Copy_of_ChatbotQA (1)

May 16, 2025

0.1 1. Environment Setup

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

[ ]:
```
!pip install --upgrade peft accelerate transformers -U
```

[ ]:
```
import logging
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,
    PeftModel
)
from transformers import (
    AutoConfig,
    AutoModelForCausalLM,
    AutoTokenizer,
    BitsAndBytesConfig,
)
from flask import Flask, request, jsonify
from safetensors.torch import load_file
```

[ ]:
```
from google.colab import drive
drive.mount('/content/drive')
```

```
Drive already mounted at /content/drive; to attempt to forcibly remount, call
drive.mount("/content/drive", force_remount=True).
```

[ ]:
```
logging.basicConfig(level=logging.INFO, format='%(asctime)s - %(name)s -␣
↪%(levelname)s - %(message)s')

# Global variables
model = None
tokenizer = None
```

[ ]:
```
def get_logger():
    """Get the logger for the current module."""
    return logging.getLogger(__name__)
```

0.2 2. Model Loading

[ ]:
```
def load_model():
    """Load the fine-tuned model."""
    global model, tokenizer
    logger = get_logger()

    try:
        # Update this path to your Colab storage location
        model_path = "/content/drive/MyDrive/falcon-7b-pubmedqa"
        checkpoint_path = os.path.join(model_path, "checkpoint-225")

        logger.info(f"Loading model from checkpoint: {checkpoint_path}")

        # Check if the checkpoint directory exists
        if not os.path.exists(checkpoint_path):
            logger.error(f"Checkpoint directory does not exist:␣
↪{checkpoint_path}")
            return False

        # Load adapter config to get base model name
        adapter_config_path = os.path.join(checkpoint_path, "adapter_config.
↪json")
        if not os.path.exists(adapter_config_path):
            logger.error(f"adapter_config.json not found in {checkpoint_path}")
            return False

        import json
        with open(adapter_config_path, 'r') as f:
            adapter_config = json.load(f)

        base_model_name = adapter_config.get("base_model_name_or_path")
        if not base_model_name:
            logger.error("base_model_name_or_path not found in adapter_config.
↪json")
            base_model_name = "tiiuae/falcon-7b"
```

```
        logger.info(f"Using default base model: {base_model_name}")
    else:
        logger.info(f"Using base model from config: {base_model_name}")

    # Configure quantization
    quantization_config = BitsAndBytesConfig(
        load_in_8bit=True,
        bnb_8bit_compute_dtype=torch.float16,
        bnb_8bit_use_double_quant=True,
        bnb_8bit_quant_type="nf4"
    )

    # Load base model
    logger.info(f"Loading base model: {base_model_name}")
    base_model = AutoModelForCausalLM.from_pretrained(
        base_model_name,
        quantization_config=quantization_config,
        trust_remote_code=True,
        low_cpu_mem_usage=True
    )

    # Load tokenizer
    logger.info("Loading tokenizer")
    tokenizer = AutoTokenizer.from_pretrained(checkpoint_path)
    tokenizer.pad_token = tokenizer.eos_token

    # Prepare model for LoRA fine-tuning
    logger.info("Preparing model for 8-bit training")
    base_model = prepare_model_for_kbit_training(base_model)

    # Create LoraConfig from adapter_config.json
    logger.info("Creating LoRA config")
    lora_config = LoraConfig(
        lora_alpha=adapter_config.get("lora_alpha", 32),
        lora_dropout=adapter_config.get("lora_dropout", 0.05),
        r=adapter_config.get("r", 32),
        bias=adapter_config.get("bias", "none"),
        task_type=adapter_config.get("task_type", "CAUSAL_LM"),
        target_modules=adapter_config.get("target_modules", [
            "query_key_value",
            "dense",
            "dense_h_to_4h",
            "dense_4h_to_h",
        ])
    )

    # Get PEFT model
```

```
        logger.info("Creating PEFT model")
        model = get_peft_model(base_model, lora_config)

        # Check for the adapter weights file
        adapter_model_path = os.path.join(checkpoint_path, "adapter_model.
↪safetensors")

        if os.path.exists(adapter_model_path):
            logger.info(f"Found adapter weights at: {adapter_model_path}")

            # Load adapter weights using safetensors
            adapter_state_dict = load_file(adapter_model_path)

            # Load adapter weights into the model
            logger.info("Loading adapter weights")
            model.load_state_dict(adapter_state_dict, strict=False)

            # Move model to GPU if available
            if torch.cuda.is_available():
                logger.info("Moving model to GPU")
                model = model.to("cuda")

            model.eval()
            logger.info("Model loaded successfully")
            return True
        else:
            logger.error(f"adapter_model.safetensors not found in␣
↪{checkpoint_path}")
            return False

    except Exception as e:
        logger.error(f"Error loading model: {str(e)}")
        import traceback
        logger.error(traceback.format_exc())
        return False
```

