**Abstract (100 Words)**

In Week 4 of my STEM research internship at Caldwell University, I developed and deployed a full-stack application for the Mistral-7B chatbot, specializing in mental health and medical domains. Over 20 hours, I built a React.js and Tailwind CSS frontend, deployed on Vercel at<https://mental-health-chatbot-blond.vercel.app/>, and integrated it with a Flask API backend hosted on Google Colab Pro+ via ngrok. The system features responsive chatbot interfaces, real-time API interactions, performance metrics, and safety disclaimers. I ensured robust logging, CORS configuration, and ethical compliance, delivering an accessible, user-friendly platform for AI-driven health support.

**Frontend:**<https://mental-health-chatbot-blond.vercel.app/>

**Google Collab Link:**<https://colab.research.google.com/drive/1PQ2llGgpL8aOzSn5iPOqUeupvXWm2isL#scrollTo=qIF7LTZPV6EV&line=3&uniqifier=1>

**GitHub Link:**<https://github.com/I-VAGAT/STEMRESEARCH>

**GitHub Frontend Link:**https://github.com/I-VAGAT/STEMRESEARCH\_WEBSITE.git

**Project Overview**

In Week 4, I focused on deploying a complete full-stack application for the Mistral-7B chatbot, fine-tuned for mental health and medical question-answering. The frontend, built with React.js and Tailwind CSS, is deployed on Vercel, while the backend, a Flask API running on Google Colab Pro+ with an A100 GPU, is exposed via ngrok. The frontend includes a homepage and dedicated chatbot interfaces for both domains, with real-time chat, toast notifications, and performance metrics. The backend handles API requests, model inference, and logging. I spent 20 hours on development, integration, testing, and deployment, ensuring a professional, safe, and responsive user experience.

**Objectives**

* **Frontend Development**: Create a React.js and Tailwind CSS-based website with a homepage, mental health, and medical chatbot interfaces.
* **Backend Deployment**: Deploy the Flask API on Colab Pro+ with ngrok for public access, handling /api/chat and /api/status endpoints.
* **Full-Stack Integration**: Connect the frontend to the backend for seamless real-time interactions.
* **Frontend Deployment**: Host the frontend on Vercel at<https://mental-health-chatbot-blond.vercel.app/>.
* **User Experience**: Implement a responsive UI with toast notifications, example questions, and safety disclaimers.
* **Performance Visualization**: Display evaluation metrics (e.g., empathy, precision, F1) in a professional format.
* **Logging and Monitoring**: Set up robust logging for debugging and interaction tracking.
* **Ethical Compliance**: Include disclaimers and crisis resources (e.g., 911, 988 Lifeline) for responsible use.
* **Testing**: Validate functionality, responsiveness, and API reliability across devices.

**Methodology**

I developed the full-stack application by combining a React.js frontend with a Flask backend, leveraging the provided backend code for API functionality. Below is the detailed methodology:

