6.0
    - huggingface-hub: 0.31.2
    - unsloth: @ git+https://github.com/unslothai/unsloth.git@64ea3b88cbb86940fe9cd860ef774ee3e0ec464f (use this version to fix issues with Colab)
    - unsloth-zoo: 2025.5.7
    - gradio: 5.29.1
- **Libraries and Dependencies Required:**

  (Include all relevant Python packages. Provide requirements.txt if available.)

  The requirement.txt file is available at this github commit.

Code: Install & Import

```
!pip -q install --no-deps bitsandbytes accelerate xformers==0.0.29.post3 peft
trl triton cut_cross_entropy unsloth_zoo
!pip -q install sentencepiece protobuf datasets huggingface_hub hf_transfer
!pip -q install --upgrade --no-cache-dir "unsloth[colab-new] @ git+https://
github.com/unslothai/unsloth.git"
!pip -q install gradio
```

---

5. Dataset Description

- **Dataset Name & Source:** DrugsCom Reviews. This dataset is originally sourced from the UCI Machine Learning Repository.
- **Access Link (if public):** https://huggingface.co/datasets/Zakia/drugscom_reviews/
- **Feature Dictionary / Variable 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 (1-10). 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:** Only two columns were used for sentiment analysis:

1. review: To put into the prompt for the LLM to predict the output.
2. rating: To be converted to sentiment labels with the rule:
   1-4: Negative, 5-6: Neutral, 7-10: Positive. This rule is based on a paper that used the same dataset in the project (Link).

Code: Load & Preprocess Dataset

Preprocessing steps, including converting ratings to sentiment labels, and wrapping the reviews with prompts

```
def convert_rating(rating):
    label = np.where(rating >= 7, "Positive", np.where(rating >= 5, "Neutral",
    "Negative"))
    return str(label)
```

```
def generating_labels_func(examples, rating_mapping=rating_mapping):
    labels = list(map(rating_mapping.get, examples['rating']))
    return {"sentiment": labels,}

def label_mapping_func(examples, label_mapping=label_mapping):
    outputs = list(map(label_mapping.get, examples['sentiment']))
    return {"y": outputs,}

def formatting_prompts_func(examples, few_shot=False, train=True):
    reviews = examples["review"]
    labels = examples["sentiment"]
    texts = []

    if few_shot:
        prompt_template = few_shot_prompt_template
    else:
        prompt_template = zero_shot_prompt_template

    for review, label in zip(reviews, labels):
        if not train:
            label = ""
        text = prompt_template.format(review, label) + EOS_TOKEN
        texts.append(text)
    return {"text": texts,}
```

Load and apply preprocessing steps on the dataset

```
test_dataset = load_dataset("Zakia/drugscom_reviews", split="test")
test_dataset = test_dataset.map(generating_labels_func, batched=True, fn_kwargs=
{'rating_mapping': rating_mapping})
test_dataset = test_dataset.map(label_mapping_func, batched=True, fn_kwargs=
{'label_mapping': label_mapping})
zero_shot_test_dataset = test_dataset.map(formatting_prompts_func,
batched=True, fn_kwargs={'few_shot': False, 'train': False})
few_shot_test_dataset = test_dataset.map(formatting_prompts_func, batched=True,
fn_kwargs={'few_shot': True, 'train': False})
```

```
README.md: 100% 6.72k/6.72k [00:00<00:00, 650kB/s]
train.tsv: 100% 84.3M/84.3M [00:00<00:00, 174MB/s]
test.tsv: 100% 28.1M/28.1M [00:00<00:00, 57.4MB/s]
Generating train split: 100% 161297/161297 [00:01<00:00, 105745.94 examples/s]
Generating test split: 100% 53766/53766 [00:00<00:00, 115819.84 examples/s]
Map: 100% 53766/53766 [00:00<00:00, 365306.14 examples/s]
Map: 100% 53766/53766 [00:00<00:00, 275407.08 examples/s]
Map: 100% 53766/53766 [00:00<00:00, 124847.96 examples/s]
Map: 100% 53766/53766 [00:01<00:00, 81698.92 examples/s]
```

---

6. 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 (F1-score)
1	Zero-shot Prompt	No examples	60%
2	Few-shot Prompt	3 examples added	61%
3	Temperature Tuning	Temperature = 0.7	61%
4	**Fine-tuning**	**2 epochs (3,000 samples) on Drugreviews**	**66%**
5	Further fine-tuning	1 epoch (1,500 samples more)	61%

From the evaluation results, I will use the first fine-tuned model (fine-tuned using 2 epochs of 3,000 training samples) as the final version.