[ ]:
```
def extract_decision(answer):
    """Extract the assessment from the answer."""
    decision = "maybe"  # default
    answer_text = answer

    # Check for the new assessment format
    assessment_markers = [
        "Assessment: Strong evidence supports this.",
        "Assessment: Evidence does not support this.",
        "Assessment: Evidence is currently limited."
    ]
```

```
    for marker in assessment_markers:
        if marker in answer:
            answer_parts = answer.split(marker)
            answer_text = answer_parts[0].strip()

            if "Strong evidence supports" in marker:
                decision = "yes"
            elif "Evidence does not support" in marker:
                decision = "no"
            else:
                decision = "maybe"

            break

    # If we don't find the new format, check for old formats as a fallback
    if answer_text == answer:  # No change made above
        for marker in ["Final Decision:", "final decision:", "Decision:",␣
↪"Assessment:"]:
            if marker in answer:
                decision_part = answer.split(marker)[-1].strip().lower()
                answer_text = answer.split(marker)[0].strip()
                if "yes" in decision_part[:5] or "support" in decision_part and␣
↪not "not support" in decision_part:
                    decision = "yes"
                elif "no" in decision_part[:5] or "not support" in␣
↪decision_part or "does not support" in decision_part:
                    decision = "no"
                elif "maybe" in decision_part[:10] or "limit" in decision_part␣
↪or "insufficient" in decision_part or "conflict" in decision_part:
                    decision = "maybe"
                break

    return answer_text, decision
```

[ ]:
```
!pip install Flask
from flask import Flask, request, jsonify
```

```
Requirement already satisfied: Flask in /usr/local/lib/python3.11/dist-packages
(3.1.0)
Requirement already satisfied: Werkzeug>=3.1 in /usr/local/lib/python3.11/dist-
packages (from Flask) (3.1.3)
Requirement already satisfied: Jinja2>=3.1.2 in /usr/local/lib/python3.11/dist-
packages (from Flask) (3.1.6)
Requirement already satisfied: itsdangerous>=2.2 in
/usr/local/lib/python3.11/dist-packages (from Flask) (2.2.0)
Requirement already satisfied: click>=8.1.3 in /usr/local/lib/python3.11/dist-
```

```
packages (from Flask) (8.1.8)
Requirement already satisfied: blinker>=1.9 in /usr/local/lib/python3.11/dist-
packages (from Flask) (1.9.0)
Requirement already satisfied: MarkupSafe>=2.0 in
/usr/local/lib/python3.11/dist-packages (from Jinja2>=3.1.2->Flask) (3.0.2)
```