1. **Backend Setup and Deployment**:
   * **Environment**: Used Google Colab Pro+ with an A100 GPU for high-performance model inference.
   * **Flask API**: Configured a Flask app to serve two endpoints:
     + /api/chat (POST): Accepts a JSON payload with a user query, processes it using the MistralChatbot class, and returns a response with domain, confidence, and safety metadata.
     + /api/status (GET): Returns the API’s health status, including model loading confirmation.
   * **Ngrok**: Set up ngrok to expose the Flask app publicly, using an authentication token for secure tunneling. The public URL (e.g., https://<ngrok-id>.ngrok-free.app) was logged for frontend integration.
   * **CORS**: Enabled CORS in Flask to allow requests from the Vercel-hosted frontend domain.
   * **Logging**: Implemented a custom logging system to save interaction details (queries, responses, errors) to timestamped files in /content/drive/My Drive/mistral\_mental\_medical\_chatbot/logs.
   * **Model Loading**: Initialized and loaded the MistralChatbot with fine-tuned Mistral-7B and routing classifier models from Google Drive.
2. **Frontend Development**:
   * **Framework**: Used React.js with Vite for fast development and Tailwind CSS for utility-first styling.
   * **Structure**: Organized the frontend into four key components:
     + HomePage.js: A landing page with hero, features, stats, case studies, and research overview sections.
     + ChatbotInterface.js: A reusable component for chatbot interfaces, handling API connections, chat UI, and toast notifications.
     + MentalHealthChatbot.js: Configures ChatbotInterface for mental health with pink accents and specific metrics.
     + MedicalChatbot.js: Configures ChatbotInterface for medical queries with blue accents and metrics.
   * **UI Features**:
     + Responsive design with Tailwind’s breakpoints (e.g., sm:, lg:).
     + Lucide React icons (e.g., Heart, Stethoscope) for visual consistency.
     + Toast notifications for connection status, errors, and warnings.
     + Example question buttons to guide users (e.g., “I’m feeling anxious” for mental health).
     + Safety disclaimers and crisis resources (911, 988 Lifeline) on all pages.
   * **Performance Metrics**: Displayed domain-specific metrics (e.g., empathy: 0.8669 for mental health, precision: 0.3528 for medical) with progress bars and strengths.
3. **Full-Stack Integration**:
   * **API Connectivity**: Used fetch in ChatbotInterface.js to send queries to /api/chat and check status via /api/status.
   * **Dynamic Endpoint**: Allowed users to input the ngrok URL in a connection form, with validation and toast feedback.
   * **Response Handling**: Parsed API responses to display messages, domain (mental health/medical), confidence scores, and safety flags in the chat UI.
   * **Error Management**: Implemented try-catch blocks and toast notifications for API errors, empty queries, or connection issues.
4. **Deployment**:
   * **Frontend**: Deployed to Vercel using Vercel CLI (vercel --prod) at<https://mental-health-chatbot-blond.vercel.app/>. Configured automatic builds via GitHub integration.
   * **Backend**: Hosted on Colab Pro+, with ngrok providing a temporary public URL. Ensured the Flask app ran in a background thread to avoid blocking.
   * **CORS Configuration**: Set CORS(app) in Flask to allow requests from the Vercel domain, resolving cross-origin issues.
   * **Persistence**: Saved models and logs to Google Drive for reliability across Colab sessions.
5. **Testing and Validation**:
   * **Frontend**: Tested responsiveness on Chrome, Firefox, Safari, and mobile devices (iPhone, Android) using Tailwind’s responsive classes.
   * **Backend**: Verified API endpoints with Postman, sending sample queries (e.g., “What are diabetes symptoms?”) and checking /api/status.
   * **Integration**: Tested end-to-end flow by entering the ngrok URL, connecting, and sending queries, ensuring responses matched expected domains and included safety metadata.
   * **Logging**: Reviewed logs in /logs/web\_chatbot\_\*.log for query-response pairs, errors, and confidence scores.
   * **Safety**: Confirmed disclaimers and crisis resources were visible, with backend safety flags triggering UI alerts for sensitive queries.
6. **Ethical Considerations**:
   * Included prominent disclaimers: “For educational purposes only, consult healthcare professionals.”
   * Integrated crisis resources (e.g., 988 Lifeline) in the footer and chat UI for mental health queries.
   * Flagged responses with safety\_applied: true in the UI with an orange alert icon, ensuring transparency.
   * Logged sensitive interactions for potential manual review to address ethical concerns.

**Time Breakdown**

I spent **20 hours** in Week 4, distributed as follows:

* **Frontend Development (10 hours)**: Built React components, styled with Tailwind CSS, and designed UI features.
* **Backend Setup (4 hours)**: Configured Flask app, ngrok, logging, and model loading on Colab Pro+.
* **Full-Stack Integration (2 hours)**: Connected frontend to backend, implemented API calls, and added error handling.
* **Deployment (1 hours)**: Deployed frontend to Vercel, set up ngrok for backend, and configured CORS.
* **Testing and Validation (2 hours)**: Tested responsiveness, API functionality, and safety features.
* **Documentation (1 hour)**: Drafted this report and organized logs/artifacts.