Code Snippets for Each Step

Different prompt formats: Zero-shot and Few-shot (3 examples added)

```
zero_shot_prompt_template = """Below is an instruction that describes a task,
paired with an input that provides further context. Write a response that
appropriately completes the request.

### Instruction:
You are a helpful medical assistant. Your task is to predict the sentiment
based on the review of a drug.
The output will be one of the following sentiments: Positive, Neutral, Negative.
Make sure that the output only includes one sentiment in the above list, do not
add any other word or explanation.

### Review: {}

### Sentiment: {}
"""

few_shot_prompt_template = """Below is an instruction that describes a task,
paired with an input that provides further context. Write a response that
appropriately completes the request.

### Instruction:
You are a helpful medical assistant. Your task is to predict the sentiment
based on the review of a drug.
The output will be one of the following sentiments: Positive, Neutral, Negative.
Make sure that the output only includes one sentiment in the above list, do not
add any other word or explanation.

Examples:
- Review: "This medication completely cleared up my skin within a week. I
couldn't be happier!"
  Sentiment: Positive

- Review: "I started taking this drug last month. Still waiting to see any
major effects."
  Sentiment: Neutral

- Review: "The side effects were unbearable — constant nausea and dizziness. I
had to stop using it."
  Sentiment: Negative

### Review: {}

### Sentiment: {}
"""
```

Function to predict sentiment based on review text, with temperature as its argument

```
def predict(data, model, tokenizer, label_mapping, temperature=1.0):
    start_time = time.time()

    prompt = data['text']
    messages = [
        {"role": "system", "content": "As a helpfule health assistant, you will
        predict sentiment based on the drug reviews."},
        {"role": "user", "content": prompt}
    ]

    text = tokenizer.apply_chat_template(
        messages,
        tokenize=False,
        add_generation_prompt=True
    )

    model_inputs = tokenizer([text], return_tensors="pt").to(model.device)

    generated_ids = model.generate(
        **model_inputs,
        max_new_tokens=64,
        temperature=temperature
    )

    generated_ids = [
        output_ids[len(input_ids):] for input_ids, output_ids in zip
        (model_inputs.input_ids, generated_ids)
    ]

    response = tokenizer.batch_decode(generated_ids, skip_special_tokens=True)
    [0]

    if response in label_mapping.keys():
        y = label_mapping[response]
    else:
        y = -1
        print(f'Wrong response: {response}')

    elapsed_time = time.time() - start_time

    return y, elapsed_time
```

Fine-tuning function with LoRA and SFT

```python
def fine_tune(model, tokenizer, train_dataset, num_sample_per_class=1000, lora_rank=16, epochs=1):
    # Prepare data
    shuffled_training_dataset = train_dataset.shuffle()
    train_dataset_label_0 = shuffled_training_dataset.filter(lambda example: example["y"] == 0).select(range
    (num_sample_per_class))
    train_dataset_label_1 = shuffled_training_dataset.filter(lambda example: example["y"] == 1).select(range
    (num_sample_per_class))
    train_dataset_label_2 = shuffled_training_dataset.filter(lambda example: example["y"] == 2).select(range
    (num_sample_per_class))
    train_dataset_sample = concatenate_datasets([train_dataset_label_0, train_dataset_label_1, train_dataset_label_2])

    # Fine-tune
    FastLanguageModel.for_training(model)
    model = FastLanguageModel.get_peft_model(
        model,
        r=lora_rank,
        target_modules=["q_proj", "k_proj", "v_proj", "o_proj",
                        "gate_proj", "up_proj", "down_proj",],
        lora_alpha=16,
        lora_dropout=0,
        bias="none",
        use_gradient_checkpointing="unsloth",
        random_state=SEED,
        use_rslora=False,
        loftq_config=None,
    )

    trainer = SFTTrainer(
        model=model,
        tokenizer=tokenizer,
        train_dataset=train_dataset_sample,
        dataset_text_field="text",
        formatting_func=None,
        max_seq_length=max_seq_length,
        dataset_num_proc=2,
        packing=False,
        args = TrainingArguments(
            per_device_train_batch_size=2,
            gradient_accumulation_steps=4,
            warmup_steps=5,
            num_train_epochs=epochs,
            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=SEED,
            output_dir="outputs",
            report_to="none",
        ),
    )

    trainer_stats = trainer.train()
```

