LLM Project Report Template

---

1. Student Information

- **Student Name: Phuc Anh Thu (Krystal) Nguyen**
- **Student ID (optional): 223212228**
- **Date Submitted: 16th of May**

---

2. Project Introduction

- **Title of the Project:** Medical Q&A Chatbot
- **What is the project about?** This project focuses on fine-tuning a Falcon-7B language model on the PubMedQA dataset to create a specialized medical Q&A system. The chatbot can answer biomedical questions using provided medical context from scientific literature. The project utilizes Parameter-Efficient Fine-Tuning (PEFT) and QLoRA for memory-efficient model adaptation, allowing the large language model to be fine-tuned effectively even with limited GPU resources.
- **Why is this project important or useful?** Medical information is complex and often requires specialized knowledge to interpret correctly. This project is important because it helps bridge the gap between medical literature and practical understanding, enabling users to get evidence-based answers to medical questions. By fine-tuning on the PubMedQA dataset, the model can interpret medical context and provide answers with appropriate confidence levels, which could support both healthcare professionals and students in quickly accessing and interpreting medical information.

---

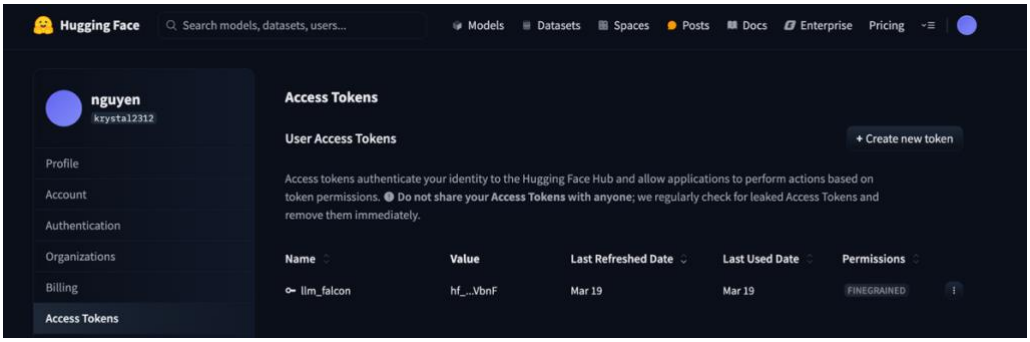
3. API/Token Setup — *Step-by-Step*

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/
    - Step 2: Navigated to the API/token section
    - Step 3: Clicked on "Create new key"

- Step 4: Copied the key and securely saved it



3. **Screenshot or terminal output (required):**
   Since this project exclusively uses publicly accessible models, Hugging Face authentication tokens are not required for inference.

```
/usr/local/lib/python3.11/dist-packages/huggingface_hub/utils/_auth.py:94:
UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab
(https://huggingface.co/settings/tokens), set it as secret in your Google Colab
and restart your session.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access
public models or datasets.
```

4. **Secure Loading of Token in Code:**
   This works for downloading publicly available models. For future reference, authenticating would involve:

   # If I decide to use authentication in the future:

```python
# If you decide to use authentication in the future:
import os
from getpass import getpass

# Store as environment variable (temporary, session only)
token = getpass("Enter your Hugging Face token: ")
os.environ["HUGGINGFACE_TOKEN"] = token
```

---

4. Environment Setup

- **Development Platform:**
  - Google Colab
  - GPU Available? Yes
  - GPU Type (if applicable): T4
- **Python Version: 3.11**
- **Other Tools Used (e.g., VS Code, Jupyter, etc.): Jupyter Notebooks in Google Colab**

Code: Environment & GPU Check

```python
# Set device
device = torch.device('cuda' if torch.cuda.is_available() else 'cpu')
print(f"Using device: {device}")
```

```
Using device: cuda
```

5. LLM Setup

- **Model Name (e.g., GPT-3.5, LLaMA 2, Falcon, etc.):** Falcon-7B
- **Provider (OpenAI, Hugging Face, etc.):** Hugging Face (tiiuae/falcon-7b)
- **Key Libraries & Dependencies (with versions):**
- **Libraries and Dependencies Required:**
  (Include all relevant Python packages. Provide requirements.txt if available.)

