Generative AI-based Medical Chatbot Application

Project Solution

Project Overview

In this project, you will design and implement a Generative AI-based medical chatbot application using a Large Language Model (LLM). The chatbot will enable doctors or healthcare professionals to interact with patients' medical records and receive intelligent, context-aware responses.

We'll use Gradio to build an interactive web interface and a pre-trained lightweight LLM to power the chatbot.

Setup

```
# Install required packages
!pip install -q transformers accelerate gradio torch
```

```
# Import essential Python libraries
import warnings
warnings.filterwarnings('ignore')

# Import Gradio for building interactive interfaces
import gradio as gr

# Import transformer libraries for the language model
from transformers import AutoTokenizer, AutoModelForCausalLM, pipeline
import torch
```

```
# Print library versions for reproducibility
import transformers
print(f"Transformers version: {transformers.__version__}")
print(f"Torch version: {torch.__version__}")
print(f"Gradio version: {gr.__version__}")
```

Task 1: Develop the Gradio Web Application Interface

Build a chatbot-style layout with text boxes for medical report input, question input, and answer output.

```
# Define a placeholder function for testing the interface
def test_interface(medical_report, question):
    """
    Test function to verify the interface layout works correctly.
    This will be replaced with the actual LLM-powered function later.
    """
    if not medical_report or not question:
        return "Please provide both a medical report and a question."

    return f"Interface is working! You asked: '{question}' about the report."

# Create the Gradio interface
demo_interface = gr.Interface(
```

```
        fn=test_interface,

        # Define input widgets
        inputs=[
            gr.Textbox(lines=10, label="Medical Report", placeholder="Paste the medical report here..."),
            gr.Textbox(lines=2, label="Question", placeholder="Type your question here...")
        ],

        # Define output widget
        outputs=gr.Textbox(lines=5, label="Answer"),

        # Add title and description
        title="Medical Chatbot - Interface Test",
        description="Enter a medical report and ask a question to test the interface.",

        # Set button label
        submit_btn=gr.Button("Get the Answer")
    )

    # Launch the interface to verify the layout
    print("\n✓ Task 1 Complete: Gradio interface created successfully")
    print("Launching interface for verification...\n")
    demo_interface.launch(share=False)
```

Task 1 Screenshot: The interface shows three text boxes (Medical Report, Question, Answer) and a "Get the Answer" button.

Task 2: Integrate a Lightweight LLM

Download and load a small pre-trained model from Hugging Face (google/flan-t5-small).

```
    # Check if GPU is available (makes inference faster)
    device = "cuda" if torch.cuda.is_available() else "cpu"
    print(f"Using device: {device}")
```

```
    # Define the model name from Hugging Face
    model_name = "TinyLlama/TinyLlama-1.1B-Chat-v1.0"

    print("Loading model... This may take 1-2 minutes.")
    print("Model: TinyLlama 1.1B - optimized for efficient inference")
```

```
    # Load the tokenizer (converts text to numbers the model understands)
    tokenizer = AutoTokenizer.from_pretrained(model_name)

    # Set padding token to avoid warnings
    tokenizer.pad_token = tokenizer.eos_token

    print("✓ Tokenizer loaded successfully")
```

```

```

```
    # Load the language model
    model = AutoModelForCausalLM.from_pretrained(
        model_name,
        torch_dtype=torch.float16 if device == "cuda" else torch.float32,  # Use half precision on GPU
        device_map="auto",  # Automatically place model on available device
        low_cpu_mem_usage=True  # Optimize for low memory usage
    )

    print("✓ Model loaded successfully")
```

Task 2 Screenshot: Console output shows successful tokenizer and model loading confirmation.

Task 3: Implement the Answering Function

Create a Python function that accepts the medical report and question as inputs and uses the LLM to generate a concise, relevant answer.

```
def generate_answer(report, question):
    """
    Generates an answer to a question based on a medical report using the LLM.

    Args:
        report (str): The medical report text
        question (str): The question about the report

    Returns:
        str: Generated answer from the LLM
    """
    # Handle cases where either input is missing
    if not report or not report.strip():
        return "Please provide a medical report."

    if not question or not question.strip():
        return "Please provide a question."

    # Create the input prompt combining the report and question
    prompt = f"""Based on the following medical report, answer the question.

Medical Report:
{report}

Question: {question}

Answer:"""

        # Tokenize the input text
    inputs = tokenizer(prompt, return_tensors="pt").to(device)

    # Generate response using the model
    with torch.no_grad():  # Disable gradient calculation for inference
        outputs = model.generate(
            **inputs,
            max_new_tokens=200,  # Maximum tokens to generate
        )

    # Decode the generated tokens back to text
    answer = tokenizer.decode(outputs[0], skip_special_tokens=True)

    return answer
```

```
# Test the answering function with sample inputs
sample_report = """
Age: 45 years
Diagnosis: Type 2 Diabetes Mellitus
Medications: Metformin 500mg twice daily
Blood Sugar: Fasting 140 mg/dL
Blood Pressure: 130/85 mmHg
Recommendations: Diet modification, regular exercise, monitor blood sugar levels."""

sample_question = "What is the patient's diagnosis?"

print("Testing the answering function...\n")
print(f"Question: {sample_question}\n")

test_answer = generate_answer(sample_report, sample_question)
print(f"Answer: {test_answer}")
```

Task 3 Screenshot: Console output shows the function definition and a working test output with a generated answer.

Task 4: Connect the Interface and Backend

Link the "Get the Answer" button to the generate_answer function and ensure the app correctly displays the output.

```
# Create the final Gradio interface with the LLM-powered function
medical_chatbot = gr.Interface(
    fn=generate_answer,

    # Define input widgets
    inputs=[
        gr.Textbox(lines=10, label="Medical Report", placeholder="Paste the medical report here..."),
        gr.Textbox(lines=2, label="Question", placeholder="Type your question about the report...")
    ],
```