HealthAI: Intelligent Healthcare Assistant Using IBM Granite

Project Documentation

1. Introduction

• Project title : Generative AI in Health AI – Medical Assistant

• Team leader : A.Silambarasan.

• Team member : BalaBarathi.M

• Team member : S.HariHaran

• Team member : M.kisoth

2. Project Overview

• **Purpose :**

The purpose of this project is to develop a Medical AI Assistant using Generative AI that can provide general health information to users. The system allows users to enter symptoms or details of their condition and generates possible disease predictions along with non-prescriptive treatment suggestions. The project does not replace professional diagnosis but helps raise awareness and encourages patients to consult qualified doctors. It demonstrates the role of generative AI in health education and medical awareness.

• **Features:**

Disease Prediction
Key Point: Symptom-based analysis
Functionality: Users enter symptoms such as fever, cough, headache etc. The assistant generates possible conditions and provides general recommendations.

Treatment Plan Generation
Key Point: General guidance for patients

Functionality: Based on condition, age, gender, and medical history, the assistant produces a personalized treatment overview including home remedies and lifestyle tips.

Generative AI Model Integration
Key Point: Natural language understanding
Functionality: Uses IBM Granite LLM for producing human-like medical responses and suggestions.

User-friendly Interface
Key Point: Accessible to all users
Functionality: Gradio interface with tabbed layout for easy navigation and outputs.

3. Architecture

Frontend (Gradio):

The frontend is designed using Gradio Blocks, providing a simple browser-based user interface. It contains tabs for different functionalities, disclaimers for safety, and clear text input and output areas.

Backend (Python with Hugging Face Transformers):

The backend handles model loading, prompt design, and response generation. It uses the IBM Granite instruction-tuned language model for text generation and response handling.

System Flow:
User enters input → Tokenizer processes input → Granite Model generates output → Output shown on UI.

4. Setup Instructions

Prerequisites:

Python 3.8 or above
pip package manager
Internet connection to download the model

Installation Process:

Clone the repository from GitHub
Install the required dependencies from requirements.txt
Run the application using app.py
Launch the Gradio link in the browser
Enter symptoms or condition details and receive outputs

5. Folder Structure

app.py – Main program that integrates model and UI
requirements.txt – Dependency file for Python packages
report.docx – Project documentation
screenshots – Folder containing sample outputs and interface images
deployment_link.txt – File containing deployed application link

6. Running the Application

To start the application:

Run the app.py file from the terminal.
Wait for the model to load and the Gradio server to start.
Open the local or share link provided in the terminal.
Navigate between tabs to use the "Disease Prediction" and "Treatment Plan" features.

7. API Documentation

The project is implemented with Gradio and does not require separate API endpoints. The functionality is provided through two core methods:

Disease Prediction – Generates possible health conditions from symptoms.
Treatment Plan – Generates a general treatment overview from patient details.

8. Authentication

The current version does not include authentication.
Future versions may include user login, role-based access for doctors and patients, and privacy protection for sensitive health data.

9. User Interface

The user interface is designed to be simple and easy to use. It includes:

Tabbed sections for different features
Textbox input for symptoms and medical history
Dropdown menu for gender selection
Number input for age
Disclaimer highlighted in red for safety
Output areas displaying results clearly

10. Testing

Testing was carried out in the following ways:

Unit Testing – Functions for disease prediction and treatment plan were validated.
Manual Testing – Multiple test cases were run with different symptom sets and conditions.
Edge Case Testing – Inputs with empty fields, very long text, or unusual cases were tested.
The results confirmed that the assistant provides meaningful outputs, though not always medically accurate, as expected.

11.Screenshots

Screenshots include:

UI of the application
Disease prediction output
Treatment plan output

12.Known Issues

Model can sometimes give vague or repetitive responses.
Large model size makes it slower to load on low-resource systems.
Responses may not always be medically accurate, as LLMs are not trained for precise diagnosis.

13.Future Enhancement

Add speech-to-text input for accessibility.
Integrate smaller optimized models for faster responses.
Extend the system to support image-based health reports such as X-rays.
Add authentication and data privacy features for real deployment.
Collaborate with healthcare professionals to validate responses and improve reliability.