Project Title:Health AI:Intelligent Health Care Assistant

Team leader: S. Keerthana

Team member: M. Vimala

Team member: A. Aarthi

Team member: S. Ramya

1.Introduction

Health AI leverages artificial intelligence to revolutionize healthcare by enhancing diagnostics, personalizing treatment plans, and streamlining clinical workflows. It combines machine learning, natural language processing, and data analytics to:

Improve patient outcomes through predictive insights and early disease detection.

Optimize clinical decision-making with evidence-based recommendations.

Enhance patient engagement via virtual assistants and personalized health monitoring.

From interpreting medical images to analyzing patient data, Health AI is transforming the future of medicine, making healthcare more efficient, accessible, and precise. Let's explore how AI can drive innovation in health! Key applications include:

Predictive analytics for disease prevention

Personalized medicine tailored to individual needs Virtual

health assistants for patient support.

2.Project Overview

Key Features:

AI Health Chat: Provides instant responses to healthrelated queries, leveraging IBM Granite's advanced language capabilities.

Disease Prediction: Analyzes symptoms to predict potential conditions and recommends next steps.

Treatment Plans: Offers personalized medical recommendations based on diagnosed conditions.

Health Analytics: Visualizes health data, such as heart rate, blood pressure, and blood sugar levels, providing valuable insights.

Technologies Used:

IBM Granite Models: Utilize advanced natural language processing and machine learning capabilities.

Streamlit: A Python library for building user-friendly interfaces.

Python: A versatile programming language for developing the assistant's core functionality.

Benefits:

Enhanced Patient Engagement: Provides 24/7 support, empowering patients to take control of their health.Improved Diagnostic Accuracy: Assists doctors in making informed decisions with evidence-based treatment options.

Streamlined Clinical Workflows: Automates administrative tasks, reducing workload and improving efficiency.

Project Goals:

Personalized Medicine: Tailors treatment plans to individual genetic profiles and medical histories.

Predictive Analytics: Identifies high-risk patients and enables early interventions.

Virtual Health Assistants: Provides instant responses to patient queries, improving engagement and satisfaction.

Real-World Applications:

Predictive Health Monitoring: Tracks vital signs and flags potential issues before they escalate.

Personalized Treatment Plans: Develops bespoke health strategies based on individual genetic data and lifestyle.

Enhanced Patient Care: Improves patient outcomes and streamlines healthcare

processes. **3.Architecture**

1. *Data Ingestion*: Collects and integrates data from various sources, such as electronic health records (EHRs), medical imaging, and wearable devices.

2. IBM Granite Models: Utilizes advanced natural language processing (NLP) and machine learning (ML) models for:

   Text Analysis: Analyzes clinical notes, medical literature, and patient data.

   Predictive Modeling: Predicts patient outcomes, disease progression, and treatment responses.

3. Application Layer: Provides a user-friendly interface for:

   Patient Engagement: Patients can access their health data, receive personalized recommendations, and interact with the assistant.

   Clinical Decision Support: Healthcare professionals receive evidence-based recommendations and insights to inform their decisions.

4. Integration Layer: Integrates with existing healthcare systems, such as EHRs, practice management systems, and health information exchanges (HIEs).

5. Security and Compliance: Ensures the confidentiality, integrity, and availability of sensitive patient data, complying with regulations like HIPAA.
6. Analytics and Insights: Provides data analytics and visualization capabilities to identify trends, patterns, and areas for improvement.

4.Setup Instructions

Prerequisites

Python installed on your system

IBM Granite model (downloaded locally)

Streamlit library for building the user interface

Step-by-Step Setup Instructions

1. Clone the Repository: Clone the Health AI repository from GitHub
2. Install Required Libraries: Install the necessary libraries by running `pip install streamlit`.
3. Download IBM Granite Model: Download the IBM Granite 3.3B model, which is used for natural language processing and machine learning tasks.
4. Configure API Keys: Set up your API keys securely to access IBM Watson services.
5. Run the Application: Launch the application using `streamlit run app.py`.