Example output of the fine-tuning process

```
fine_tune(model, tokenizer, train_dataset, num_sample_per_class=1000, lora_rank=16, epochs=2)

Filter: 100% ████████████████████ 161297/161297 [00:06<00:00, 26747.55 examples/s]
Filter: 100% ████████████████████ 161297/161297 [00:07<00:00, 17537.24 examples/s]
Filter: 100% ████████████████████ 161297/161297 [00:06<00:00, 26594.29 examples/s]
Unsloth 2025.5.5 patched 28 layers with 28 QKV layers, 28 O layers and 28 MLP layers.
Unsloth: Tokenizing ["text"] (num_proc=2): 100% ████████████ 3000/3000 [00:03<00:00, 1022.92 examples/s]
==((====))==  Unsloth - 2x faster free finetuning | Num GPUs used = 1
   \\   /|    Num examples = 3,000 | Num Epochs = 2 | Total steps = 750
O^O/ \_/ \    Batch size per device = 2 | Gradient accumulation steps = 4
\        /    Data Parallel GPUs = 1 | Total batch size (2 x 4 x 1) = 8
 "-____-"     Trainable parameters = 40,370,176/7,000,000,000 (0.58% trained)
Unsloth: Will smartly offload gradients to save VRAM!
[750/750 1:48:22, Epoch 2/2]
```

7. Benchmarking & Evaluation

Required Components:

- **Metrics Used:** Accuracy, Precision, Recall, F1-score.
- **Why those metrics?** These metrics are useful for classification tasks, which indicate how good the model predict each label in sentiment analysis. Especially, this dataset is heavy imbalanced with the neutral sentiment consists of just more than 11% of the data, F1-score is a key metric to evaluate the performance of the model.
- **Benchmark Dataset & Sample Size:** The first 500 samples in the test dataset were used for benchmarking.

Code:

Metric Calculation

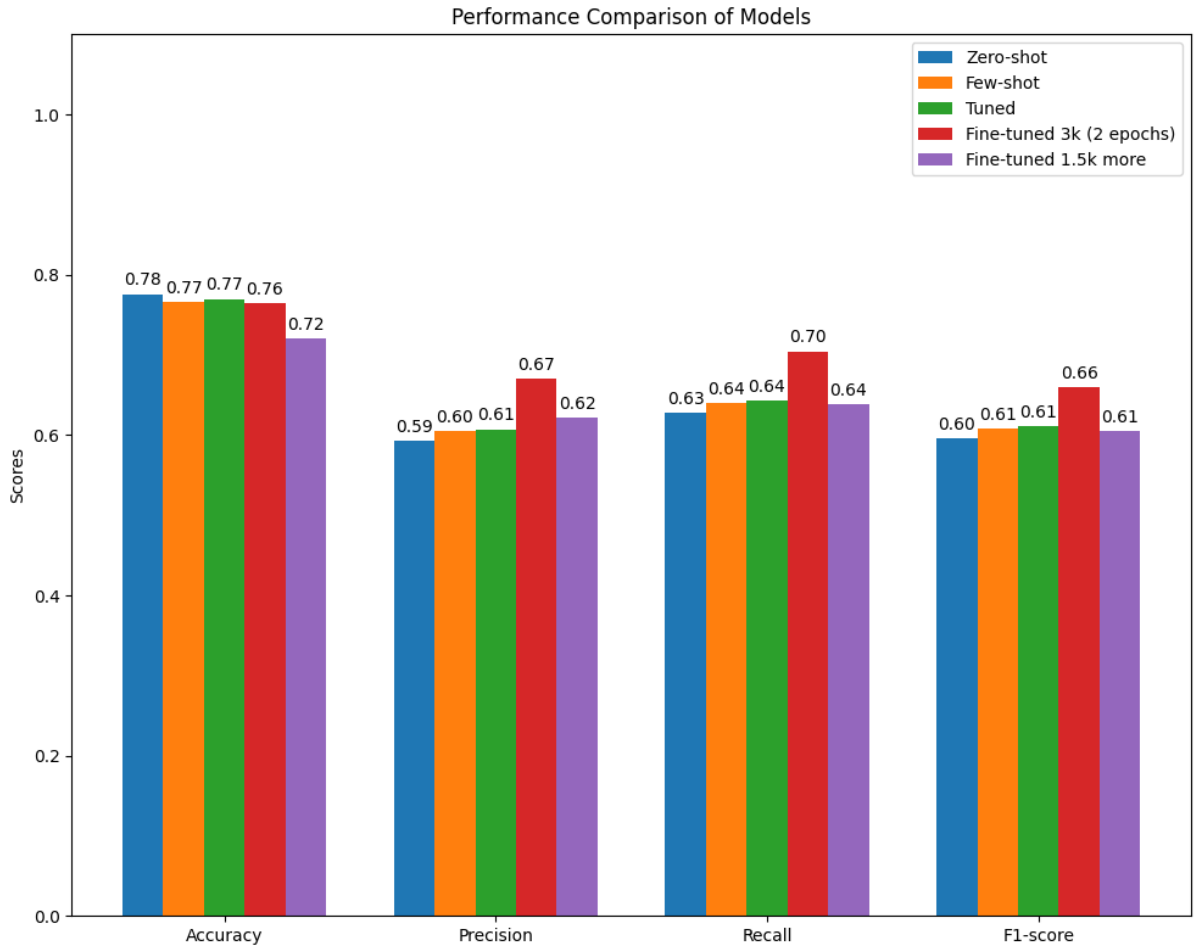
```
def get_metrics(true, pred):
    return [
        accuracy_score(true, pred),
        precision_score(true, pred, average='macro', zero_division=0),
        recall_score(true, pred, average='macro', zero_division=0),
        f1_score(true, pred, average='macro', zero_division=0)
    ]
```

```
zero_shot__metrics = get_metrics(zero_shot_base__true_labels, zero_shot_base__pred_labels)
few_shot__metrics = get_metrics(few_shot_base__true_labels, few_shot_base__pred_labels)
tuned__metrics = get_metrics(few_shot_temp_07_base__true_labels, few_shot_temp_07_base__pred_labels)
ft1__metrics = get_metrics(ft1__true_labels, ft1__pred_labels)
ft2__metrics = get_metrics(ft2__true_labels, ft2__pred_labels)
```

