HealthAi: Intelligent Health Care Assistant

Project Title: health AI intelligent Health Care assistant

Team Leader: KOKILA.D

Team member: NITHYA SREE.K

Team member: RUBINI.E

Team member: NATHIYA.D

Team member: PANDESHWARI.N

1. **INTRODUCTION**

---

1. PROJECT OVERVIEW

HealthAi is an intelligent AI-powered health care assistant that uses advanced machine learning algorithms to provide personalized health advice, monitor vital statistics, assist in diagnosing symptoms, and remind users about medication. The system integrates with wearables and medical data sources to offer actionable health insights in real-time.

Key Features:

- **Health Monitoring: Tracks vital signs and connects with wearable devices.**
- **Symptom Analysis: Uses AI to suggest possible health conditions based on symptoms entered by the user.**
- **Medication Reminders: Alerts users to take medication on time.**
- **Emergency Assistance: Provides emergency contact and healthcare service recommendations.**
- **Data Insights: Offers users data-driven insights to track and improve their health.**

This project documentation covers the complete setup, running instructions, and technical architecture required to deploy and operate HealthAi effectively.

---

2. ARCHITECTURE

HealthAi is designed with a modular, scalable architecture to ensure high availability, maintainability, and flexibility. Below is an overview of the system architecture:

2.1. Front-End

- **Mobile App (iOS and Android): Users interact with the mobile app to input data, receive health insights, and communicate with healthcare professionals.**
- **Web Portal for Professionals: A dashboard for doctors and health professionals to monitor patient health, trends, and communicate with patients.**

2.2. Back-End

- **API Layer: Manages user requests, data retrieval, and communication between the front-end and AI models.**
- **AI/ML Engine: Handles health prediction, diagnosis, and personalized advice generation.**
- **Data Storage: Secure databases store user health data, historical medical information, and sensor data.**

2.3. Security Layer

- **Data Encryption: All sensitive health data is encrypted.**
- **Authentication: OAuth2.0 and biometric authentication methods.**
- **Access Control: Role-based access for healthcare professionals and patients.**

2.4. Integration Layer

- **Wearable Device Integration: Interfaces with devices like smartwatches, blood pressure cuffs, and glucose monitors.**
- **Third-party APIs: Integration with health services, medical databases, and emergency services.**

---

3. SET UP INSTRUCTIONS

To get started with the HealthAi project, follow these setup instructions:

3.1. Prerequisites

- **Node.js and npm (for front-end and back-end)**
- **Python (for AI models)**
- **Docker (for containerization)**
- **MongoDB (or PostgreSQL depending on preference)**
- **Cloud Service Account (AWS, Azure, etc.)**

3.2. Step-by-Step Setup

1. **Clone the Repository**
2. **git clone https://github.com/your-repo/HealthAi.git**
3. **cd HealthAi**
4. **Install Backend Dependencies**
   Navigate to the back-end directory and install required dependencies.
5. **cd backend**
6. **npm install**
7. **Set up Database**
   - **Install MongoDB locally or set up a cloud instance.**
   - **Configure your database connection in the config/db.js file.**
8. **Install Front-End Dependencies**
   Navigate to the front-end directory and install dependencies.
9. **cd frontend**
10. **npm install**
11. **Configure API Keys**
    - **Set up third-party API keys for wearable integrations and emergency services.**
    - **Add these keys to the .env file.**
12. **Start the Application**
13. **npm run start**
14. **Launch the Mobile App**
    - **For iOS: Open the HealthAi.xcodeproj in Xcode and run.**
    - **For Android: Open the project in Android Studio and run.**

---

4. FOLDER STRUCTURE

The HealthAi project follows a modular structure for better scalability and maintainability. Here's a breakdown of the folder structure:

HealthAi/

```
|
├── backend/              # Backend code and APIs
```

| ├── controllers/ # API Controllers

| ├── models/ # Database models

| ├── routes/ # API routes

| ├── services/ # Business logic

| └── utils/ # Helper functions

|

├── frontend/ # Frontend code for mobile app

| ├── components/ # Reusable UI components

| ├── screens/ # Different screens in the app

| ├── utils/ # Utility functions and hooks

| └── assets/ # Images and fonts

|

├── docs/ # Documentation files

├── config/ # Configuration files (database, API keys)

├── scripts/ # Automation scripts (build, deploy, etc.)

└── .env # Environment variables (API keys, secrets)

---

5. RUNNING THE APPLICATION

After setting up the environment as described in the Set Up Instructions, you can run the HealthAi application using the following steps:

5.1. Running the Back-End

To run the back-end server:

cd backend

npm start

This will start the API server at http://localhost:3000.

5.2. Running the Front-End (Mobile App)

- **For iOS:**
  Open the HealthAi.xcodeproj file in Xcode and run the app.

- **For Android:**
  Open the project in Android Studio and run the app on an emulator or device.

5.3. Running the Full Application

To run the full system with Docker containers:

docker-compose up

This will launch all necessary services (backend, frontend, database, etc.) in separate containers.

