Medical AI Assistant

1.Introduction

Project Title : Health AI

Project Member : Rangeetha.R

Project Member : Yogalakshmi.

Project Member : Deepika.B

2. Project Review

The Medical AI Assistant is an AI-powered web application built using Gradio, PyTorch, and Hugging Face Transformers. Its main purpose is to provide informational healthcare support, such as predicting possible diseases based on symptoms and generating general treatment plans. The project uses the IBM Granite LLM for generating natural language responses tailored to medical queries. It is not a replacement for a doctor but serves as a knowledge assistant for users to get quick insights. The app features a user-friendly interface with two tabs: one for disease prediction and another for treatment suggestions. Each response emphasizes the importance of consulting healthcare professionals for accurate diagnosis and treatment. The project highlights how AI models can be integrated into practical applications to assist in healthcare. It also demonstrates how open-source frameworks like Gradio can simplify deployment of interactive AI applications. Overall, this project is a proof-of-concept for AI-assisted healthcare guidance. The project can also be used as a learning tool for students exploring medical AI applications.It showcases how ethical design choices, like disclaimers, can build user trust.It demonstrates how open-source AI models can be adapted to domain-specific needs.

3. Architecture

The architecture of the Medical AI Assistant is designed in a modular and layered structure. At the core, the IBM Granite model is used for generating responses based on input prompts. The model and tokenizer are loaded through Hugging Face Transformers, which provide efficient preprocessing and text generation. The logic layer contains functions like disease_prediction and treatment_plan, which structure prompts before sending them to the model. On top of this, the Gradio Blocks UI handles input, output, and user interaction in a simple browser-based interface. The application runs locally or via a shareable link, thanks to Gradio's built-in server. The architecture ensures scalability as additional tabs or functions can easily be added. It separates data handling, model inference, and UI into cleanly defined components. This modular approach makes the system easy to understand, maintain, and extend. The design supports integration with databases or APIs in the future. It is flexible enough to allow model replacement if a better healthcare LLM becomes available.The architecture balances simplicity and extensibility, ensuring it works for both demos and real-world expansions.

4. Setup Instructions

To set up the project, you need Python 3.8 or higher installed on your system. Install the required libraries using pip install gradio torch transformers. Once the dependencies are installed, clone or copy the project code into a working folder. Ensure that your system has a GPU (CUDA-enabled) for faster inference, although the code will also work on CPU with reduced speed. The IBM Granite model will be automatically downloaded from Hugging Face Hub when you run the application for the first time. Make sure you have a stable internet connection during the first run, as the model files can be large. If running on limited resources, adjust the max_length or use a smaller model. After setup, the application can be launched by executing the script with python app.py. Once started, you will get a local URL and a shareable public link. You can also create a virtual environment to isolate and manage dependencies. Using a requirements.txt file ensures easy reinstallation of libraries on other systems.The setup process does not require additional tools like Docker, but containerization can be added later for portability.

5. Folder Structure

A simple folder structure can be followed for this project:

/medical_ai_assistant → Main project folder

app.py → The main script containing the model, functions, and Gradio UI.

requirements.txt → List of required dependencies.

/models/ → Cache folder where Hugging Face stores the Granite model.

/docs/ → Project documentation and manuals.

/tests/ → Testing scripts for unit and functional tests.

A dedicated /assets/ folder can be created for images, UI icons, or resources.

A /logs/ directory can be maintained to capture debugging and runtime logs.

Structuring this way ensures professional organization and smoother collaboration in teams.

This structure ensures clarity and helps organize code, models, and documentation separately. It also supports future scalability, as more features can be added under their own subdirectories.

The separation of files promotes clean coding practices and easier collaboration.

6. Running the Application

To run the application, ensure that setup has been completed and all dependencies are installed. Execute the script using python app.py in your terminal or IDE. Once running, Gradio will start a local server and display a URL in the terminal (e.g., http://127.0.0.1:7860). You can open this link in your browser to access the Medical AI Assistant. Additionally, a shareable public link will be generated if share=True is set, allowing remote access. The interface will show two tabs: Disease Prediction and Treatment Plans. Users can enter symptoms, conditions, and personal details to generate responses. The app remains active until the terminal session is closed. This lightweight deployment makes it easy to demonstrate and share without needing complex web servers. The program can also be executed on Google Colab for cloud-based usage. Advanced users may deploy it on cloud platforms like AWS or Azure for production. Running the app on GPU significantly improves performance compared to CPU-only setups.