Plot function

```
# Data for plotting
metric_names = ['Accuracy', 'Precision', 'Recall', 'F1-score']
x = np.arange(len(metric_names))  # label locations
width = 0.15  # width of the bars

# Create plot
fig, ax = plt.subplots(figsize=(10, 8))
bars1 = ax.bar(x - width * 2, zero_shot__metrics, width, label='Zero-shot')
bars2 = ax.bar(x - width, few_shot__metrics, width, label='Few-shot')
bars3 = ax.bar(x, tuned__metrics, width, label='Tuned')
bars4 = ax.bar(x + width, ft1__metrics, width, label='Fine-tuned 3k (2 epochs)')
bars5 = ax.bar(x + width * 2, ft2__metrics, width, label='Fine-tuned 1.5k more')

# Add labels and legend
ax.set_ylabel('Scores')
ax.set_title('Performance Comparison of Models')
ax.set_xticks(x)
ax.set_xticklabels(metric_names)
ax.set_ylim(0, 1.1)
ax.legend()

# Display values on bars
for bars in [bars1, bars2, bars3, bars4, bars5]:
    for bar in bars:
        height = bar.get_height()
        ax.annotate(f'{height:.2f}',
                    xy=(bar.get_x() + bar.get_width() / 2, height),
                    xytext=(0, 3),  # vertical offset
                    textcoords="offset points",
                    ha='center', va='bottom')

plt.tight_layout()
plt.show()
```

Plot Results



Interpretation:

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

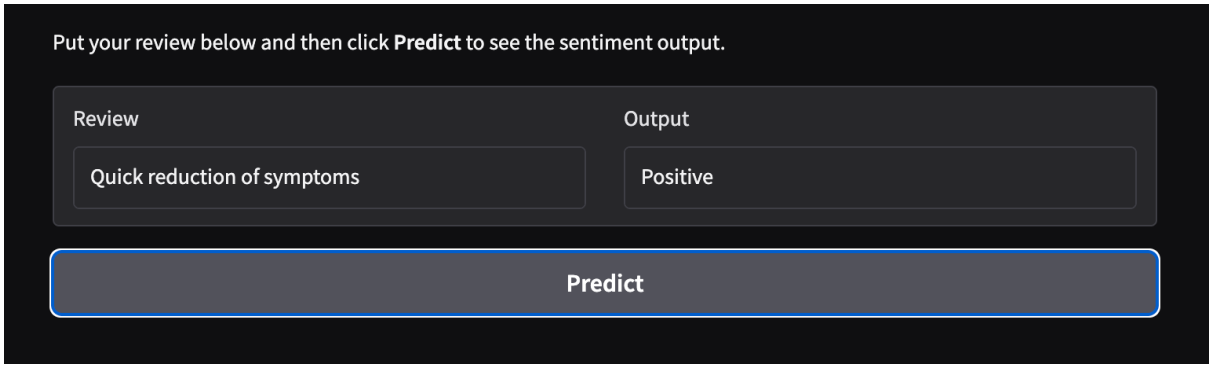
This graph shows the performance of different versions of the model and generally, zero-shot < few-shot < hyper-parameters tuned < fine-tuned model.