5.Folder Structure

Root Folder: HealthAI/

App FFil: `app.py` - The main application file that runs the Streamlit app

Shared Model File: `shared_model.py` - Handles shared model functionality

Patient Chat File: `patient_chat.py` - Manages patient chat functionality

Disease Prediction File: `disease_prediction.py` - Predicts diseases based on symptoms
Treatment Plans File: `treatment_plans.py` - Provides personalized treatment suggestions

Health Analytics File: `health_analytics.py` - Visualizes health data and provides insights.

6.Running the Application

Create a Virtual Environment: Run `python -m venv venv` and activate it using `venv\Scripts\activate` on Windows.

Install Dependencies: Install required libraries by running `pip install streamlit`.

Download IBM Granite Model: Download the IBM Granite 3.3B model and place it in the `granite/` folder.

Run the Application: Launch the app using `streamlit run app.py`.

7.API Documentation

Key Features

AI Health Chat: Answers general health queries using IBM Granite's natural language processing capabilities.

Disease Prediction: Predicts potential diseases based on symptoms and provides recommendations.

Treatment Plans: Offers personalized treatment suggestions based on diagnosed conditions.

Health Analytics: Visualizes health data, such as heart rate, blood pressure, and blood sugar levels, providing valuable insights.

Technical Details

IBM Granite Model: Utilizes the IBM Granite 3.3B model for natural language processing and machine learning tasks.

Streamlit Integration: Built using Streamlit for a user-friendly interface.

API Integration: Integrates with existing healthcare systems using RESTful APIs and FHIR endpoints for secure data exchange.

8.Authentication

- *Secure API Key Management*: The application uses secure API key management to access IBM Watson services, ensuring that sensitive data is protected.

- *Data Encryption*: The system likely employs end-to-end encryption to safeguard patient data, both in transit and at rest.

- *Access Control*: Role-based access control may be implemented to restrict access to authorized personnel only.

- *Compliance*: The application is designed to comply with relevant healthcare regulations, such as HIPAA, ensuring the confidentiality, integrity, and availability of sensitive patient data.

Security Measures

- *IBM Watson Integration*: The application leverages IBM Watson's advanced security features, including data encryption and secure authentication.

- *Streamlit Security*: Streamlit, the framework used to build the application, provides built-in security features, such as secure session management and encryption.

- *Regular Updates*: The application's developers likely prioritize regular updates and patches to ensure the security of the system.

9.User Interface

- *Intuitive Design*: The interface is built using Streamlit, offering an intuitive and easytonavigate design that caters to users with varying levels of technical expertise.

- *Health Chat*: A conversational interface allows users to ask health-related questions and receive instant responses powered by IBM Granite's natural language processing capabilities.

- *Symptom Checker*: Users can input symptoms to receive potential diagnoses and recommendations for next steps.

- *Personalized Recommendations*: The assistant provides tailored health advice and treatment plans based on individual health data and medical history.

- *Health Analytics*: Visualizations of health data, such as heart rate, blood pressure, and blood sugar levels, help users track their health progress.

10.Testing

Testing Objectives

Functional Testing: Verify that the assistant's features, such as health chat, symptom checker, and personalized recommendations, work as expected.

Accuracy Testing: Evaluate the accuracy of the assistant's responses, diagnoses, and recommendations.

Performance TestingPerformanc: Assess the assistant's response time, scalability, and reliability under various loads.

Testing Methods

Unit Testing: Test individual components, such as the health chat module or symptom checker, to ensure they function correctly.

Integration Testing: Verify that the different components work together seamlessly.

User Acceptance Testing (UAT): Conduct testing with real users to ensure the assistant meets their needs and expectations.