7. API Document

The project does not expose a REST API by default but uses Gradio's built-in API layer for function calls. Two main functions form the application's core: disease_prediction(symptoms) and treatment_plan(condition, age, gender, medical_history). Both functions internally call the generate_response(prompt) function, which interacts with the Granite model. Inputs are text-based, and outputs are AI-generated text strings. For external usage, Gradio provides an automatic API endpoint that can be consumed programmatically. Developers can also extend the app by creating a Flask/FastAPI wrapper if needed. This design ensures flexibility for both end-users (via UI) and developers (via API). Overall, the API is simple, text-in/text-out, with AI inference happening in the backend. Inputs are validated with truncation to prevent exceeding model limits. The API can be extended with error handling and detailed logging for developers. A structured API design allows future integration with electronic health systems

8. Authentication

Currently, the application does not implement user authentication, as it is intended for local or demo usage. Any user accessing the public link can use the application. For production environments, authentication methods like API keys, OAuth2, or password protection should be added. Hugging Face handles model access, so no additional authentication is required for model downloads. In case of enterprise deployment, security layers should be implemented to protect sensitive medical inputs. Role-based access control (RBAC) could also be integrated if healthcare professionals are intended users. This project demonstrates functionality but leaves security and authentication as future enhancements. Proper authentication would also help in tracking and monitoring usage in real deployments. Security can be improved by enforcing HTTPS encryption for data transfer. Storing user sessions securely would prevent data leaks in multi-user environments. Audit logs can be introduced to monitor and track usage activity.

9. User Interface

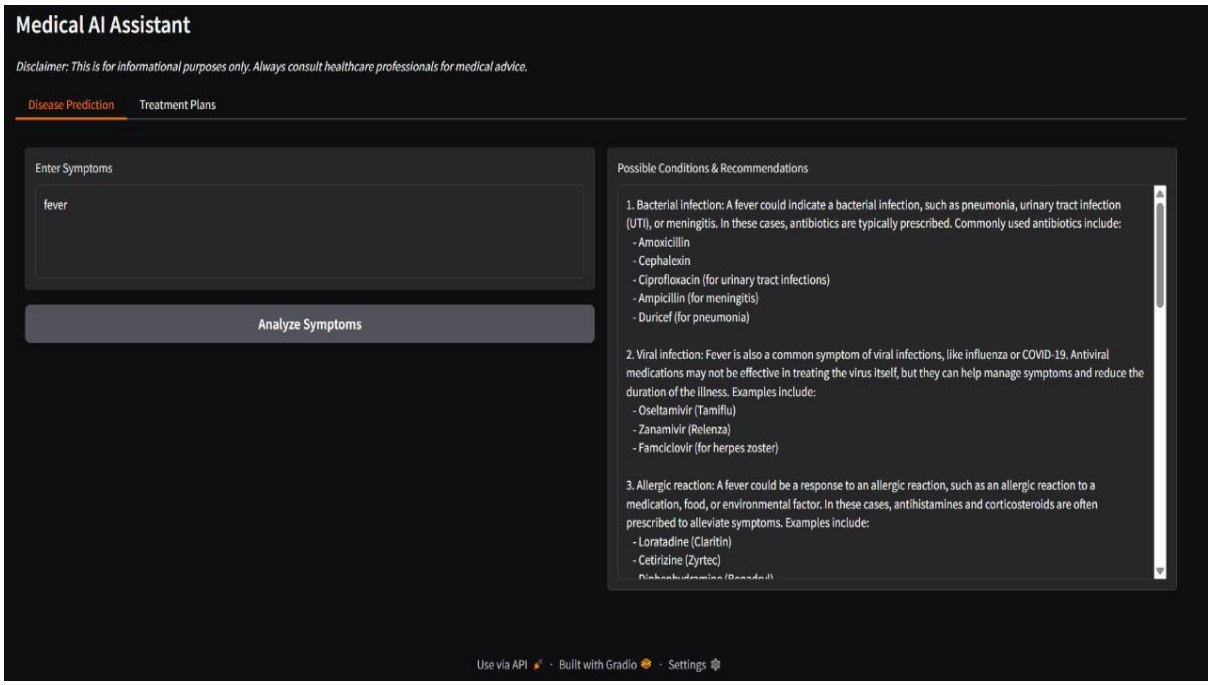
The user interface is built with Gradio Blocks, making it intuitive and user-friendly. It contains a simple layout with tabs, rows, and columns for better organization. The first tab lets users input symptoms and get possible disease predictions. The second tab provides treatment plans based on condition, age, gender, and medical history. Textboxes are used for input and output, and buttons trigger the AI model. The design is clean, lightweight, and mobile-friendly. Important disclaimers are shown clearly at the top to ensure users understand the tool's limitations. Since Gradio automatically adapts to the browser, the UI works on desktops, tablets, and phones. The minimal design keeps the focus on functionality and clarity. Developers can customize the interface with colors, themes, or extra widgets. Adding icons or medical illustrations can improve engagement and usability. The UI design ensures a balance of simplicity, clarity, and professional appearance.