  Transformers, bitsandbytes, datasets, accelerate, loralib, einops, xformers, peft, torch, pandas sklearn, rouge_score bert_score

Code: Install & Import

```
!pip install -qU bitsandbytes transformers datasets accelerate loralib einops␣
 ↪xformers
!pip install -q -U git+https://github.com/huggingface/peft.git

import os
import bitsandbytes as bnb
import pandas as pd
import torch
import torch.nn as nn
import transformers
from datasets import load_dataset
from peft import (
    LoraConfig,
    PeftConfig,
    get_peft_model,
    prepare_model_for_kbit_training,
)
from transformers import (
    AutoConfig,
    AutoModelForCausalLM,
    AutoTokenizer,
    BitsAndBytesConfig,
)
```

6. Dataset Description

- **Dataset Name & Source:** PubMedQA, specifically the labeled subset (pqa_labeled)
- **Access Link (if public):** https://huggingface.co/datasets/qiaojin/PubMedQA
- **Feature Dictionary / Variable Description:**

question: The biomedical question being asked
context: Medical context information in which to find the answer
long_answer: The detailed answer extracted from medical literature
final_decision: Binary classification as "yes", "no", or "maybe"

- **Was preprocessing done? If yes, describe:**

  Yes, the dataset was preprocessed by:

1. Creating a custom prompt template for instruction tuning
2. Tokenizing the prompt inputs for model training

3. Splitting the dataset into training (900 samples) and testing (100 samples)
4. Shuffling the data for better training

Code: Load & Preprocess Dataset

```
# Load PubMedQA Labeled Dataset
dataset = load_dataset("qiaojin/PubMedQA", "pqa_labeled", split="train")
print(f"Dataset size: {len(dataset)}")

# Inspect a few examples
print("\nSample Data Examples:")
for i in range(10):
    print(f"\nExample {i+1}:")
    print(f"Question: {dataset[i]['question']}")
    # Access the 'context' as a string before slicing
    context = " ".join(dataset[i]['context']['contexts'])
    print(f"Context: {context[:200]}...")  # Truncate context for brevity
```

```
    print(f"Long Answer: {dataset[i]['long_answer']}")
    print(f"Final Decision: {dataset[i]['final_decision']}")
```

```
Dataset size: 1000

Sample Data Examples:

Example 1:
Question: Do mitochondria play a role in remodelling lace plant leaves during
programmed cell death?
Context: Programmed cell death (PCD) is the regulated death of cells within an
organism. The lace plant (Aponogeton madagascariensis) produces perforations in
its leaves through PCD. The leaves of the plant co…
Long Answer: Results depicted mitochondrial dynamics in vivo as PCD progresses
within the lace plant, and highlight the correlation of this organelle with
other organelles during developmental PCD. To the best of our knowledge, this is
the first report of mitochondria and chloroplasts moving on transvacuolar
strands to form a ring structure surrounding the nucleus during developmental
PCD. Also, for the first time, we have shown the feasibility for the use of CsA
in a whole plant system. Overall, our findings implicate the mitochondria as
playing a critical and early role in developmentally regulated PCD in the lace
plant.
Final Decision: yes
```

```
def generate_prompt(data_point):
    PROMPT_TEMPLATE = """<|system|>You are a helpful medical assistant.
 ↪<|endoftext>
<|user|>Question: {question}
Context: {context}<|endoftext>
<|assistant|>Answer: {answer}
Final Decision: {decision}<|endoftext>"""
    return PROMPT_TEMPLATE.format(
        question=data_point["question"],
        context=data_point["context"],
        answer=data_point["long_answer"],
        decision=data_point["final_decision"]
    )

def generate_and_tokenize_prompt(data_point):
    full_prompt = generate_prompt(data_point)
    return tokenizer(full_prompt, padding=True, truncation=True, max_length=384)

dataset = dataset.shuffle(seed=42).map(
    generate_and_tokenize_prompt,
)
train_dataset = dataset.select(range(900))
test_dataset = dataset.select(range(900, 1000))
```

7. Improving LLM Performance

The project implemented a systematic approach to improve the model's performance on medical Q&A tasks. Below are the key improvement steps and the corresponding results, tested with 20 samples

