HealthAI: AI-Enhanced Healthcare Assistant

- Team Leader: ANBUKUMARAN.K.A
- Team Members: AVINASH.M.B
- Team Members: BHARANI.B
- Team Members:BHARANIDHARAN.R

1. Introduction:

HealthAI is an AI-powered healthcare assistant that leverages IBM Granite LLMs to enhance the medical support lifecycle. It automates patient interaction, disease prediction, treatment planning, and documentation, providing accessible and secure healthcare guidance.

By integrating generative AI models with Gradio, HealthAI ensures fast, reliable, and user-friendly medical support. The goal is to bridge the gap between technology and healthcare by offering patients, caregivers, and medical practitioners intelligent insights for decision-making.

2. Project Overview:

The purpose of HealthAI is to create a smart healthcare assistant that simplifies interaction with medical knowledge. Users can:

Chat with an AI-powered patient assistant.

Receive early disease predictions based on symptoms.

Explore personalized treatment plans.

Access reliable, easy-to-understand healthcare information.

This project is deployed on Google Colab with GPU acceleration, ensuring scalability and accessibility.

3. Architecture:

Frontend (Gradio):

Interactive web interface with tab-based layout (Patient Chat, Disease Prediction, Treatment Plans).

Lightweight, real-time interactions.

Backend (PyTorch + Transformers):

IBM Granite LLM handles NLP tasks like symptom analysis and treatment suggestion.

Torch enables GPU-powered model inference.

Document & Model Integration:

Hugging Face API integrates Granite models.

Python scripts handle patient data input, analysis, and output generation.

Deployment:

Runs in Google Colab with T4 GPU for cost-effective and scalable deployment.

4. Setup Instructions:

Prerequisites:

Python 3.9+

Gradio Framework → Docs

IBM Granite Models on Hugging Face → Models

Google Colab with T4 GPU

GitHub Account for version control

Installation:

```
!pip install transformers torch gradio -q
```

5. Folder Structure:

(For Colab deployment, logical structure)

```
📂 HealthSDLC
│── 📂 app/            # Core backend logic
│   │── patient_chat.py  # Patient interaction module
│   │── predictor.py     # Disease prediction
│   │── treatment.py     # Treatment planning
│   │── utils.py         # Utility functions
│
│── 📂 ui/              # Gradio interface
│   │── components/      # Layouts and widgets
│   │── pages/           # Chat, prediction, treatment tabs
│
│── granite_llm.py       # IBM Granite integration
│── app.py               # Entry point (Gradio App)
│── requirements.txt     # Dependencies
│── README.md            # Documentation
│── .gitignore
```

6. Running the Application:

1. Open Google Colab → New Notebook.

2. Change Runtime → T4 GPU.

3. Install dependencies:

```
!pip install transformers torch gradio -q
```

4. Run app.py.

5. Access Gradio interface for real-time interaction.

7. Module Documentation:

Patient Chat:

Conversational assistant for healthcare queries.

Provides AI-generated medical explanations.

Disease Prediction:

Input symptoms → AI model suggests likely conditions.

Outputs probability scores for diseases.

Treatment Plan Generator:

Suggests general treatment guidelines and lifestyle adjustments.

Encourages consultation with medical professionals.

8. Authentication:

Currently runs in an open demo environment.
For production:

Secure API key integration.

Patient identity management (OAuth 2.0, JWT).

HIPAA-compliant data handling.

9. User Interface:

Main Header: "🩺 Health AI: Intelligent Healthcare Assistant"

Tabbed Layout:

Chat Tab → General patient queries.

Disease Prediction Tab → Symptom input and diagnosis output.

Treatment Plan Tab → AI-suggested treatment guidance.

