

Health AI:

Intelligent Healthcare Assistant

Project Documentation

1.Introduction

- Project title : Health AI: Intelligent Healthcare Assistant
- Team member : SRIKA.S
- Team member : PAVITHRA.P
- Team member : AARTHI.R
- Team member: KOMALA.V
- Team member : RUDHRA.V

2.project overview

- The **IBM Naan Muthalvan** project is a collaboration that focuses on developing an intelligent healthcare assistant. The primary goal is to improve access to quality healthcare services using AI-driven tools, which can assist both healthcare professionals and patients. Below are the main components of this project: **Purpose:**

The purpose of HealthAI is to provide patients, doctors, and healthcare institutions with an AI-powered assistant for improved healthcare delivery. It leverages IBM Watsonx AI, real-time patient data, and predictive analytics to support diagnosis, treatment, and monitoring

The purpose of this initiative aligns with IBM's commitment to innovation in healthcare through technology, focusing on improving outcomes, increasing efficiency, and providing accessible healthcare to underserved populations.

Objective and Vision:

.Objective: To create an AI-powered healthcare assistant capable of providing diagnostic support, patient management, and healthcare advice.

- **Vision:** To empower healthcare providers with technology that will enable them to deliver more accurate, timely, and cost-effective healthcare solutions, especially in rural areas.

Features:

- **Conversational Healthcare Assistant** - Patients and doctors can interact using natural language.
- **Medical Record Summarization** - Summarizes lengthy medical history into concise insights.
- **Health Forecasting** - Predicts trends in vitals and possible health risks using ML.
- **Anomaly Detection** - Detects abnormal patterns in health data for early alerts.
- **Wellness Recommendations** - Provides personalized lifestyle and wellness tips.
- **Doctor-Patient Feedback Loop** - Collects patient feedback for improved healthcare services.
- **Multimodal Input** - Supports medical PDFs, images, and lab reports.
- **User-Friendly Dashboard** - Provides intuitive dashboards for patients and doctors.

Key Features of the Conversational Interface

- **Text and Voice Interaction:**
 - Users can interact with the system using both text-based chat (inapp or web-based chat interface) and voice-based communication (via smart speakers, mobile apps, or voice assistants).
 - NLP ensures that the assistant understands a wide range of queries, from simple questions to more complex medical inquiries.
- **Personalized Experience:**
 - The interface is capable of tailoring conversations based on the user's previous interactions, medical history, and preferences. For example, the assistant might address the user by name or adjust its tone depending on the user's health concerns (e.g., a more empathetic tone during critical situations).
- **Real-time Symptom Check:**
 - Patients can describe their symptoms, and the conversational interface will analyze the input using AI-driven algorithms. Based on the information, it will suggest potential conditions or encourage users to visit a healthcare professional if necessary.
- **Interactive Guidance for Healthcare Tasks:**
 - **Appointment Scheduling:** The assistant can help schedule medical appointments, send reminders, or connect the user to available doctors and healthcare providers.
 - **Medication Reminders:** It can

- issue reminders for medication timings, provide information on dosage, and track compliance.
 - **Post-Care Support:** After a visit or procedure, the assistant provides follow-up instructions, exercises, or recovery tips, offering ongoing support.
- **Multilingual Support:**
 - To cater to diverse populations, the conversational interface will support multiple languages, making healthcare accessible to a wider audience. Users can switch between languages with ease.
- **Emotional Intelligence:**
 - The assistant is designed to understand and respond empathetically to user emotions. For instance, if a user is anxious or stressed, the assistant will recognize these emotional cues and respond with a reassuring tone.

3. Architecture

- **Frontend:** Streamlit/Gradio (chat UI, dashboards, reports)
- **Backend:** FastAPI (handles chat, patient data, reminders, ML models)
- **LLM Integration:** IBM Watsonx Granite (for medical Q&A, summaries)
- **Database:** MongoDB / SQLite for patient data
- **ML Modules:** Symptom prediction, anomaly detection (using scikitlearn)

1. Frontend (Streamlit)

- Interactive dashboards
- Chatbot interface
- Health record upload & reports

2. Backend (FastAPI)

- API handling & data processing
- ML model integration
- Secure communication with frontend & services

3. LLM (IBM Watsonx Granite)

- Medical record summarization
- Natural language queries
- Conversational AI (chatbot)

4. Database & Vector Search (Pinecone)

- Store patient records as embeddings
- Semantic search for similar cases

5. ML Models

- Disease prediction

- Anomaly detection
- Health trend forecasting

4. Setup Instructions

- Python 3.9+, IBM Watsonx API key, FastAPI, Streamlit
- Steps to clone repo, install dependencies, configure `.env`, run servers

