HealthAI: Intelligent Healthcare Assistant Using IBM Granite

Project Documentation

1.Introduction

• Project title : HealthAI: Intelligent Healthcare Assistant Using IBM Granite

• Team member : MONICA S
• Team member : NANGHINI B
• Team member : ISHWARYA M
• Team member :ARPUTHAVALLI P

2.project overview

• Purpose :

HealthAI harnesses **IBM Watson Machine Learning** and **Generative AI** to provide intelligent healthcare assistance. It analyzes patient data to support clinical decision-making and offers personalized guidance tailored to individual health records. By generating insights from large medical datasets, HealthAI helps identify risks, trends, and potential outcomes. It also automates administrative tasks like documentation, reporting, and appointment scheduling. With these capabilities, HealthAI enhances both the efficiency and accuracy of healthcare delivery, acting as a smart assistant for medical professionals and patients alike.

- **Patient Chat:**

Offers an AI-driven conversational interface that understands natural language, enabling patients to ask complex health questions, receive clarifications, and access real-time guidance tailored to their medical history.

- **Disease Prediction:**

Utilizes advanced machine learning algorithms on historical and real-time patient data to identify early signs of diseases, assess risk factors, and provide probabilistic forecasts to aid preventive care.

- **Treatment Plans:**

Generates personalized care pathways by integrating clinical guidelines, patient-specific data, and predictive analytics, helping healthcare providers design optimized treatment strategies while supporting shared decision-making with patients.

- **Health Analytics:** Aggregates and analyzes vast volumes of structured and unstructured medical data, uncovering patterns, trends, and anomalies to support research, improve operational efficiency, and enhance patient outcomes.

3.Architecture

Frontend (Streamlit): HealthAI's frontend is built using Streamlit, providing a simple and interactive user interface. A sidebar enables easy navigation between modules like Patient Chat, Disease Prediction, and Analytics. Interactive charts with Plotly allow users to explore data visually, while a responsive design ensures the dashboard works smoothly across desktops, tablets, and mobile devices.

Sidebar & Navigation: A side menu lets users switch between different pages (like Patient Chat or Analytics) without cluttering the main screen.

Interactive Charts with Plotly: Graphs that users can hover over, zoom, and explore to understand trends and data better.

Responsive Design: Layout adjusts to different screen sizes, so it works well on computers, tablets, and phones.

This keeps the dashboard **easy to use, interactive, and accessible**.

4.Setup Instructions

Prerequisites:

- Python 3.9 or later
- pip and virtual environment tool
- API keys for IBM Watsonx and Pinecon
- Internet access

Installation Steps:

- Clone the repositor
- Install dependencies from `requirements.txt`
- Create a `.env` file and configure credentials
- Run the backend server using FastAPI
- Launch the frontend via Streamlit
- Upload data and interact with the modules

5.Folder Structure:

- **app/** – Backend logic and API routes for chat, predictions, and analytics.
- **ui/** – Frontend components for Streamlit pages and forms.
- **smart_dashboard.py** – Launches the main dashboard.
- **granite_llm.py** – Handles AI communication.
- **document_embedder.py, kpi_file_forecaster.py, anomaly_file_checker.py, report_generator.py** – Manage data processing, forecasting, anomaly detection, and report generation.

Activities:

- Main logic in `app.py`
- Functions for each feature
- Prompting strategies for IBM Granite AI

6. Running the Application

- Launch the FastAPI server to expose backend endpoints.
- Run the Streamlit dashboard to access the web interface.
- Navigate through pages via the **sidebar**.
- Upload documents or CSVs, interact with the **chat assistant**, and view outputs like **reports, summaries, and predictions**.
- All interactions are **real-time**, with the frontend dynamically updated via backend APIs.

Frontend (Streamlit):

Interactive web UI with dashboards, file uploads, chat, feedback forms, and report viewers.**Sidebar navigation** using the `streamlit-option-menu` library.Modular page design for **scalability**.