0.3 UI Chatbot

0.3.1 Backend | Python/Flask/FastAPI | API for UI model communication

0.3.2 Frontend | React | User-facing chatbot interface

0.3.3 Deploy via ngrok

[ ]:
```python
def generate_response(question, pdf_content=None, max_new_tokens=300):
    """Generate a response using the fine-tuned model with the specified prompt.
    """
    global model, tokenizer
    logger = get_logger()

    # Create a focused, concise medical Q&A prompt
    prompt = f"""<|system|>
You are a concise medical assistant providing brief, accurate answers to␣
 ↪medical questions. Follow these guidelines:

1. Keep answers BRIEF - 3-4 sentences maximum
2. Focus on the most relevant facts only
3. Use clear, accessible medical language
4. Avoid unnecessary technical jargon unless required
5. Never repeat information

Question: {question}
Answer (3-4 sentences):"""

    logger.info(f"Generating response with prompt: {prompt[:100]}...")
    inputs = tokenizer(prompt, return_tensors="pt").to(model.device)

    with torch.no_grad():
        outputs = model.generate(
            input_ids=inputs["input_ids"],
            attention_mask=inputs["attention_mask"],
            max_new_tokens=max_new_tokens,
            do_sample=True,
            temperature=0.2,
            top_p=0.85,
            repetition_penalty=2.0,  # Significantly increased to prevent␣
 ↪repetition
            no_repeat_ngram_size=3,
```

```
            max_length=inputs["input_ids"].shape[1] + 100,  # Stricter limit on␣
 ↪response length
            pad_token_id=tokenizer.eos_token_id
        )

    response = tokenizer.decode(
        outputs[0][inputs["input_ids"].shape[1]:], skip_special_tokens=True).
 ↪strip()
    logger.info(f"Generated response: {response[:100]}...")
    return response


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;
            flex-shrink: 0;
        }

        .header h1 {
            margin: 0;
            font-size: 1.8rem;
            font-weight: 500;
        }

        .header p {
            margin: 5px 0 0;
            opacity: 0.9;
            font-size: 0.9rem;
        }

        .chat-container {
            display: flex;
            flex-direction: column;
            height: 100%;
            overflow: hidden;
        }

        .chat-sidebar {
            width: 280px;
            background-color: #f8f9fa;
            border-right: 1px solid #e0e0e0;
            padding: 15px;
            flex-shrink: 0;
        }

        .chat-main {
```

```
            flex: 1;
            display: flex;
            flex-direction: column;
            overflow: hidden;
        }

        .chat-messages {
            flex: 1;
            overflow-y: auto;
            padding: 20px;
            display: flex;
            flex-direction: column;
        }

        .message {
            max-width: 75%;
            padding: 12px 16px;
            margin-bottom: 15px;
            border-radius: var(--border-radius);
            position: relative;
            line-height: 1.5;
        }

        .message-user {
            background-color: var(--message-user);
            align-self: flex-end;
            border-bottom-right-radius: 5px;
        }

        .message-bot {
            background-color: var(--message-bot);
            align-self: flex-start;
            border-bottom-left-radius: 5px;
        }

        .message-content {
            word-wrap: break-word;
        }

        .bot-info {
            display: flex;
            align-items: center;
            margin-bottom: 5px;
        }

        .bot-avatar {
            width: 28px;
```

```
            height: 28px;
            border-radius: 50%;
            background-color: var(--primary-color);
            display: flex;
            align-items: center;
            justify-content: center;
            margin-right: 8px;
            color: white;
            font-weight: bold;
            font-size: 12px;
        }

        .bot-name {
            font-weight: 500;
            font-size: 0.9rem;
            color: var(--primary-color);
        }

        .message-time {
            font-size: 0.75rem;
            color: #757575;
            margin-top: 5px;
            text-align: right;
        }

        .chat-input {
            display: flex;
            padding: 15px;
            background-color: white;
            border-top: 1px solid #e0e0e0;
            align-items: center;
            flex-shrink: 0;
        }

        .chat-input textarea {
            flex: 1;
            padding: 12px 15px;
            border: 1px solid #e0e0e0;
            border-radius: 24px;
            resize: none;
            height: 24px;
            max-height: 120px;
            transition: height 0.2s ease;
            font-family: inherit;
            font-size: 0.95rem;
            overflow-y: auto;
        }
```

```
.chat-input textarea:focus {
    outline: none;
    border-color: var(--primary-color);
    box-shadow: 0 0 0 2px rgba(63, 81, 181, 0.2);
}

.send-button {
    background-color: var(--primary-color);
    color: white;
    border: none;
    width: 42px;
    height: 42px;
    border-radius: 50%;
    margin-left: 10px;
    cursor: pointer;
    display: flex;
    align-items: center;
    justify-content: center;
    transition: var(--transition);
}

.send-button:hover {
    background-color: #3a4aa3;
}

.send-button:disabled {
    background-color: #bdbdbd;
    cursor: not-allowed;
}

.typing-indicator {
    display: none;
    align-items: center;
    background-color: var(--message-bot);
    padding: 12px 16px;
    border-radius: var(--border-radius);
    border-bottom-left-radius: 5px;
    max-width: 75%;