5. Folder Structure

```

healthcare_ibm_project/
|
|├── app/                # FastAPI backend logic
| |├── api/              # Modular API routes
| | |├── chat.py         # Handles chatbot queries
| | |├── records.py      # Upload and manage medical records
| | |├── appointment.py  # Appointment scheduling API
| | |├── feedback.py     # Patient feedback and analytics
| | |└── auth.py         # Authentication and role management
| |
| |├── models/           # Data models (patient, doctor, records)
| |├── services/         # Core backend services
| |└── utils/            # Utility scripts, validators
| |
|├── ml_models/          # Machine learning modules
| |├── symptom_predictor.py # Predicts disease based on symptoms
| |├── anomaly_detector.py  # Detects irregularities in vitals
| |└── report_summarizer.py # Summarizes uploaded medical reports
| |
|├── ui/                 # Streamlit/Gradio frontend
| |├── dashboard.py       # Patient/doctor dashboard
| |├── chatbot_ui.py      # Conversational interface
| |├── appointment_ui.py  # Appointment scheduler
| |└── reports_ui.py      # Report viewer & summarizer
| |
|├── static/             # Static files (icons, images, css)
| |
|├── tests/              # Unit & API test cases
| |├── test_chat.py
| |├── test_records.py
| |└── test_auth.py
| |
|├── .env                # API keys and environment variables
|├── requirements.txt     # Dependencies
|├── README.md            # Project documentation
|└── run.py               # Entry script to launch app

```

(Similar to the sample: app/, api/, ui/, ml_models/, report_generator.py)

6. Running the Application

- Start backend server
- Run Streamlit frontend

- Login/Register as patient/doctor
- Use chatbot, upload reports, get predictions

7. API Documentation

- POST /chat/symptom-check
- POST /upload-records
- GET /appointments
- GET /get-health-tips □ POST /feedback

8. Authentication

- Secure login (JWT, IBM Cloud OAuth)
- Role-based (patient, doctor, admin)

9. User Interface

- Chatbot window
- Dashboard with patient vitals & charts
- Appointment scheduler
- Medical report summarizer
- PDF download of reports

10. Testing

- Unit testing (symptom checker, ML models)
- API testing (Swagger/Postman) Manual testing with patient records

✓ 2m

```
app.launch(share=True)
```

 $\langle \rangle$

vocab.json 777k? [00:00<00:00, 7.55MB/s]

merges.txt 442k? [00:00<00:00, 11.3MB/s]

```
tokenizer.json: 3.48MiB [00:00<00:00, 43.0MiB/s]
```

added_tokens.json: 100% 87.0/87.0 [00:00<00:00, 2.35kB/s]

```
special_tokens_map.json: 100% 701/701 [00:00<00:00, 11.4kB/s]
```

```
config.json: 100% ██████████ 786/786 [00:00<00:00, 26.6kB/s]
```

```

`torch_dtype` is deprecated! Use `dtype` instead!

```

```
model safetensors index.json: 29.8k? [00:00<00:00, 2.88MB/s]
```

Fetching 2 files: 100% 2/2 [01:06<00:00, 66.85s/it]

model-00001-of-00002 safetensors: 100% 5.00G/5.00G | 01:06<00:00, 136

model-00002-of-00002 safetensors: 100% 67.1M/67.1M [00.02<00.00, 30.6]

Loading checkpoint shards: 100% 2/2 [00:20<00:00, 8.67s/it]

```
generation_config.json: 100%  137/137 [00:00<00:00, 4.71kB/s]
```

Colab notebook detected. To show errors in colab notebook, set debug=True in launch()

