Project Title

Health AI: Intelligent Healthcare Assistant

Project Documentation

1. Introduction

- **Project Title**: Health AI: Intelligent Healthcare Assistant
- **Team Members**:
    - Sumithra
    - Sowmiya
    - Srivardhini
    - Sriharini

2. Project Overview

Purpose:
The purpose of Health AI is to provide intelligent healthcare support to patients and doctors by leveraging AI-powered conversational assistance, health data analysis, and personalized medical insights. The system aims to improve accessibility, reduce workload for healthcare professionals, and empower patients to better manage their health.

Features:

- **Conversational Interface**
    - Natural language interaction with patients and doctors for medical queries.
- **Symptom Checker**
    - Provides possible conditions based on symptoms and suggests next steps.
- **Medical Report Summarization**
    - Converts lengthy medical reports into concise, patient-friendly summaries.
- **Medication & Appointment Reminders**
    - Notifies patients of prescribed medicine schedules and upcoming appointments.

- **Health Risk Prediction**
    - Uses AI models to forecast potential risks like diabetes, heart disease, etc.
- **Doctor Recommendation**
    - Suggests specialists based on patient symptoms and location.
- **Feedback Loop**
    - Allows patients to give feedback to improve the system.
- **Multimodal Input Support**
    - Accepts text, images (like prescriptions, lab reports), and PDFs for analysis.
- **User-Friendly Dashboard**
    - Provides intuitive access to health summaries, reminders, and recommendations.

3. Architecture

- **Frontend (Streamlit/Gradio):**
  Interactive UI for patients and doctors, including chat interface, report upload, and reminders.

- **Backend (FastAPI):**
  Handles medical data processing, chat interactions, and report summarization.

- **LLM Integration (OpenAI / IBM Watsonx):**
  Used for natural language understanding, report summarization, and chatbot responses.

- **Database (MongoDB / PostgreSQL):**
  Stores patient data, medical history, and reminders securely.

- **ML Modules:**
    - Symptom-to-condition prediction models
    - Risk forecasting (e.g., diabetes, heart disease)
    - Anomaly detection in medical reports

4. Setup Instructions

Prerequisites:

- Python 3.9 or later
- pip and virtual environment tools
- API keys for LLM and database access
- Internet access

Installation Process:

1. Clone the repository
2. Install dependencies (`requirements.txt`)
3. Configure `.env` with credentials
4. Run backend server with FastAPI
5. Launch frontend (Streamlit/Gradio)
6. Upload reports or chat with the assistant

5. Folder Structure

```
app/                  # FastAPI backend
app/api/              # API routes for chat, reports, reminders
ui/                   # Streamlit/Gradio frontend pages
health_dashboard.py   # Entry script for UI
symptom_checker.py    # AI-based symptom analysis
report_summarizer.py  # Summarizes medical reports
risk_predictor.py     # Predicts chronic disease risks
reminder_system.py    # Medicine & appointment reminders
```

6. Running the Application

- Start FastAPI backend server
- Launch Streamlit/Gradio dashboard
- Use sidebar to navigate (chat, reports, reminders, risk predictions)
- Upload medical reports, ask queries, and receive AI-powered insights

7. API Documentation

- **POST /chat/ask** → Patient queries answered

- **POST /upload-report** → Upload and analyze medical reports
- **GET /symptom-checker** → Provides possible conditions
- **GET /risk-predict** → Predicts potential health risks
- **POST /set-reminder** → Schedule medication or appointment reminders
- **POST /feedback** → Collects patient feedback

8. Authentication

- Token-based authentication (JWT)
- Role-based access (Patient, Doctor, Admin)
- Optional OAuth2 for secure login

9. User Interface

- Sidebar navigation
- Chat with AI assistant
- Upload & summarize reports
- Health dashboard with KPIs (risks, reminders, appointments)
- Downloadable summaries/reports

10. Testing

- Unit Testing (AI models, symptom checker)
- API Testing (Postman/Swagger)
- Manual Testing (chat, reports, reminders)
- Edge Case Handling (invalid symptoms, large reports)