    align-self: flex-start;
    margin-bottom: 15px;
}

.typing-indicator span {
    height: 8px;
    width: 8px;
    float: left;
```

```
            margin: 0 1px;
            background-color: #9E9EA1;
            display: block;
            border-radius: 50%;
            opacity: 0.4;
        }

        .typing-indicator span:nth-of-type(1) {
            animation: typing 1s infinite;
        }

        .typing-indicator span:nth-of-type(2) {
            animation: typing 1s 0.25s infinite;
        }

        .typing-indicator span:nth-of-type(3) {
            animation: typing 1s 0.5s infinite;
        }

        @keyframes typing {
            0% {
                opacity: 0.4;
                transform: translateY(0);
            }
            50% {
                opacity: 1;
                transform: translateY(-5px);
            }
            100% {
                opacity: 0.4;
                transform: translateY(0);
            }
        }

        .file-upload {
            margin-right: 10px;
            position: relative;
        }

        .file-upload input {
            position: absolute;
            top: 0;
            right: 0;
            bottom: 0;
            left: 0;
            width: 100%;
            height: 100%;
```

```
            opacity: 0;
            cursor: pointer;
        }

        .file-upload button {
            background-color: transparent;
            border: none;
            color: #757575;
            padding: 8px;
            border-radius: 50%;
            cursor: pointer;
            transition: var(--transition);
            display: flex;
            align-items: center;
            justify-content: center;
        }

        .file-upload button:hover {
            background-color: #f5f5f5;
        }

        .upload-preview {
            display: none;
            margin: 10px 0;
            padding: 8px 12px;
            background-color: #f5f5f5;
            border-radius: 8px;
            font-size: 0.85rem;
        }

        .upload-preview span {
            display: flex;
            align-items: center;
            justify-content: space-between;
        }

        .upload-preview button {
            background: none;
            border: none;
            color: #f44336;
            cursor: pointer;
            font-size: 1rem;
            padding: 0 5px;
        }

        .footer {
            text-align: center;
```

```
        padding: 10px 0;
        color: #757575;
        font-size: 0.8rem;
        border-top: 1px solid #e0e0e0;
        flex-shrink: 0;
    }

    /* Example questions section */
    .example-questions {
        margin: 15px 0;
    }

    .example-questions h4 {
        margin: 0 0 10px;
        font-size: 0.9rem;
        color: #616161;
    }

    .example-question {
        padding: 8px 12px;
        background-color: #f0f4ff;
        border-radius: 8px;
        margin-bottom: 8px;
        cursor: pointer;
        transition: var(--transition);
        font-size: 0.9rem;
    }

    .example-question:hover {
        background-color: #e3eaff;
        transform: translateY(-2px);
    }

    /* File data indicator */
    .file-data-indicator {
        background-color: #f0f7ff;
        border-left: 3px solid var(--primary-color);
        padding: 8px 12px;
        margin-bottom: 15px;
        font-size: 0.85rem;
        border-radius: 4px;
        display: none;
    }

    /* Responsive layout */
    @media screen and (min-width: 768px) {
        .chat-container {
```

```
                flex-direction: row;
            }

            .chat-sidebar {
                display: block;
            }
        }

        @media screen and (max-width: 767px) {
            .chat-sidebar {
                display: none;
            }

            .message {
                max-width: 85%;
            }
        }

        #loadModelBtn {
            display: none; /* Hide the load model button in the UI */
        }
    </style>
</head>
<body>
    <div class="container">
        <div class="header">
            <h1>Medical Chatbot Assistant</h1>
            <p>Fine-tuned Falcon-7B model</p>
        </div>

        <div class="chat-container">
            <div class="chat-sidebar">
                <div class="example-questions">
                    <h4>Example Questions</h4>
                    <div class="example-question" onclick="setExample(this)">Do␣
 ↪mitochondria play a role in remodelling lace plant leaves during programmed␣
 ↪cell death?</div>
                    <div class="example-question" onclick="setExample(this)">Is␣
 ↪there evidence that statins reduce the risk of cardiovascular events in␣
 ↪diabetic patients?</div>
                    <div class="example-question"␣
 ↪onclick="setExample(this)">Can metformin improve insulin sensitivity in␣
 ↪patients with polycystic ovary syndrome?</div>
                </div>

                <button id="loadModelBtn" onclick="loadModel()">Load Model</
 ↪button>
```

```
            </div>

            <div class="chat-main">
                <div id="chatMessages" class="chat-messages">
                    <div class="message message-bot">
                        <div class="bot-info">
                            <div class="bot-avatar">M</div>
                            <div class="bot-name">Medical Assistant</div>
                        </div>
                        <div class="message-content">
                            Hello! I'm your medical assistant. I can answer␣
↪medical questions based on my training. You can also upload PDF documents␣
↪for me to analyze. How can I help you today?
                        </div>