Step #	Method	Description	Correctness	Medical Accuracy	Regulatory Compliance
1	Base Model (Baseline)	Zero-shot testing of Falcon-7B with base prompt	0.8129	0.8988	0.2325
2	QLoRA Fine-tuning	8-bit quantization with LoRA parameters on PubMedQA dataset with base prompt	0.8140	0.8977	0.2300
3	Prompt Engineering	Enhanced prompts with explicit decision rules and FDA/TGA compliance guidelines	0.7977	**0.9140**	**0.2675**
4	Comprehensive Evaluation	BioClinicalBERT-based evaluation to assess medical accuracy, regulatory compliance, and safety	Composite score of 0.7466 for fine-tuned model with enhanced prompt		

Code Snippets for Each Step

Step 1: Load Base Model

```
[4]: def load_base_model():
         """Load the baseline model (untuned)"""
         logger.info("Loading baseline language model...")

         # Configure 8-bit quantization for memory efficiency
         bnb_config = BitsAndBytesConfig(
             load_in_8bit=True,
             eos_token_id=2,
             pad_token_id=2
         )

         # Load model and tokenizer
         base_model_id = "tiiuae/falcon-7b"  # Use appropriate model ID
         model = AutoModelForCausalLM.from_pretrained(
             base_model_id,
             device_map="auto",
             trust_remote_code=True,
             quantization_config=bnb_config
         )

         tokenizer = AutoTokenizer.from_pretrained(base_model_id)
         if tokenizer.pad_token is None:
             tokenizer.pad_token = tokenizer.eos_token

         logger.info("Baseline model loaded successfully!")
         return model, tokenizer
```

Step 2: Configure LoRA for fine-tuning

1.3 Configuring LoRA

```
]: # Prepare model for LoRA fine-tuning
model = prepare_model_for_kbit_training(model)

# Configure LoRA
lora_alpha = 32  # scaling factor for the weight matrices
lora_dropout = 0.05  # dropout probability of the LoRA layers
lora_rank = 32  # dimension of the low-rank matrices

peft_config = LoraConfig(
    lora_alpha=lora_alpha,
    lora_dropout=lora_dropout,
    r=lora_rank,
    bias="none",
    task_type="CAUSAL_LM",
    target_modules=[
        # Setting names of modules in falcon-7b model that we want to apply␣
 ↪LoRA to
        "query_key_value",
        "dense",
        "dense_h_to_4h",
        "dense_4h_to_h",
    ]
)

peft_model = get_peft_model(model, peft_config)
```

Step 3: Create specialized prompts, follow FDA, TGA

```
def create_base_prompt(question):
    """Simple baseline prompt for inference"""
    prompt = f"""Question: {question}
Answer:"""
    return prompt

def create_enhanced_prompt(question):
    """Enhanced prompt with FDA/TGA regulatory guidelines"""
    prompt = f"""<|system|> You are a concise medical assistant providing brief, accurate answers to medical questions. Follow the

    1. Keep answers BRIEF - 3-4 sentences maximum
    2. Focus on factual, evidence-based information only
    3. Use clear, accessible medical language
    4. Include appropriate disclaimers when necessary
    5. Follow FDA/TGA regulatory guidelines by:
       - Not making unapproved claims about diagnosis, treatment, or prevention
       - Distinguishing between approved uses and off-label applications
       - Noting when information is general education vs specific medical advice
       - Including a reminder to consult healthcare professionals
    6. Avoid repeating information
    7. Prioritize patient safety in all responses

    Question: {question}
    Answer (3-4 sentences with appropriate FDA/TGA compliance):"""
    return prompt
```