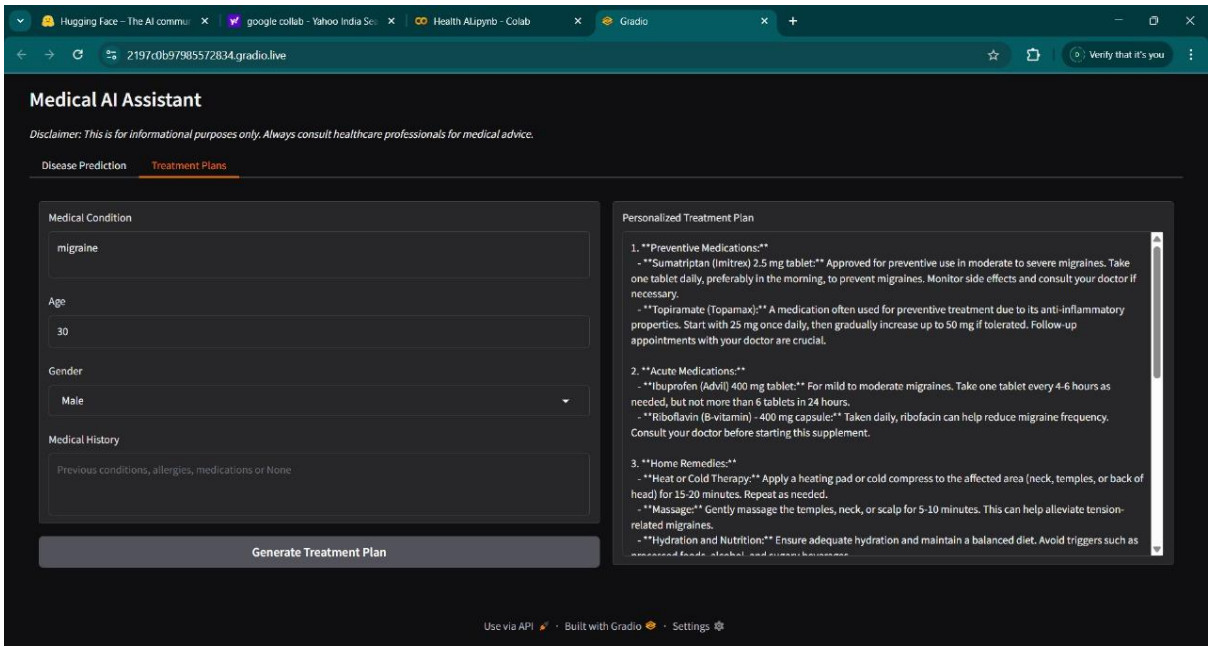
10. Testing

Testing for the application involves both functional and user-level checks. Functional testing ensures that the disease_prediction and treatment_plan functions correctly call the model and return outputs. Unit testing can be performed using Python's unittest or pytest frameworks. Load testing helps verify performance when multiple users connect via the shareable link. Usability testing with sample users checks if the interface is intuitive and easy to navigate. Since AI outputs vary, test cases should focus on response consistency, completeness, and disclaimers. Testing also involves checking error handling, such as empty inputs or very long texts. Overall, testing aims to ensure the app remains stable, useful, and safe. Continuous feedback from users can also guide improvements and bug fixes. Automated testing pipelines can be added to a CI/CD workflow. Regression testing ensures that updates

do not break existing functionality. End-to-end tests validate the complete workflow from input to AI response.

11. Screenshots





12. Known Issues

There are a few known limitations in the current version of the application. The Granite model responses may sometimes be too general or repetitive. Since it is not a medical-grade AI, it may miss certain rare conditions. Running on CPU can be slow due to the model's large size. There is no authentication system, making the public link open to anyone. The application does not store user history, so all interactions are session-based. Long medical histories may exceed the model's maximum input length. Internet is required initially for downloading the model from Hugging Face. These issues highlight areas for improvement and optimization in future releases. The application may not function properly in offline environments due to model dependencies. Hugging Face outages or model unavailability could temporarily affect usage. Some outputs may mix medical advice with general lifestyle tips, which might confuse users.

13. Future Enhancements

Several enhancements can be added to improve the project. Implementing user authentication and access control would make the app more secure. Integration with medical databases or symptom checkers could provide more accurate results. A multi-language interface would allow global usage. Support for speech input and output could improve accessibility. Integration with FastAPI or Flask could expose REST APIs for broader use. Adding analytics and usage reports would help track application impact. Optimizing the model for mobile or edge devices could increase availability. Eventually, partnerships with healthcare providers could make it a trusted companion tool. Future work can turn this prototype into a real-world healthcare assistant with professional validation. Integration of personal health records could provide more personalized advice. Adding AI explainability features (like reasoning steps) would increase transparency. Professional medical validation and certification could make it a trustworthy tool in healthcare.