                        <div class="message-time">Just now</div>
                    </div>
                </div>

                <div id="fileDataIndicator" class="file-data-indicator">
                    Using data from: <span id="filenameDisplay"></span>
                    <button onclick="clearFileData()" style="float:right;␣
↪background:none; border:none; color:#f44336; cursor:pointer;">×</button>
                </div>

                <div id="typingIndicator" class="typing-indicator"␣
↪style="display: none;">
                    <span></span>
                    <span></span>
                    <span></span>
                </div>

                <div class="chat-input">
                    <div class="file-upload">
                        <button>
                            <svg xmlns="http://www.w3.org/2000/svg" width="24"␣
↪height="24" viewBox="0 0 24 24" fill="none" stroke="currentColor"␣
↪stroke-width="2" stroke-linecap="round" stroke-linejoin="round">
                                <path d="M21 15v4a2 2 0 0 1-2 2H5a2 2 0 0␣
↪1-2-2v-4"></path>
                                <polyline points="7 10 12 15 17 10"></polyline>
                                <line x1="12" y1="15" x2="12" y2="3"></line>
                            </svg>
                        </button>
                        <input type="file" id="fileInput" accept=".pdf,.txt,.
↪docx" onchange="handleFileUpload(this)">
                    </div>
```

```
                    <div id="uploadPreview" class="upload-preview"></div>
                    <textarea id="userInput" placeholder="Type your medical␣
↪question here..." rows="1" oninput="autoResize(this)"></textarea>
                    <button id="sendButton" class="send-button"␣
↪onclick="sendMessage()">
                        <svg xmlns="http://www.w3.org/2000/svg" width="18"␣
↪height="18" viewBox="0 0 24 24" fill="none" stroke="currentColor"␣
↪stroke-width="2" stroke-linecap="round" stroke-linejoin="round">
                            <line x1="22" y1="2" x2="11" y2="13"></line>
                            <polygon points="22 2 15 22 11 13 2 9 22 2"></
↪polygon>
                        </svg>
                    </button>
                </div>
            </div>
        </div>

        <div class="footer">
            <p>© 2025 Medical Chatbot Assistant | Powered by Falcon-7B␣
↪fine-tuned model</p>
        </div>
    </div>

    <script>
        // Global variables
        let modelLoaded = true; // Assume model is loaded for UI purposes
        let uploadedFile = null;
        let fileContent = null;
        let messageHistory = [];

        // Auto-resize the textarea as user types
        function autoResize(textarea) {
            textarea.style.height = 'auto';
            textarea.style.height = (textarea.scrollHeight) + 'px';
        }

        // Load the model (hidden in UI, called automatically)
        function loadModel() {
            fetch('/load_model', {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json',
                }
            })
            .then(response => response.json())
            .then(data => {
                if (data.success) {
```

```
                    modelLoaded = true;
                    console.log("Model loaded successfully");
                } else {
                    console.error("Failed to load model:", data.error);
                    alert("Failed to load the medical model. Please try␣
↪refreshing the page.");
                }
            })
            .catch(error => {
                console.error('Error:', error);
            });
        }

        // Add example question to input
        function setExample(element) {
            document.getElementById('userInput').value = element.textContent;
            autoResize(document.getElementById('userInput'));
        }

        // Handle file upload
        function handleFileUpload(input) {
            const file = input.files[0];
            if (!file) return;

            const preview = document.getElementById('uploadPreview');
            preview.style.display = 'block';
            preview.innerHTML = `
                <span>
                    <svg xmlns="http://www.w3.org/2000/svg" width="16"␣
↪height="16" viewBox="0 0 24 24" fill="none" stroke="currentColor"␣
↪stroke-width="2" stroke-linecap="round" stroke-linejoin="round">
                        <path d="M14 2H6a2 2 0 0 0-2 2v16a2 2 0 0 0 2 2h12a2 2␣
↪0 0 0 2-2V8z"></path>
                        <polyline points="14 2 14 8 20 8"></polyline>
                        <line x1="16" y1="13" x2="8" y2="13"></line>
                        <line x1="16" y1="17" x2="8" y2="17"></line>
                        <polyline points="10 9 9 9 8 9"></polyline>
                    </svg>
                    ${file.name}
                    <button onclick="removeFile()">×</button>
                </span>
            `;

            uploadedFile = file;

            // Create FormData to send the file
            const formData = new FormData();
```

```
            formData.append('file', file);

            // Show a message about processing the file
            addMessage(`I'm uploading a document: ${file.name}`, 'user');

            // Show typing indicator
            document.getElementById('typingIndicator').style.display = 'flex';

            // Upload the file to the server
            fetch('/upload_file', {
                method: 'POST',
                body: formData
            })
            .then(response => response.json())
            .then(data => {
                // Hide typing indicator
                document.getElementById('typingIndicator').style.display =␣
↪'none';