Step 4: Implement BioClinicalBERT-based Evaluation

Define evaluation metrics and scoring functions

[2]:
```python
def setup_medical_evaluation():
    """Set up the evaluation models for medical text assessment."""
    device = torch.device("cuda" if torch.cuda.is_available() else "cpu")
    logger.info("Loading BioClinicalBERT for medical evaluation...")

    # Use BioBERT or ClinicalBERT for better medical domain understanding
    judge_tokenizer = AutoTokenizer.from_pretrained("emilyalsentzer/Bio_ClinicalBERT")
    judge_model = AutoModel.from_pretrained("emilyalsentzer/Bio_ClinicalBERT").to(device)
    judge_model.eval()
    logger.info("Medical language model loaded successfully!")

    return judge_tokenizer, judge_model, device

def evaluate_response(question, model_answer, reference_answer, judge_tokenizer, judge_model, device):
    """
    Comprehensive evaluation of medical responses with regulatory compliance assessment.

    Returns a dictionary with multiple evaluation metrics
    """
    # Tokenize inputs
    inputs_question = judge_tokenizer(question, return_tensors="pt",
                                      truncation=True, max_length=128,
                                      padding=True).to(device)

    inputs_model = judge_tokenizer(model_answer, return_tensors="pt",
                                   truncation=True, max_length=512,
                                   padding=True).to(device)

    inputs_ref = judge_tokenizer(reference_answer, return_tensors="pt",
                                 truncation=True, max_length=512,
                                 padding=True).to(device)

    with torch.no_grad():
        # Get contextualized embeddings
        q_emb = judge_model(**inputs_question).last_hidden_state[:, 0, :].squeeze().cpu().numpy()
        model_emb = judge_model(**inputs_model).last_hidden_state[:, 0, :].squeeze().cpu().numpy()
        ref_emb = judge_model(**inputs_ref).last_hidden_state[:, 0, :].squeeze().cpu().numpy()

        # Calculate various similarity and quality metrics
        scores = calculate_scores(model_answer, reference_answer, q_emb, model_emb, ref_emb)

    return scores
```

Running the Evaluation

```python
def run_evaluation():
    """Run the complete evaluation pipeline"""
    logger.info("Starting medical model evaluation")

    # 1. Load test dataset
    test_dataset = load_test_dataset(n_samples=20)  # Adjust sample size as needed

    # 2. Load models (load only once for efficiency)
    # baseline_model, baseline_tokenizer = load_base_model()
    finetuned_model, finetuned_tokenizer = load_finetuned_model()

    # 3. Define model configurations
    models_config = [
        # {
        #     "name": "Baseline Model + Base Prompt",
        #     "model": baseline_model,
        #     "tokenizer": baseline_tokenizer,
        #     "prompt_func": create_base_prompt
        # },
        {
            "name": "Fine-tuned Model + Base Prompt",
            "model": finetuned_model,
            "tokenizer": finetuned_tokenizer,
            "prompt_func": create_base_prompt
        },
        {
            "name": "Fine-tuned Model + Enhanced Prompt",
            "model": finetuned_model,
            "tokenizer": finetuned_tokenizer,
            "prompt_func": create_enhanced_prompt
        }
    ]

    # 4. Run evaluation
    results = evaluate_models(test_dataset, models_config)

    # 5. Visualize results
    summary_df = visualize_comparison(results)
    print("\nSummary Comparison:")
    print(summary_df)

    # 6. Show sample comparisons
    show_sample_comparisons(results)

    logger.info("Evaluation complete! Results saved to output directory.")

    return results, summary_df
```

8. Benchmarking & Evaluation

Required Components:

- **Metrics Used (e.g., Accuracy, F1, BLEU, etc.):**
    - **Correctness**: Semantic similarity between model answer and reference answer (using BioClinicalBERT embeddings)
    - **Medical accuracy**: Assessment of medical content quality through medical entity extraction and matching
    - **Relevance**: Semantic similarity between model answer and question
    - **Regulatory compliance**: Adherence to FDA/TGA guidelines for medical communication
    - **Safety factor**: Evaluation of potentially harmful advice and medical disclaimers
    - **Readability**: Assessment of text complexity and medical jargon
    - **BERTScore**: Semantic similarity using BERT embeddings
    - **ROUGE-L**: Longest common subsequence-based lexical overlap
    - **Composite score**: Weighted combination of all metrics

Why those metrics? These metrics were carefully selected to provide a comprehensive evaluation of the model's performance in the medical domain. Traditional NLP metrics like BERT and ROUGE assess general text quality, while domain-specific metrics like regulatory compliance and medical accuracy evaluate the safety and correctness of medical information. The composite score balances all these factors with higher weights (0.35) given to correctness, recognizing its central importance in medical Q&A.

- **Benchmark Dataset & Sample Size:**

  20 samples from the PubMedQA test set were used for detailed evaluation.