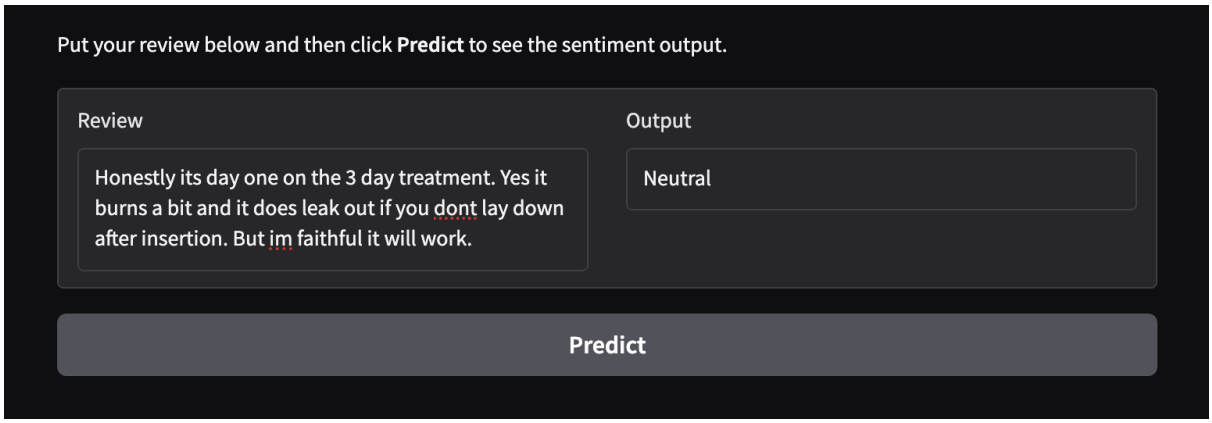
- Biggest gain: Precision (from 0.59 to 0.67) and F1-score (0.60 to 0.66) after tuning on 3k samples.
- Improvement tapers off: After the first fine-tuning with 3k samples in 2 epochs, further fine-tuning with an additional 1.5k samples does not improve performance, and in some metrics (e.g., Accuracy), it degrades quite significantly.
- Conclusion: Fine-tuning helps significantly up to a point (around 3k samples), after which overfitting may occur.

---

8. UI Integration

- **Tool Used:** Gradio
- **Key Features of the Interface:** It has a text box to input the review of the drug and a text box for the predicted sentiment.
- **Include 2+ Screenshots of Working UI**





Code: UI Implementation

```
[41] def single_predict(review):
        prompt_template = """Below is an instruction that describes a task, paired with an input that provides further context. Write a re

        ### Instruction:
        You are a helpful medical assistant. Your task is to predict the sentiment based on the review of a drug.
        The output will be one of the following sentiments: Positive, Neutral, Negative.
        Make sure that the output only includes one sentiment in the above list, do not add any other word or explanation.

        Examples:
        - Review: "This medication completely cleared up my skin within a week. I couldn't be happier!"
          Sentiment: Positive

        - Review: "I started taking this drug last month. Still waiting to see any major effects."
          Sentiment: Neutral

        - Review: "The side effects were unbearable — constant nausea and dizziness. I had to stop using it."
          Sentiment: Negative

        ### Review: {}

        ### Sentiment:
        """

        messages = [
            {"role": "system", "content": "As a health assistant, you will predict sentiment based on the drug reviews."},
            {"role": "user", "content": prompt_template.format(review)}
        ]

        text = tokenizer.apply_chat_template(
            messages,
            tokenize=False,
            add_generation_prompt=True,
            temperature=0.8
        )

        model_inputs = tokenizer([text], return_tensors="pt").to(model.device)

        generated_ids = model.generate(
            **model_inputs,
            max_new_tokens=64
        )

        generated_ids = [
            output_ids[len(input_ids):] for input_ids, output_ids in zip(model_inputs.input_ids, generated_ids)
        ]

        response = tokenizer.batch_decode(generated_ids, skip_special_tokens=True)[0]
        return response
```

```
[42] with gr.Blocks() as demo:
        gr.Markdown("Put your review below and then click **Predict** to see the sentiment output.")
        with gr.Row():
            inp = gr.Textbox(placeholder="Type the review", label="Review")
            out = gr.Textbox(label="Output")
        btn = gr.Button("Predict")
        btn.click(fn=single_predict, inputs=inp, outputs=out)

    demo.launch()
```