**Code Structure and Explanation**

Below, I discuss the provided backend code’s imports and functionality, followed by the frontend structure.

**Backend Imports and Their Role**

The provided Flask backend code (app.py) uses several Python libraries to support the chatbot API. Here’s a detailed breakdown of each import:

1. **import os**:
   * **Purpose**: Provides functions for interacting with the operating system, such as creating directories and handling file paths.
   * **Usage**: Used in setup\_logging() to create the log directory (os.makedirs(log\_dir, exist\_ok=True)) and ensure paths are correctly formatted for Google Drive (/content/drive/My Drive/...).
   * **Why Critical**: Ensures platform-independent file operations, crucial for saving logs and accessing models in Colab’s Linux environment.
2. **import sys**:
   * **Purpose**: Allows manipulation of the Python runtime environment, particularly the module search path.
   * **Usage**: Adds the chatbot directory to the system path (sys.path.append('/content/drive/My Drive/mistral\_mental\_medical\_chatbot')) to import the custom MistralChatbot module.
   * **Why Critical**: Enables importing the MistralChatbot class, which is not in the default Python path, ensuring the Flask app can access the chatbot logic.
3. **import logging**:
   * **Purpose**: Provides a flexible framework for emitting log messages from the application.
   * **Usage**: Configures a logger in setup\_logging() to save interaction details (queries, responses, errors) to timestamped files with a specific format (%(asctime)s - %(name)s - %(levelname)s - %(message)s).
   * **Why Critical**: Facilitates debugging and monitoring by logging API requests, model errors, and interaction metadata, stored persistently on Google Drive.
4. **from datetime import datetime**:
   * **Purpose**: Handles date and time operations for timestamping.
   * **Usage**: Generates timestamps for log files (datetime.now().strftime("%Y%m%d\_%H%M%S")) and status responses (datetime.now().isoformat()).
   * **Why Critical**: Ensures logs and API responses are time-stamped for traceability and debugging.
5. **import json**:
   * **Purpose**: Supports JSON data parsing and serialization.
   * **Usage**: Used implicitly by Flask’s request.get\_json() and jsonify() to handle JSON payloads in /api/chat and return JSON responses.
   * **Why Critical**: Enables communication between frontend and backend via JSON, the standard format for API data exchange.
6. **import asyncio**:
   * **Purpose**: Provides support for asynchronous programming using coroutines.
   * **Usage**: Not directly used in the provided code, suggesting potential for future async operations (e.g., non-blocking model inference).
   * **Why Included**: Likely a placeholder for optimizing chatbot response generation, as Mistral-7B inference could benefit from async I/O in high-traffic scenarios.
7. **import threading**:
   * **Purpose**: Enables running multiple threads for concurrent execution.
   * **Usage**: Runs the Flask server in a background thread (threading.Thread(target=run\_flask, daemon=True)) to avoid blocking the main thread in Colab.
   * **Why Critical**: Allows the Flask app to serve requests while keeping the Colab notebook interactive for debugging or monitoring.
8. **from flask import Flask, request, jsonify, render\_template\_string**:
   * **Purpose**: Core Flask components for building a web API.
   * **Flask**: Creates the web application instance (app = Flask(\_\_name\_\_)).
   * **request**: Accesses incoming request data (e.g., request.get\_json() for query payloads).
   * **jsonify**: Returns JSON responses with proper headers.
   * **render\_template\_string**: Renders a simple HTML home page for the root endpoint (/).
   * **Why Critical**: Forms the backbone of the API, handling HTTP requests and responses for chatbot interactions.
9. **from flask\_cors import CORS**:
   * **Purpose**: Enables Cross-Origin Resource Sharing to allow requests from different domains.
   * **Usage**: Applied to the Flask app (CORS(app)) to permit frontend requests from the Vercel domain (<https://mental-health-chatbot-blond.vercel.app/>).
   * **Why Critical**: Resolves CORS errors, ensuring the frontend can communicate with the backend hosted on a different domain (ngrok).
10. **from pyngrok import ngrok**:
    * **Purpose**: Creates secure public tunnels to expose local servers.
    * **Usage**: Sets up a tunnel on port 5000 (ngrok.connect(5000, bind\_tls=True)) and authenticates with a token to provide a public URL.
    * **Why Critical**: Makes the Colab-hosted Flask app accessible to the frontend and external clients, overcoming Colab’s local-only limitation.
11. **import torch**:
    * **Purpose**: Provides the PyTorch framework for deep learning operations.
    * **Usage**: Used implicitly by MistralChatbot to load and run the fine-tuned Mistral-7B model and routing classifier on the A100 GPU.
    * **Why Critical**: Essential for model inference, leveraging GPU acceleration for efficient response generation.
12. **from huggingface\_hub import login**:
    * **Purpose**: Authenticates with the Hugging Face Hub for model access.
    * **Usage**: Logs in with a token (login(token="hf\_lOZwgAeNfoSnSsMtdcxcQCkXFCyzGlIeaB")) to access the Mistral-7B model or related resources.
    * **Why Critical**: Ensures secure access to Hugging Face models and datasets, required for initializing MistralChatbot.