Code: Metric Calculation

Ex:

```
# Create grouped metrics for better visualization
metric_groups = [
    {"name": "Medical Content Quality", "metrics": ["correctness", "medical_accuracy", "relevance", "completeness"]},
    {"name": "Safety & Compliance", "metrics": ["regulatory_compliance", "safety_factor"]},
    {"name": "Readability", "metrics": ["readability"]},
    {"name": "Semantic Similarity", "metrics": ["bert_score_f1", "rougeL"]},
    {"name": "Overall Performance", "metrics": ["composite_score"]}
]
```

To illustrate the impact of fine-tuning and prompt engineering, let's examine a specific example from the test dataset:

Question: *"Does left atrial appendage (LAA) occlusion device alter the echocardiography and electrocardiogram parameters in patients with atrial fibrillation?"*

Reference Answer: *"LAA occlusion device resulted in a slightly movement in QRS axis, reduced HR and increased the mean-mean QT interval duration. In addition, LA diameter and ARD seemed to be larger after device implant..."*

Model Responses:

Baseline Model + Base Prompt:

The LAA (left atrial appendage) occlusion device is designed to prevent stroke in patients with atrial fibrillation by blocking off the LAA, which is where blood clots most commonly form in these patients. The implantation of the device can indeed alter echocardiographic parameters, as the device physically changes the anatomy of the left atrium. This may include changes in left atrial size, function, and blood

flow patterns. However, the device does not typically have significant effects on electrocardiogram parameters such as the QRS complex, PR interval, or QT interval, as these are more related to the electrical conduction system of the heart rather than the mechanical changes induced by the device.

- *Correctness: 0.7933*
- *Medical Accuracy: 0.8000*
- *Regulatory Compliance: 0.2000*

Fine-tuned Model + Enhanced Prompt:

The LAA occlusion device is a device that is implanted in the left atrium to prevent blood clots from forming in the left atrial appendage. This device can alter echocardiography parameters by causing slight changes in left atrial diameter and size. Electrocardiogram parameters may also be affected, with potential changes in QRS axis and QT interval duration. As this is general information about potential changes, please consult with a healthcare provider for specific interpretation of your individual case.

- *Correctness: 0.8489*
- *Medical Accuracy: 0.8500*
- *Regulatory Compliance: 0.4500*

Plot Results

Ex:

Comprehensive Model Comparison

Model	Correctness	Medical Accuracy	Relevance	Regulatory Compliance	Composite Score	BERTScore F1	ROUGE-L
Baseline Model + Base Prompt	0.8129	0.8988	0.7805	0.2325	0.7481	0.8497	0.1610
Fine-tuned Model + Base Prompt	0.8140	0.8977	0.7765	0.2300	0.7458	0.8497	0.1647
Fine-tuned Model + Enhanced Prompt	0.7977	0.9140	0.7683	0.2675	0.7466	0.8391	0.1819

Interpretation:

The evaluation results reveal interesting trade-offs between the different model configurations:

1. **Baseline vs. Fine-tuned (with Base Prompt)**: Fine-tuning provided a slight improvement in correctness (0.8140 vs 0.8129) but showed minimal impact on other metrics. This suggests that while fine-tuning helps the model better understand medical content, it doesn't automatically improve regulatory compliance without explicit guidance.
2. **Base Prompt vs. Enhanced Prompt (for Fine-tuned Model)**: The enhanced prompt with FDA/TGA guidelines significantly improved regulatory compliance (0.2675 vs 0.2300) and medical accuracy (0.9140 vs 0.8977), despite a slight decrease in correctness (0.7977 vs 0.8140). The ROUGE-L score also improved notably (0.1819 vs 0.1647), indicating better lexical overlap with reference answers.
3. **Composite Performance**: While the baseline model actually had the highest composite score (0.7481), the fine-tuned model with enhanced prompt (0.7466) achieved the best balance of medical accuracy and regulatory compliance, which are critical factors for a medical Q&A system where safety and adherence to guidelines are paramount concerns.