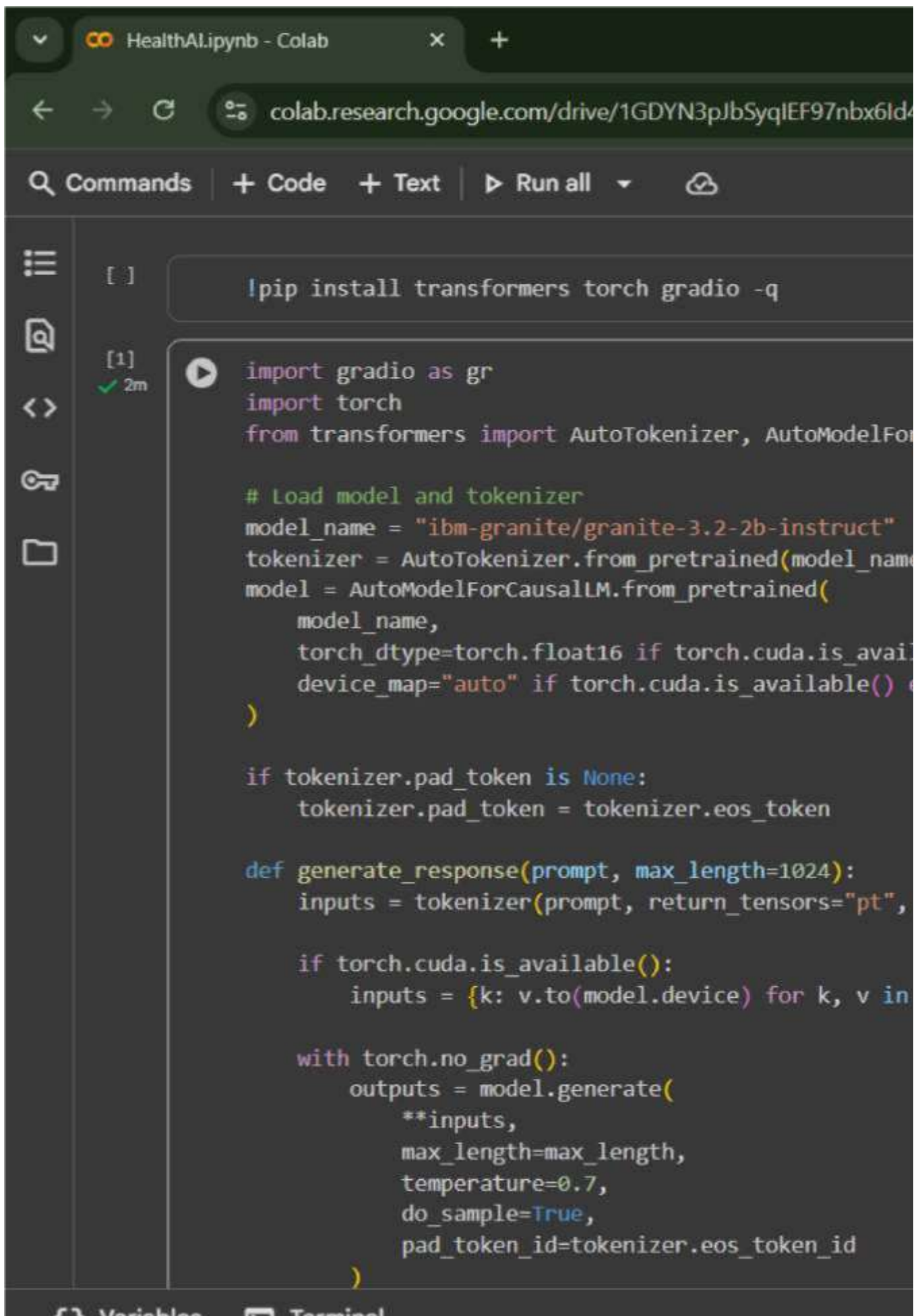
* Running on public URL: <https://e7b20a2d3c7bf0b011.gradio.live>

This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy`

Medical AI Assistant

Disclaimer: This is for informational purposes only. Always consult healthcare professionals for medical advice.

11.Screen shots



The screenshot shows a Google Colab notebook interface. The browser tab is titled "HealthAI.ipynb - Colab". The address bar shows the URL "colab.research.google.com/drive/1GDYN3pJbSyqlEF97nbx6ld4". The notebook has a dark theme. The left sidebar contains icons for a menu, search, expand/collapse, key, and file explorer. The top toolbar includes "Commands", "+ Code", "+ Text", "Run all", and a cloud icon. The notebook content area shows two code cells. The first cell is empty. The second cell, labeled "[1]" with a green checkmark and "2m", contains the following Python code:

```
!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()}

    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
        )
```

At the bottom of the interface, there are tabs for "Variables" and "Terminal".



Medical AI Assistant

Disclaimer: This is for informational purposes only. Always consult healthcare profes

Disease Prediction

Treatment Plans

Enter Symptoms

Treatment plans

Analyze Symptoms



Medical AI Assistant

Disclaimer: This is for informational purposes only. Always consult healthcare profes

Disease Prediction

Treatment Plans

Medical Condition

This blank should be filled with any relevant medical conditions the user might have, such as "e.g., diabetes, hypertension, migraine...". For instance, it could be filled with "Asthma and Hypertension".

Age

30

Gender

Male

Medical History

This blank should be filled with details regarding "Previous conditions, allergies, medications, etc.". For example, it could be filled with "Seasonal allergies to pollen, currently taking anti-inflammatory medication for arthritis."

Generate Treatment Plan

12. Known Issues

- Limited dataset scope □ Requires more diverse training data

13. Future Enhancements

- Integration with IoT devices (wearables)
- Voice-based interaction
- Multilingual support