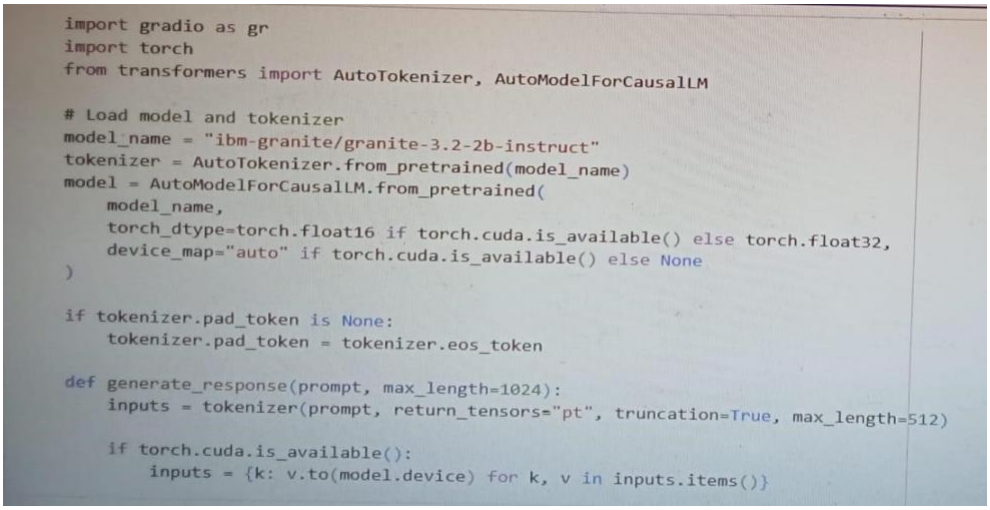
Testing Scenarios

Health Chat: Test the assistant's ability to respond accurately to various health-related queries.

Symptom Checker: Evaluate the assistant's ability to provide accurate diagnoses and recommendations based on symptoms.

Personalized Recommendations: Test the assistant’s ability to provide tailored health advice and treatment plans.

11.Screen Shots



```
import gradio as gr
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM

# Load model and tokenizer
model_name = "ibm-granite/granite-3.2-2b-instruct"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(
    model_name,
    torch_dtype=torch.float16 if torch.cuda.is_available() else torch.float32,
    device_map="auto" if torch.cuda.is_available() else None
)

if tokenizer.pad_token is None:
    tokenizer.pad_token = tokenizer.eos_token

def generate_response(prompt, max_length=1024):
    inputs = tokenizer(prompt, return_tensors="pt", truncation=True, max_length=512)

    if torch.cuda.is_available():
        inputs = {k: v.to(model.device) for k, v in inputs.items()}
```

```python
# Create Gradio interface
with gr.Blocks() as app:
    gr.Markdown("# Medical AI Assistant")
    gr.Markdown("**Disclaimer: This is for informational purposes only. Always consult healthcare professionals for medical advice.**")

    with gr.Tabs():
        with gr.TabItem("Disease Prediction"):
            with gr.Row():
                with gr.Column():
                    symptoms_input = gr.Textbox(
                        label="Enter Symptoms",
                        placeholder="e.g., fever, headache, cough, fatigue...",
                        lines=4
                    )
                    predict_btn = gr.Button("Analyze Symptoms")

                with gr.Column():
                    prediction_output = gr.Textbox(label="Possible Conditions & Recommendations", lines=20)

            predict_btn.click(disease_prediction, inputs=symptoms_input, outputs=prediction_output)

        with gr.TabItem("Treatment Plans"):
            with gr.Row():
                with gr.Column():
```

```
app.launch(share=True)

/usr/local/lib/python3.12/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),
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(
tokenizer_config.json: 8.88k/? [00:00<00:00, 577kB/s]
vocab.json: 777k/? [00:00<00:00, 11.1MB/s]
merges.txt: 442k/? [00:00<00:00, 21.3MB/s]
tokenizer.json: 3.48M/? [00:00<00:00, 83.3MB/s]
added_tokens.json: 100% 87.0/87.0 [00:00<00:00, 9.28kB/s]
special_tokens_map.json: 100% 701/701 [00:00<00:00, 45.6kB/s]
config.json: 100% 786/786 [00:00<00:00, 94.0kB/s]
`torch_dtype` is deprecated! Use `dtype` instead!
model.safetensors.index.json: 29.8k/? [00:00<00:00, 1.81MB/s]
Fetching 2 files: 100% 2/2 [02:33<00:00, 153.03s/it]
Terminal
```