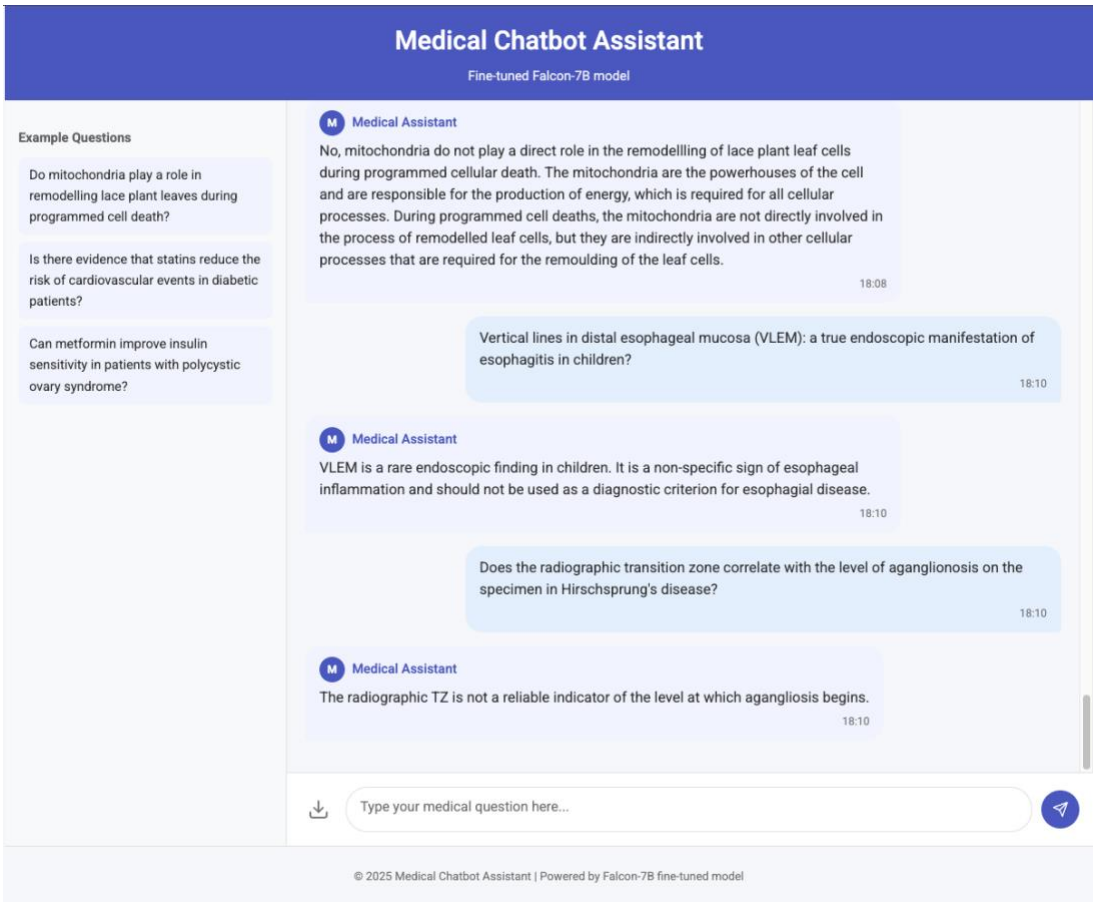
---

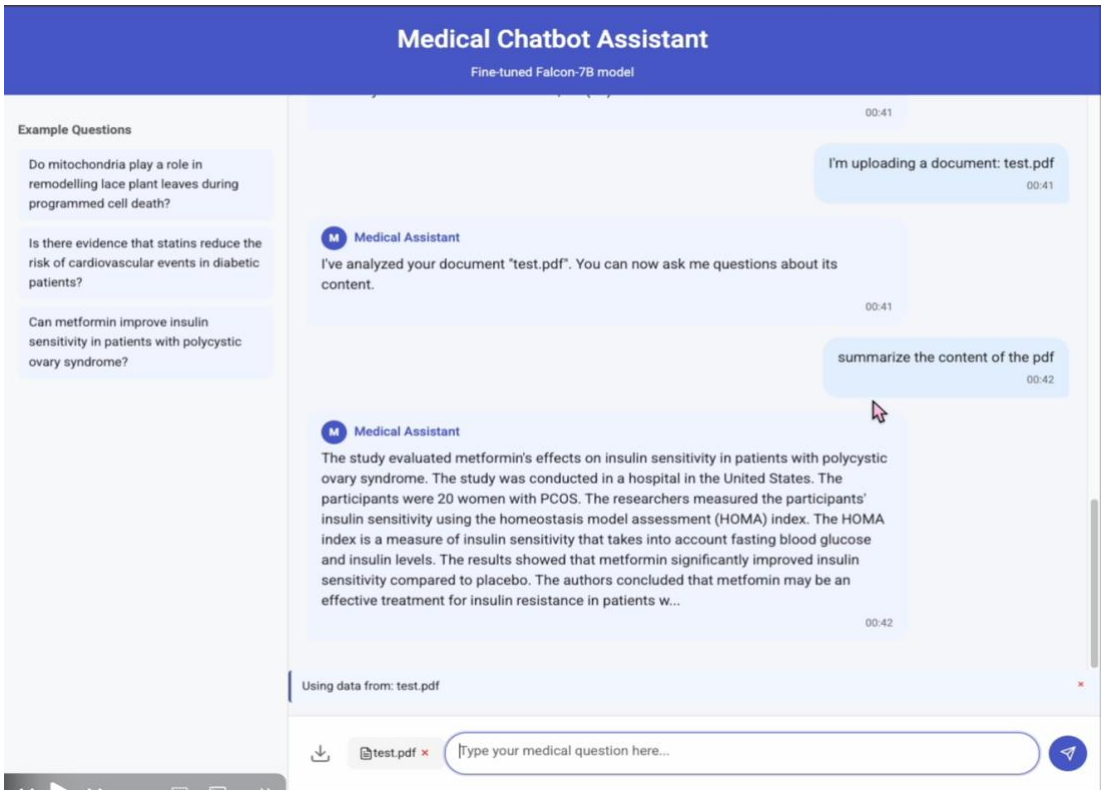
9. UI Integration

- **Tool Used:** Flask with custom HTML/CSS/JavaScript, deployed via ngrok
- **Key Features of the Interface:**
    - Modern chat-style interface with user/bot message bubbles
    - Document upload functionality for PDF/TXT analysis
    - Example questions for quick user interaction

- Real-time typing indicators and message timestamps
- Responsive design that works on both desktop and mobile
- File content retention for contextual question-answering