Backend (FastAPI):

REST API framework handling document processing, chat, report generation, and vector embedding.Optimized for **asynchronous performance** and supports **Swagger** for API documentation

7. API Documentation

The backend exposes the following APIs:

- **POST /chat/ask** – Accepts a user query and returns an AI-generated response.
- **POST /upload-doc** – Uploads documents and stores embeddings in Pinecone.
- **GET /search-docs** – Retrieves semantically similar policies for a given query.
- **GET /get-eco-tips** – Provides sustainability tips on energy, water, or waste.
- **POST /submit-feedback** – Stores citizen feedback for review and analytics.

All endpoints are **tested and documented in Swagger UI**, allowing quick inspection and trial during development.

8.Authentication

All endpoints are tested and documented in Swagger UI for easy inspection and trial.**The current version runs in an open environment for demonstration purposes.**

- For secure deployments, the system can integrate:
- **Token-based authentication** (JWT or API keys)**OAuth2** with IBM Cloud credentials
- **Role-based access** (admin, citizen, researcher)

Planned enhancements include **user sessions and history tracking** to improve security and personalization.

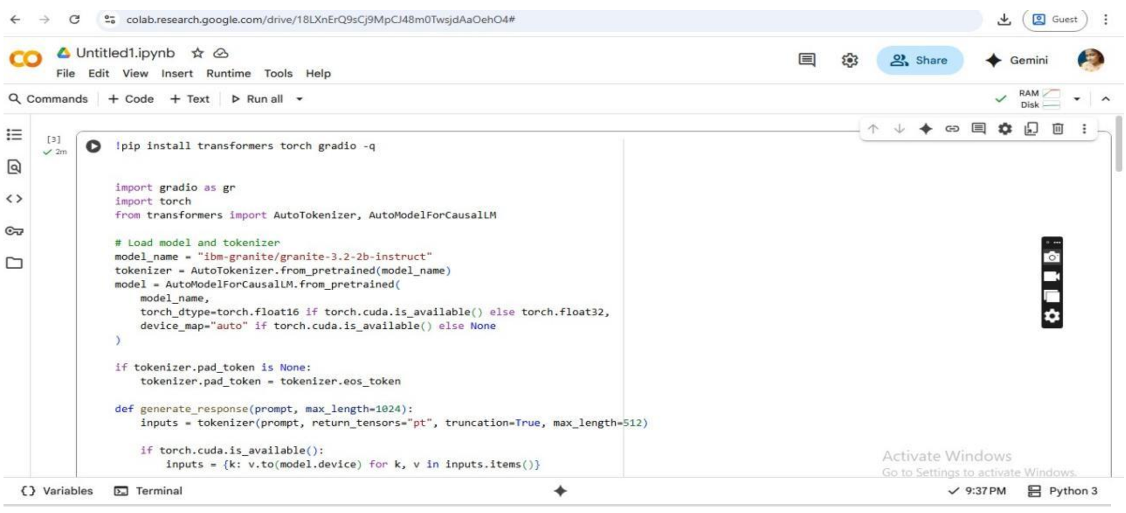
9. User Interface

- Minimalist and functional design for **non-technical users**.
- **Sidebar navigation** for easy access to modules.
- **KPI visualizations** with summary cards.
- **Tabbed layouts** for chat, eco tips, and forecasting.
- **Real-time form handling** and **PDF report downloads**.
- Design focuses on **clarity, speed, and intuitive guidance** with help texts.