                if (data.success) {
                    // Store the file content
                    fileContent = data.content_preview;

                    // Show file data indicator
                    document.getElementById('fileDataIndicator').style.display␣
↪= 'block';
                    document.getElementById('filenameDisplay').textContent =␣
↪file.name;

                    // Add a response from the bot
                    addMessage(`I've analyzed your document "${file.name}". You␣
↪can now ask me questions about its content.`, 'bot');
                } else {
                    addMessage(`I couldn't process your document: ${data.
↪error}`, 'bot');
                    removeFile();
                }
            })
            .catch(error => {
                // Hide typing indicator
                document.getElementById('typingIndicator').style.display =␣
↪'none';

                console.error('Error uploading file:', error);
                addMessage("I'm sorry, I encountered an error processing your␣
↪document. Please try again.", 'bot');
```

```
                removeFile();
            });
        }

        // Remove uploaded file
        function removeFile() {
            uploadedFile = null;
            fileContent = null;
            document.getElementById('uploadPreview').style.display = 'none';
            document.getElementById('fileInput').value = '';
            document.getElementById('fileDataIndicator').style.display = 'none';
        }

        // Clear file data but keep file uploaded
        function clearFileData() {
            fileContent = null;
            document.getElementById('fileDataIndicator').style.display = 'none';
            addMessage("I'm no longer using the uploaded document for context.
", 'bot');
        }

        // Send a message
        function sendMessage() {
            const input = document.getElementById('userInput');
            const message = input.value.trim();

            if (!message) return;

            // Add user message to chat
            addMessage(message, 'user');

            // Clear input and reset height
            input.value = '';
            input.style.height = '24px';

            // Show typing indicator
            document.getElementById('typingIndicator').style.display = 'flex';

            // Disable send button while processing
            document.getElementById('sendButton').disabled = true;

            // Send to backend
            fetch('/analyze', {
                method: 'POST',
                headers: {
                    'Content-Type': 'application/json',
                },
```

```
                body: JSON.stringify({
                    question: message,
                    pdf_content: fileContent
                }),
            })
            .then(response => response.json())
            .then(data => {
                // Hide typing indicator
                document.getElementById('typingIndicator').style.display =␣
↪'none';

                // Add bot response
                if (data.error) {
                    addMessage(`Error: ${data.error}`, 'bot');
                } else {
                    addMessage(data.answer, 'bot');
                }

                // Re-enable send button
                document.getElementById('sendButton').disabled = false;
            })
            .catch(error => {
                // Hide typing indicator
                document.getElementById('typingIndicator').style.display =␣
↪'none';

                // Show error
                console.error('Error:', error);
                addMessage("I'm sorry, I encountered an error processing your␣
↪request. Please try again.", 'bot');

                // Re-enable send button
                document.getElementById('sendButton').disabled = false;
            });
        }

        // Add a message to the chat
        function addMessage(text, sender) {
            const chatMessages = document.getElementById('chatMessages');
            const messageDiv = document.createElement('div');
            messageDiv.className = `message message-${sender}`;

            let messageContent = '';

            if (sender === 'bot') {
                messageContent += `
                    <div class="bot-info">
```

```
                        <div class="bot-avatar">M</div>
                        <div class="bot-name">Medical Assistant</div>
                    </div>
                `;
            }

            messageContent += `<div class="message-content">${text}</div>`;

            const now = new Date();
            const timeString = now.getHours().toString().padStart(2, '0') + ':' +
                               now.getMinutes().toString().padStart(2, '0');

            messageContent += `<div class="message-time">${timeString}</div>`;

            messageDiv.innerHTML = messageContent;
            chatMessages.appendChild(messageDiv);

            // Scroll to bottom
            chatMessages.scrollTop = chatMessages.scrollHeight;

            // Add to message history
            messageHistory.push({
                text: text,
                sender: sender,
                timestamp: new Date()
            });
        }

        // Handle Enter key in textarea
        document.getElementById('userInput').addEventListener('keydown', function(e) {
            if (e.key === 'Enter' && !e.shiftKey) {
                e.preventDefault();
                sendMessage();
            }
        });

        // Initialize - load model as soon as page loads
        window.onload = function() {
            loadModel();
        };
    </script>
</body>
</html>
"""
```