10. Testing:

1. Model Loading Test – Verify IBM Granite initializes without errors.

2. Chat Interaction Test – Input general health query, validate clarity.

3. Disease Prediction Test – Enter sample symptoms, check predictions.

4. Treatment Plan Test – Generate plan, confirm relevance.

5. UI Test – Ensure all Gradio tabs load properly.

6. Edge Cases – Empty input, invalid characters, excessive text.

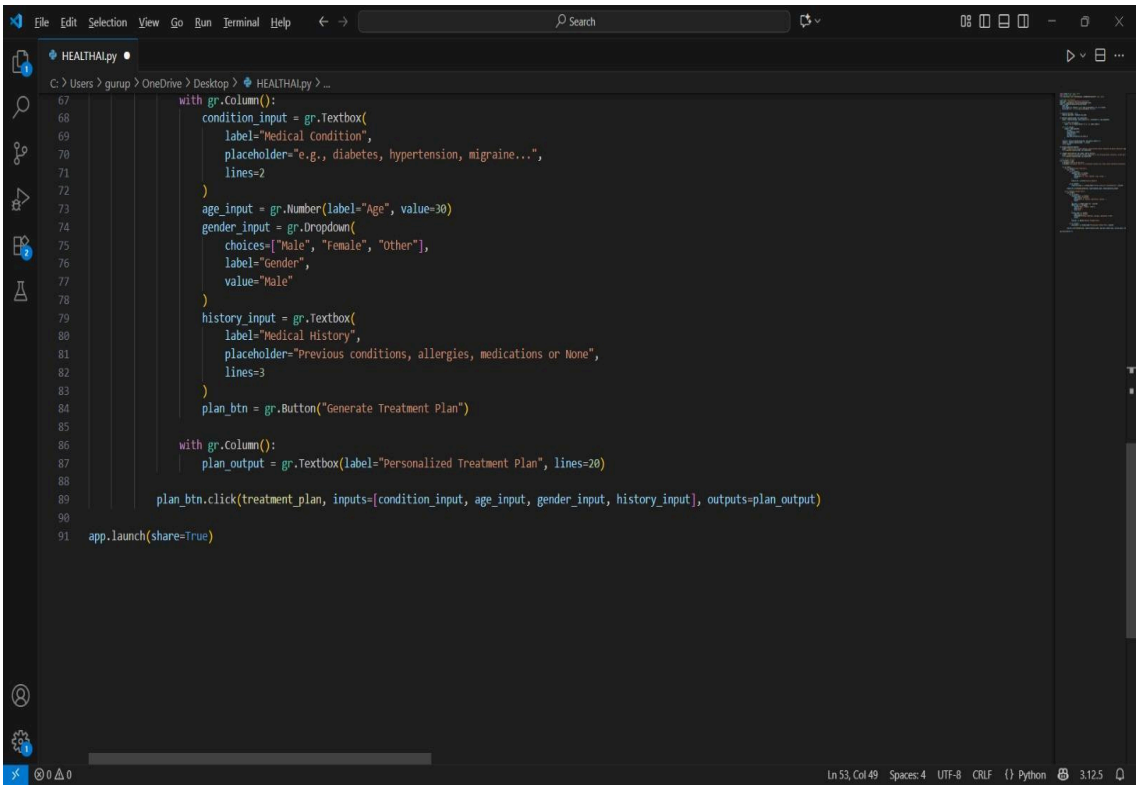
11. Screenshots & Outputs:

(To be inserted: Gradio interface, sample chat, prediction results, treatment suggestions.)

Program:

```python
(function) def disease_prediction(symptoms: Any) -> Any
def disease_prediction(symptoms):
    prompt = f"Based on the following symptoms, provide possible medical conditions and general medication suggestions. Always emphasize the importance of consulting a do
    return generate_response(prompt, max_length=1200)

def treatment_plan(condition, age, gender, medical_history):
    prompt = f"Generate personalized treatment suggestions for the following patient information. Include home remedies and general medication guidelines.\n\nMedical Cond
    return generate_response(prompt, max_length=1200)

# 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():
                    condition_input = gr.Textbox(
                        label="Medical Condition",
                        placeholder="e.g., diabetes, hypertension, migraine...",
                        lines=2
```

```python
import gradio as gr # type: ignore
import torch # type: ignore
from transformers import AutoTokenizer, AutoModelForCausalLM # type: ignore

# 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()}

    with torch.no_grad():
        outputs = model.generate(
            **inputs,
            max_length=max_length,
            temperature=0.7,
            do_sample=True,
            pad_token_id=tokenizer.eos_token_id
        )

    response = tokenizer.decode(outputs[0], skip_special_tokens=True)
    response = response.replace(prompt, "").strip()
    return response

def disease_prediction(symptoms):
    prompt = f"Based on the following symptoms, provide possible medical conditions and general medication suggestions. Always emphasize the importance of consulting a do
```



```python
            with gr.Column():
                condition_input = gr.Textbox(
                    label="Medical Condition",
                    placeholder="e.g., diabetes, hypertension, migraine...",
                    lines=2
                )
                age_input = gr.Number(label="Age", value=30)
                gender_input = gr.Dropdown(
                    choices=["Male", "Female", "Other"],
                    label="Gender",
                    value="Male"
                )
                history_input = gr.Textbox(
                    label="Medical History",
                    placeholder="Previous conditions, allergies, medications or None",
                    lines=3
                )
                plan_btn = gr.Button("Generate Treatment Plan")

            with gr.Column():
                plan_output = gr.Textbox(label="Personalized Treatment Plan", lines=20)

        plan_btn.click(treatment_plan, inputs=[condition_input, age_input, gender_input, history_input], outputs=plan_output)

app.launch(share=True)
```

HUGGING FACE LINK:

https://huggingface.co/spaces/avinxsh77/Health_AI

GITHUB LINK:

https://github.com/AVINASH-MB/HEALTH_AI.git