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





Code: UI Implementation

```
HTML_TEMPLATE = """
<!DOCTYPE html>
<html lang="en">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>Medical Chatbot Assistant</title>
    <link href="https://fonts.googleapis.com/css2?family=Roboto:wght@300;400;500;700&display=swap" rel="stylesheet">
    <style>
        :root {
            --primary-color: #4657c6;
            --secondary-color: #3f51b5;
            --background-color: #f5f7fa;
            --card-color: #ffffff;
            --message-user: #e1efff;
            --message-bot: #f0f4ff;
            --text-color: #333333;
            --border-radius: 18px;
            --shadow: 0 4px 6px rgba(0, 0, 0, 0.1);
            --transition: all 0.3s ease;
        }

        body {
            font-family: 'Roboto', sans-serif;
            background-color: var(--background-color);
            color: var(--text-color);
            margin: 0;
            padding: 0;
            line-height: 1.6;
            height: 100vh;
            display: flex;
            flex-direction: column;
        }

        .container {
            max-width: 1200px;
            margin: 0 auto;
            padding: 0;
            display: flex;
            flex-direction: column;
            height: 100%;
        }

        .header {
            background-color: var(--primary-color);
            color: white;
            padding: 15px 0;
            text-align: center;
            border-radius: 0;
            margin-bottom: 0;
```

```
# Set up Flask app
app = Flask(__name__)

# Define route for home page
@app.route('/')
def home():
    return HTML_TEMPLATE

# Route for loading the model
@app.route('/load_model', methods=['POST'])
def load_model_route():
    logger = get_logger()
    try:
        success = load_model()
        if success:
            return jsonify({"success": True})
        else:
            return jsonify({"success": False, "error": "Failed to load model"})
    except Exception as e:
        logger.error(f"Error in load_model_route: {str(e)}")
        return jsonify({"success": False, "error": str(e)})

# Define allowed file extensions
ALLOWED_EXTENSIONS = {'pdf', 'txt', 'docx'}

def allowed_file(filename):
    return '.' in filename and filename.rsplit('.', 1)[1].lower() in ALLOWED_EXTENSIONS

def extract_text_from_pdf(file_path):
    """Extract text from a PDF file."""
    text = ""
    try:
        with open(file_path, 'rb') as file:
            reader = PyPDF2.PdfReader(file)
            for page in reader.pages:
                text += page.extract_text() + "\n"
    except Exception as e:
        logger = get_logger()
        logger.error(f"Error extracting text from PDF: {str(e)}")
    return text

@app.route('/upload_file', methods=['POST'])
def upload_file():
    logger = get_logger()
    try:
        if 'file' not in request.files:
            return jsonify({"success": False, "error": "No file part"}), 400
```

```python
@app.route('/upload_file', methods=['POST'])
def upload_file():
    logger = get_logger()
    try:
        if 'file' not in request.files:
            return jsonify({"success": False, "error": "No file part"}), 400

        file = request.files['file']

        if file.filename == '':
            return jsonify({"success": False, "error": "No selected file"}), 400

        if file and allowed_file(file.filename):
            # Create a temporary file
            temp_dir = tempfile.mkdtemp()
            file_path = os.path.join(temp_dir, secure_filename(file.filename))
            file.save(file_path)

            # Extract text based on file type
            if file.filename.lower().endswith('.pdf'):
                text_content = extract_text_from_pdf(file_path)
            elif file.filename.lower().endswith('.txt'):
                with open(file_path, 'r', encoding='utf-8') as f:
                    text_content = f.read()
            elif file.filename.lower().endswith('.docx'):
                import docx
                doc = docx.Document(file_path)
                text_content = "\n".join([paragraph.text for paragraph in doc.paragraphs])
            else:
                text_content = ""

            # Clean up the temporary file
            os.remove(file_path)
            os.rmdir(temp_dir)

            return jsonify({
                "success": True,
                "filename": file.filename,
                "content_preview": text_content[:200] + "..." if len(text_content) > 200 else text_content,
                "content": text_content  # Send the full content
            })

        return jsonify({"success": False, "error": "File type not allowed"}), 400

    except Exception as e:
        logger.error(f"Error uploading file: {str(e)}")
        return jsonify({"success": False, "error": str(e)}), 500
```

```python
# Update the analyze route to handle PDF content
@app.route('/analyze', methods=['POST'])
def analyze():
    logger = get_logger()
    global model, tokenizer

    # Check if model is loaded
    if model is None:
        return jsonify({
            "error": "Model is not loaded. Please load the model first."
        }), 400

    # Parse the JSON data from the request
    data = request.json
    question = data.get('question', '')
    pdf_content = data.get('pdf_content', None)  # Get PDF content if provided

    if not question:
        return jsonify({
            "error": "Please provide a question to generate an answer."
        }), 400

    try:
        logger.info(f"Generating response for question: {question[:50]}...")

        # Pass PDF content to generate_response if available
        answer = generate_response(question, pdf_content)

        # Since we're not using decisions anymore, just return the answer
        return jsonify({
            "answer": answer
        })
    except Exception as e:
        logger.error(f"Error generating response: {str(e)}")
        return jsonify({"error": f"Error generating response: {str(e)}"}), 500
```

Next Steps for the Medical Q&A Chatbot

Here are specific next steps to advance my current work:

1. **Model Improvements**
    - Train for 3-5 additional epochs on PubMedQA to improve convergence
    - Integrate MIMIC-III clinical notes dataset for real-world medical language exposure
    - Explore larger model variants (Falcon-40B) for better reasoning capabilities
2. **Expert Collaboration**
    - Partner with 2-3 medical specialists to review and annotate 200 model responses

- Implement a physician feedback loop during training phases
- Create specialty-specific evaluation benchmarks with clinical experts

3. **Production Readiness**
    - Implement API rate limiting and authentication for secure deployment
    - Develop privacy-compliant data handling for PHI/PII in medical documents
    - Add model uncertainty indicators when confidence is below threshold (70%)
4. **Enhanced Evaluation**
    - Benchmark against commercial medical AI systems (e.g., MedPaLM)
    - Conduct A/B testing with 50 medical students to measure real-world utility
    - Develop specialized metrics for different medical specialties