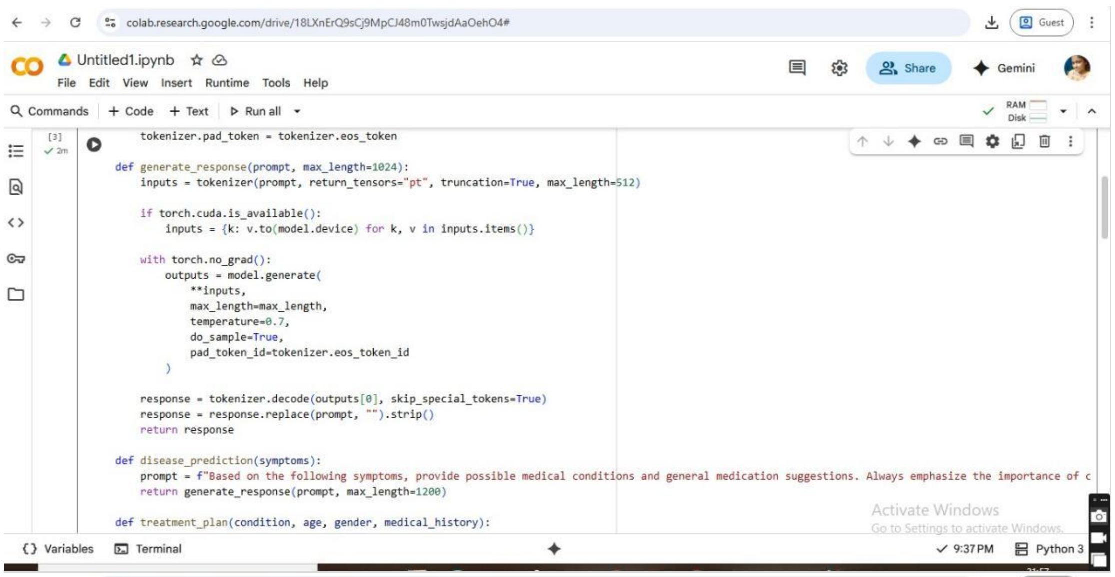
10. Testing

- **Unit Testing:** For prompt engineering functions and utility scripts.
- **API Testing:** Via Swagger UI, Postman, and test scripts.
- **Manual Testing:** For file uploads, chat responses, and output consistency.
- **Edge Case Handling:** Malformed inputs, large files, invalid API keys.
- Ensures reliability in both **offline and API-connected modes**.\

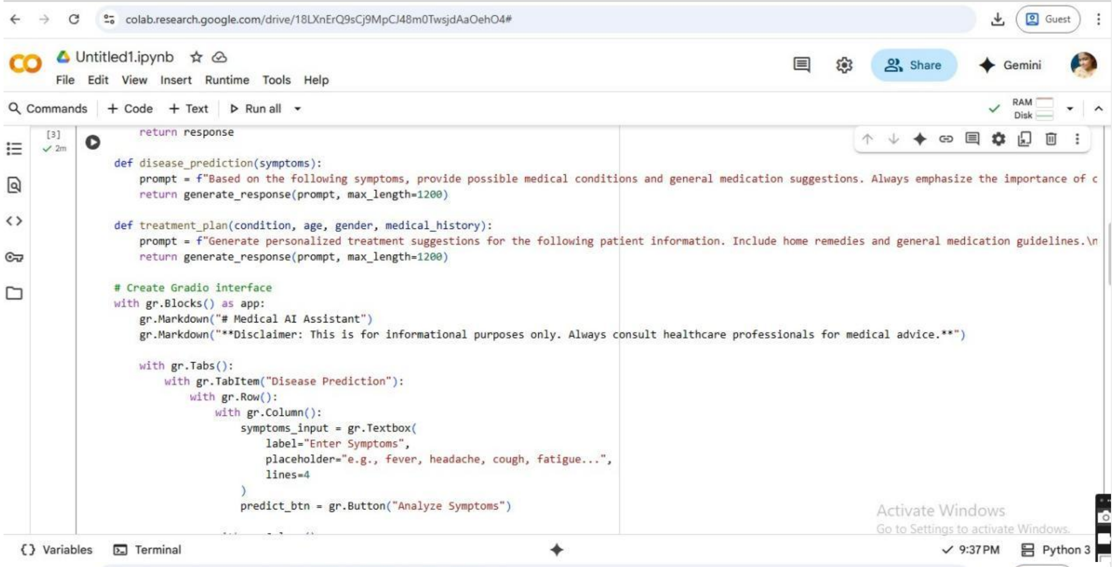
11.Screenshort


colab.research.google.com/drive/18LXnErQ9sCj9MpCJ48m0TwsjdAaOehO4#
Guest
Untitled1.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands
+ Code
+ Text
Run all
RAM
Disk
[3]
2m
Activate Windows
Go to Settings to activate Windows.
Variables
Terminal
9:37 PM
Python 3