[ ]:
```
from flask import request, jsonify, render_template
from werkzeug.utils import secure_filename
import os
```

[ ]:
```
# For PDF processing
!pip install PyPDF2 pdfplumber python-docx
import PyPDF2
from werkzeug.utils import secure_filename
import tempfile
import os
```

0.3.4 Funtions for PDF handling

[ ]:
```
# 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

        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

# 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
```

[ ]:
```
!pip install pyngrok
```

```
Requirement already satisfied: pyngrok in /usr/local/lib/python3.11/dist-
packages (7.2.8)
Requirement already satisfied: PyYAML>=5.1 in /usr/local/lib/python3.11/dist-
packages (from pyngrok) (6.0.2)
```

[ ]:
```
from pyngrok import ngrok
```

0.3.5 Deploy using ngrok

[ ]:
```
def launch_app_with_ngrok():
    # Get a list of active tunnels
    tunnels = ngrok.get_tunnels()

    # Disconnect existing tunnels if any
    if tunnels:
        ngrok.disconnect(tunnels[0].public_url)

    # Set up ngrok
    http_tunnel = ngrok.connect(addr=port)
    public_url = http_tunnel.public_url
    print(f"Your Medical QA app is running at: {public_url}")
    print("\nInstructions:")
    print("1. Click the link above to open the interface")
    print("2. Enter your medical question and click 'Generate Answer'")

    # Model will be loaded on-demand via the web interface
    print("Model will be loaded when requested from the web interface.")

    # Start Flask app
    app.run(port=port, debug=False, use_reloader=False)
```