11. Screenshots

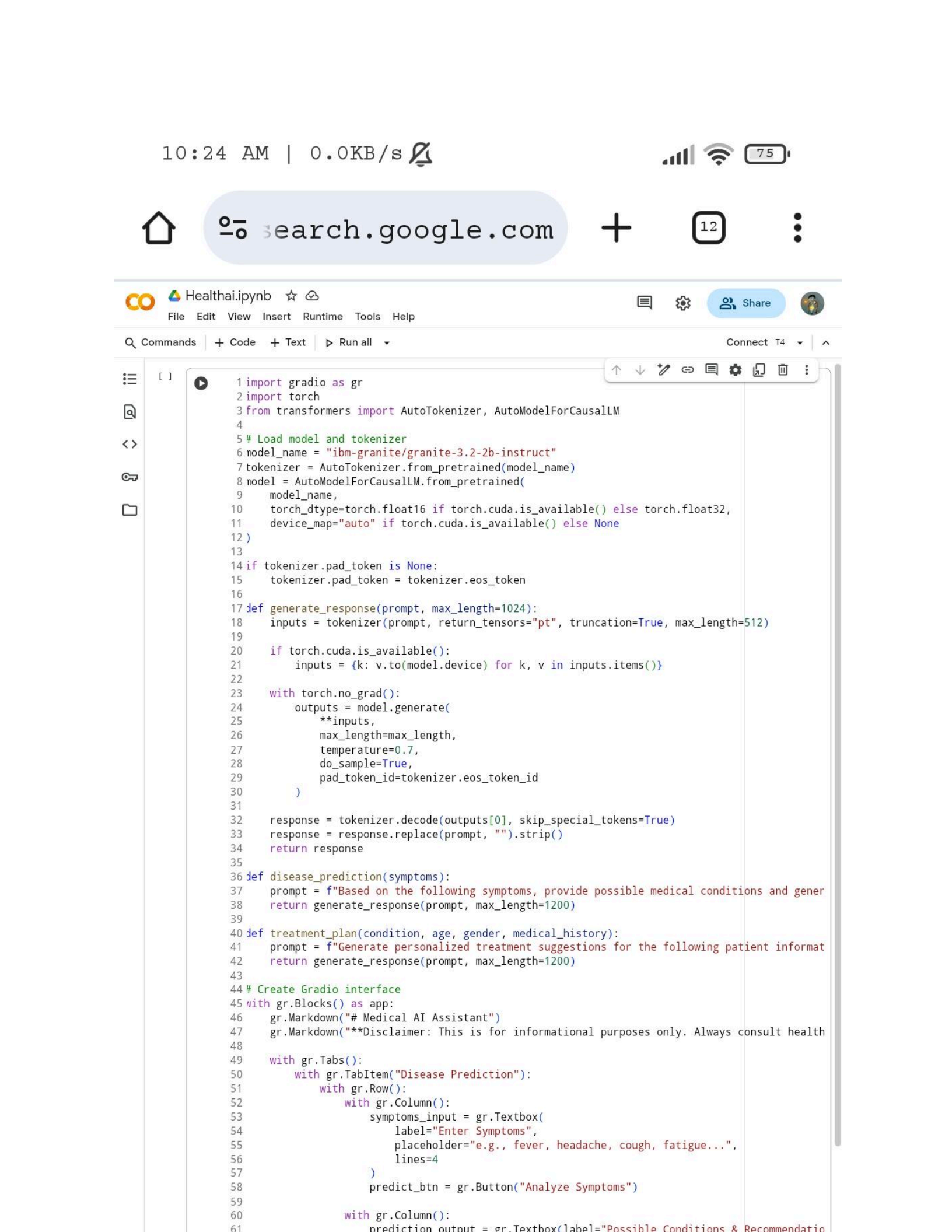
- *To be added once UI is implemented*

12. Known Issues

- Limited accuracy in rare medical conditions
- Dependency on internet for cloud AI services

13. Future Enhancements

- Integration with wearable devices (smartwatch, fitness trackers)
- Multilingual support for regional languages
- Voice-based interaction
- Emergency alert system (e.g., fall detection, abnormal vitals)




```python
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
        )

    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 gener
    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 informat
    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 health

    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 & Recommendatio
```

Medical AI Assistant

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

Disease Prediction | Treatment Plans

Enter Symptoms

e.g., fever, headache, cough, fatigue...

Analyze Symptoms

Possible Conditions & Recommendations