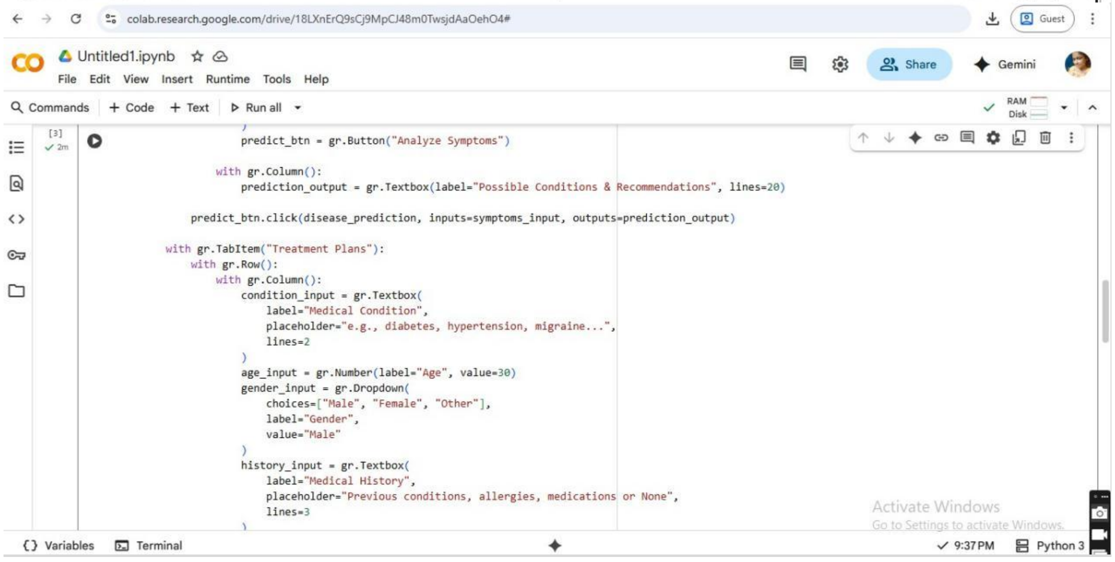
```
!pip install transformers torch gradio -q


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()}
```


colab.research.google.com/drive/18LXnErQ9sCj9MpCJ48m0TwsjdAaOehO4#
Untitled1.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Variables
Terminal
9:37 PM
Python 3
Activate Windows
Go to Settings to activate Windows.

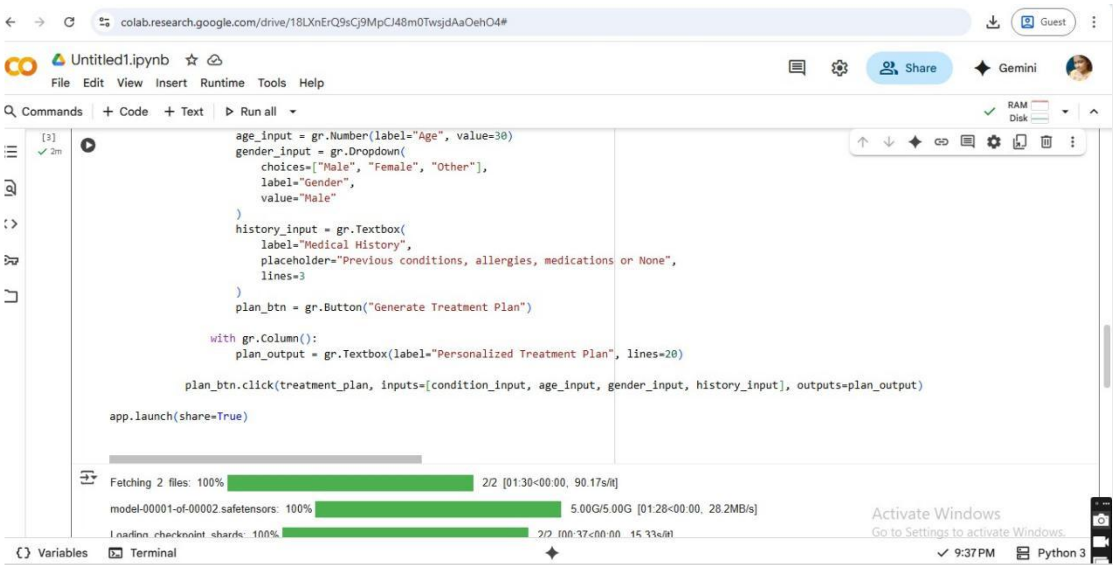
```
    tokenizer.pad_token = tokenizer.eos_token

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 c
    return generate_response(prompt, max_length=1200)

def treatment_plan(condition, age, gender, medical_history):
```