[ ]:
```
# Main execution
if __name__ == "__main__":
  port = 1234
    # launch_app_with_colab()  # Option 1: Use Colab's proxy
  launch_app_with_ngrok()
```

```
Your Medical QA app is running at: https://9000-34-139-142-30.ngrok-free.app

Instructions:
1. Click the link above to open the interface
2. Enter your medical question and click 'Generate Answer'
Model will be loaded when requested from the web interface.
 * Serving Flask app '__main__'
 * Debug mode: off
```

```
INFO:werkzeug:WARNING: This is a development server. Do not use it in a
production deployment. Use a production WSGI server instead.
 * Running on http://127.0.0.1:1234
INFO:werkzeug:Press CTRL+C to quit
INFO:werkzeug:127.0.0.1 - - [13/May/2025 14:33:43] "GET / HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [13/May/2025 14:33:44] "GET / HTTP/1.1" 200 -
INFO:werkzeug:127.0.0.1 - - [13/May/2025 14:33:44] "GET /favicon.ico
HTTP/1.1" 404 -
/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.
  warnings.warn(
WARNING:transformers_modules.tiiuae.falcon-
7b.ec89142b67d748a1865ea4451372db8313ada0d8.configuration_falcon:
WARNING: You are currently loading Falcon using legacy code contained in the
model repository. Falcon has now been fully ported into the Hugging Face
transformers library. For the most up-to-date and high-performance version of
the Falcon model code, please update to the latest version of transformers and
then load the model without the trust_remote_code=True argument.


Loading checkpoint shards:   0%|          | 0/2 [00:00<?, ?it/s]
INFO:werkzeug:127.0.0.1 - - [13/May/2025 14:33:53] "POST /analyze
HTTP/1.1" 400 -
INFO:werkzeug:127.0.0.1 - - [13/May/2025 14:34:15] "POST /upload_file HTTP/1.1"
200 -
You are using an old version of the checkpointing format that is deprecated (We
will also silently ignore `gradient_checkpointing_kwargs` in case you passed
it).Please update to the new format on your modeling file. To use the new
format, you need to completely remove the definition of the method
`_set_gradient_checkpointing` in your model.
INFO:werkzeug:127.0.0.1 - - [13/May/2025 14:35:25] "POST /load_model HTTP/1.1"
200 -
/usr/local/lib/python3.11/dist-packages/bitsandbytes/autograd/_functions.py:315:
UserWarning: MatMul8bitLt: inputs will be cast from torch.float32 to float16
during quantization
  warnings.warn(f"MatMul8bitLt: inputs will be cast from {A.dtype} to float16
during quantization")
INFO:werkzeug:127.0.0.1 - - [13/May/2025 14:37:01] "POST /analyze HTTP/1.1" 200
-
```

```
INFO:werkzeug:127.0.0.1 - - [13/May/2025 14:41:30] "POST /analyze HTTP/1.1" 200
-
INFO:werkzeug:127.0.0.1 - - [13/May/2025 14:41:53] "POST /upload_file HTTP/1.1"
200 -
INFO:werkzeug:127.0.0.1 - - [13/May/2025 14:42:17] "POST /analyze HTTP/1.1" 200
-
INFO:werkzeug:127.0.0.1 - - [13/May/2025 14:42:38] "POST /analyze HTTP/1.1" 200
-
INFO:werkzeug:127.0.0.1 - - [13/May/2025 14:43:07] "POST /analyze HTTP/1.1" 200
-
INFO:werkzeug:127.0.0.1 - - [13/May/2025 15:09:44] "POST /analyze HTTP/1.1" 200
-
INFO:werkzeug:127.0.0.1 - - [13/May/2025 15:10:04] "POST /analyze HTTP/1.1" 200
-
INFO:werkzeug:127.0.0.1 - - [13/May/2025 15:10:38] "POST /analyze HTTP/1.1" 200
-
```