**Backend Code Functionality**

The backend code sets up a robust API for the Mistral-7B chatbot:

1. **Logging Setup (setup\_logging)**:
   * Creates a timestamped log file in a Google Drive directory.
   * Configures a logger with INFO level, formatting messages with timestamp, module name, and level.
   * Saves logs for queries, responses, domains, confidence scores, and errors, aiding debugging and monitoring.
2. **Chatbot Initialization**:
   * Instantiates MistralChatbot with paths to the fine-tuned model and routing classifier.
   * Loads models using chatbot.load\_models(), leveraging PyTorch and the A100 GPU.
   * Logs success or failure, with exceptions raised to halt execution if initialization fails.
3. **Flask API**:
   * **Root Endpoint (/)**: Renders a simple HTML page with API details and endpoint documentation.
   * **Chat Endpoint (/api/chat)**:
     + Accepts POST requests with a JSON payload containing a user query.
     + Validates the query, returning 400 errors for missing or empty inputs.
     + Calls chatbot.generate\_response(query) to generate a response with domain, confidence, and safety metadata.
     + Logs the interaction and returns the response as JSON.
     + Handles errors with 500 responses and detailed logging (exc\_info=True).
   * **Status Endpoint (/api/status)**:
     + Returns a JSON object indicating API health, model loading status, and timestamp.
     + Logs errors if the status check fails.
4. **Ngrok Tunnel (setup\_ngrok)**:
   * Kills existing ngrok processes to avoid conflicts.
   * Authenticates with a token and creates a secure tunnel on port 5000.
   * Extracts and logs the public URL for frontend integration.
   * Returns None on failure, with errors printed for debugging.
5. **Server Execution**:
   * Runs the Flask app in a background thread to keep Colab interactive.
   * Logs startup details and the ngrok URL, confirming readiness for requests.

**Frontend Code Structure**

The frontend, built with React.js and Tailwind CSS, is modular and reusable:

1. **ChatbotInterface.js**:
   * A core component handling API connections, chat UI, and toast notifications.
   * Features:
     + Connection form for entering the ngrok URL, with a test button (/api/status).
     + Chat window with message bubbles, timestamps, and metadata (domain, confidence, safety).
     + Toast system for success (connection), errors (API failures), and warnings (invalid inputs).
     + Example question buttons and a disclaimer footer.
   * Uses Tailwind for styling (e.g., bg-gradient-to-r, rounded-2xl) and Lucide icons.
2. **MentalHealthChatbot.js**:
   * Configures ChatbotInterface with:
     + Pink accent color, heart icon, and mental health examples (e.g., “I’m struggling with depression”).
     + Metrics: empathy (0.8669), overall score (9/10).
     + Strengths: empathetic phrases, crisis resources (988 Lifeline), actionable advice.
3. **MedicalChatbot.js**:
   * Configures ChatbotInterface with:
     + Blue accent color, stethoscope icon, and medical examples (e.g., “What causes migraines?”).
     + Metrics: precision (0.3528), F1 (0.3846), overall score (6-7/10).
     + Strengths: clear disclaimers, accurate high-scoring responses.
4. **HomePage.js**:
   * A comprehensive landing page with:
     + Hero section with animated orbs, medical pattern, and CTAs for chat pages.
     + Stats (e.g., 16K+ mental health Q&A pairs), features (empathy, accuracy, safety), and case studies (e.g., anxiety support).
     + Research overview detailing objectives, technical implementation, and outcomes.
     + Footer with quick links, disclaimers, and crisis resources.