colab.research.google.com/drive/18LXnErQ9sCj9MpCJ48m0TwsjdAaOehO4#
Untitled1.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Variables
Terminal
9:37 PM
Python 3
Activate Windows
Go to Settings to activate Windows.

```
    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 c
    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
    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")
```


colab.research.google.com/drive/18LXnErQ9sCj9MpCJ48m0TwsjdAaOehO4#
Untitled1.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Variables
Terminal
9:37 PM
Python 3
Activate Windows
Go to Settings to activate Windows.

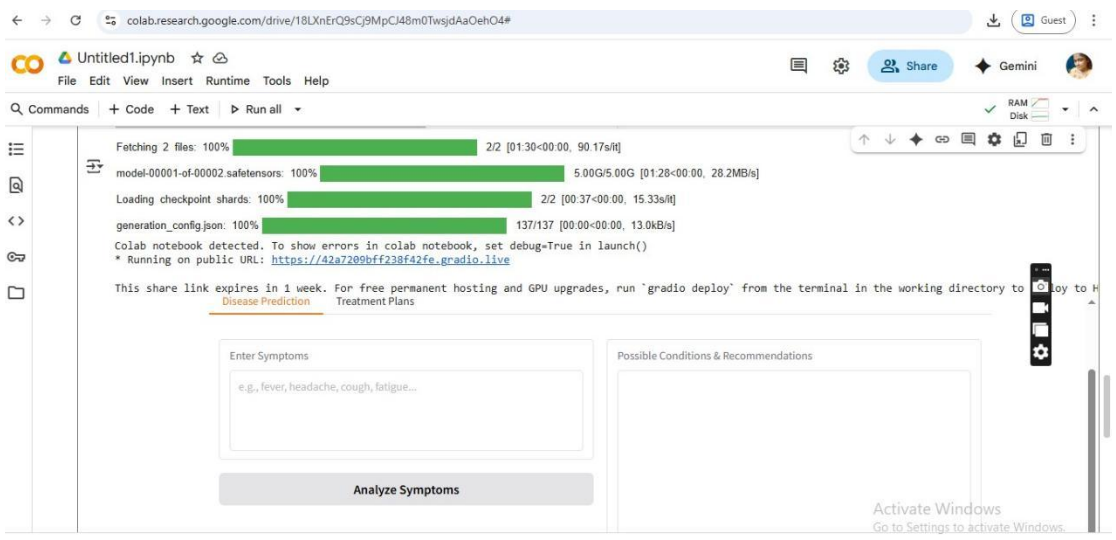
```
                    predict_btn = gr.Button("Analyze Symptoms")

                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
                    )
                    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
                    )
```



```
                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)
```

```
Fetching 2 files: 100% 2/2 [01:30<00:00, 90.17s/it]
model-00001-of-00002.safetensors: 100% 5.00G/5.00G [01:28<00:00, 28.2MB/s]
Loading checkpoint shards: 100% 2/2 [00:37<00:00, 15.33s/it]
```

Untitled1.ipynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share
Gemini
Variables
Terminal
9:37 PM
Python 3



```
Fetching 2 files: 100% 2/2 [01:30<00:00, 90.17s/it]
model-00001-of-00002.safetensors: 100% 5.00G/5.00G [01:28<00:00, 28.2MB/s]
Loading checkpoint shards: 100% 2/2 [00:37<00:00, 15.33s/it]
generation_config.json: 100% 137/137 [00:00<00:00, 13.0kB/s]
Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://42a7209bff238f42fe.gradio.live

This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the working directory to  loy to H
```

12. Known Issues

- Some features may have **limited performance** with very large datasets.
- **Real-time response delays** can occur under high load.
- Minor **UI glitches** on smaller screens or specific browsers.
- **API rate limits** may affect frequent requests in demo mode.

13. Future Enhancements

- Implement **advanced user authentication** with sessions and roles.
- Add **historical data tracking** and analytics dashboards.
- Optimize **performance and scalability** for large datasets.
- Introduce **additional AI features** like predictive health insights and personalized recommendations.