**Results**

* **Deployment**: Frontend live at<https://mental-health-chatbot-blond.vercel.app/>; backend accessible via ngrok on Colab Pro+.
* **Functionality**: Real-time chat with domain-specific responses, confidence scores, and safety flags.
* **User Experience**: Responsive UI with intuitive navigation, toast feedback, and clear metrics (9/10 mental health, 6-7/10 medical).
* **Logging**: Interaction details saved to Google Drive logs, including queries, responses, and errors.
* **Safety**: Disclaimers and crisis resources (911, 988) prominent; safety flags displayed in UI.
* **Performance**: Backend handles A100 GPU inference efficiently; frontend renders smoothly across devices.

**Challenges Addressed**

1. **Ngrok Instability**: Managed dynamic ngrok URLs with a frontend input form and validation, ensuring connectivity.
2. **CORS Errors**: Configured CORS(app) to allow Vercel domain requests, resolving cross-origin issues.
3. **Model Loading**: Handled GPU memory constraints by sequential model loading in MistralChatbot.
4. **Error Handling**: Implemented robust backend error logging (exc\_info=True) and frontend toasts for user feedback.
5. **Colab Persistence**: Saved models and logs to Google Drive, mitigating session resets.

**Future Improvements**

* **Backend Hosting**: Migrate Flask app to AWS or Heroku for stable, non-Colab hosting.
* **Async Inference**: Utilize asyncio for non-blocking model inference to handle multiple requests.
* **User Analytics**: Track interactions via backend logging or frontend analytics (e.g., Google Analytics).
* **Accessibility**: Enhance WCAG compliance with ARIA labels and keyboard navigation.
* **Medical Accuracy**: Retrain Mistral-7B with additional medical data to improve precision and F1 scores.

**Conclusion**

In Week 4, I deployed a full-stack Mistral-7B chatbot application, with a React.js and Tailwind CSS frontend on Vercel (<https://mental-health-chatbot-blond.vercel.app/>) and a Flask API backend on Colab Pro+ via ngrok. The system offers responsive, domain-specific chat interfaces, robust logging, and ethical safeguards. The backend’s imports (os, sys, logging, etc.) enable reliable file handling, module imports, and monitoring, while the frontend delivers a professional user experience. This deployment validates the chatbot’s capabilities (9/10 mental health, 6-7/10 medical) and sets the stage for further enhancements.

**References**

1. **Python Libraries**:
   * [os](https://docs.python.org/3/library/os.html)
   * [sys](https://docs.python.org/3/library/sys.html)
   * [logging](https://docs.python.org/3/library/logging.html)
   * [datetime](https://docs.python.org/3/library/datetime.html)
   * [json](https://docs.python.org/3/library/json.html)
   * [asyncio](https://docs.python.org/3/library/asyncio.html)
   * [threading](https://docs.python.org/3/library/threading.html)
   * [Flask](https://flask.palletsprojects.com/)
   * [flask-cors](https://flask-cors.readthedocs.io/)
   * [pyngrok](https://pyngrok.readthedocs.io/)
   * [torch](https://pytorch.org/)
   * [huggingface\_hub](https://huggingface.co/docs/hub/)
2. **Frontend**:
   * [React.js](https://react.dev/)
   * [Tailwind CSS](https://tailwindcss.com/)
   * [Vercel](https://vercel.com/docs)
   * [Lucide Icons](https://lucide.dev/)
3. **Mistral-7B**: Jiang, A. Q., et al. (2023). "Mistral 7B." *arXiv:2310.06825*. [Link](https://arxiv.org/abs/2310.06825)