---

6. API DOCUMENTATION

The HealthAi back-end provides a RESTful API for communication between the mobile app, web portal, and external services. Below are the key API endpoints:

6.1. Authentication

- **POST /api/auth/login: Logs in a user.**
- **POST /api/auth/register: Registers a new user.**
- **POST /api/auth/logout: Logs out a user.**

6.2. Health Data

- **GET /api/vitals: Retrieves the latest health vitals (e.g., heart rate, blood pressure).**
- **POST /api/vitals: Submit new health data (e.g., from a wearable).**
- **GET /api/medications: Retrieves a list of medication reminders for the user.**
- **POST /api/medications: Adds a new medication schedule.**

6.3. Symptom Checker

- **POST /api/symptoms: Analyze entered symptoms and return possible diagnoses.**

6.4. Emergency Services

- **GET /api/emergency: Get emergency contact information based on the user's location.**

---

7. USER INTERFACE

The user interface (UI) of HealthAi is designed to be simple, intuitive, and accessible. Key features of the UI include:

7.1. Mobile App UI

- **Dashboard: Displays user health stats (e.g., vitals, exercise, medication).**
- **Symptom Checker: A user-friendly form for entering symptoms.**

- **Health Insights: Graphs and reports visualizing trends in health data.**

7.2. Web Portal UI for Professionals

- **Patient Dashboard: Doctors can view patients' health metrics, trends, and historical data.**
- **Consultation Screen: Virtual consultation options for patient interaction.**
- **Notifications: Alerts for critical health data or patient requests.**

---

8. TESTING

Testing ensures that HealthAi is robust, secure, and performs well under different scenarios. The project includes:

8.1. Unit Testing

- **Back-End: Jest and Mocha are used for testing API routes, business logic, and utility functions.**
- **Front-End: Jest and React Testing Library are used for testing UI components and functionality.**

8.2. Integration Testing

- **Tests cover end-to-end interactions between the back-end and front-end, ensuring that data flows correctly between the user interface and server.**

8.3. Manual Testing

- **Mobile App: Manual testing on various devices and OS versions.**
- **Back-End: API testing using Postman.**

PROJECT DOCUMENTATION

---

1. PROJECT OVERVIEW

HealthAi is an intelligent AI-powered health care assistant that uses advanced machine learning algorithms to provide personalized health advice, monitor vital statistics, assist in diagnosing symptoms, and remind users about medication. The system integrates with wearables and medical data sources to offer actionable health insights in real-time.

Key Features:

- **Health Monitoring**: Tracks vital signs and connects with wearable devices.
- **Symptom Analysis**: Uses AI to suggest possible health conditions based on symptoms entered by the user.
- **Medication Reminders**: Alerts users to take medication on time.
- **Emergency Assistance**: Provides emergency contact and healthcare service recommendations.

- **Data Insights**: Offers users data-driven insights to track and improve their health.

This project documentation covers the complete setup, running instructions, and technical architecture required to deploy and operate **HealthAi** effectively.

---

2. ARCHITECTURE

HealthAi is designed with a modular, scalable architecture to ensure high availability, maintainability, and flexibility. Below is an overview of the system architecture:

2.1. Front-End

- **Mobile App (iOS and Android)**: Users interact with the mobile app to input data, receive health insights, and communicate with healthcare professionals.
- **Web Portal for Professionals**: A dashboard for doctors and health professionals to monitor patient health, trends, and communicate with patients.

2.2. Back-End

- **API Layer**: Manages user requests, data retrieval, and communication between the front-end and AI models.
- **AI/ML Engine**: Handles health prediction, diagnosis, and personalized advice generation.
- **Data Storage**: Secure databases store user health data, historical medical information, and sensor data.

2.3. Security Layer

- **Data Encryption**: All sensitive health data is encrypted.
- **Authentication**: OAuth2.0 and biometric authentication methods.
- **Access Control**: Role-based access for healthcare professionals and patients.

2.4. Integration Layer

- **Wearable Device Integration**: Interfaces with devices like smartwatches, blood pressure cuffs, and glucose monitors.
- **Third-party APIs**: Integration with health services, medical databases, and emergency services.

---

3. SET UP INSTRUCTIONS

To get started with the **HealthAi** project, follow these setup instructions:

3.1. Prerequisites

- **Node.js** and **npm** (for front-end and back-end)

- **Python** (for AI models)
- **Docker** (for containerization)
- **MongoDB** (or PostgreSQL depending on preference)
- **Cloud Service Account** (AWS, Azure, etc.)

3.2. Step-by-Step Setup

1. **Clone the Repository**
2. git clone https://github.com/your-repo/HealthAi.git
3. cd HealthAi
4. **Install Backend Dependencies**
   Navigate to the back-end directory and install required dependencies.
5. cd backend
6. npm install
7. **Set up Database**
    - Install MongoDB locally or set up a cloud instance.
    - Configure your database connection in the config/db.js file.
8. **Install Front-End Dependencies**
   Navigate to the front-end directory and install dependencies.
9. cd frontend
10. npm install
11. **Configure API Keys**
    - Set up third-party API keys for wearable integrations and emergency services.
    - Add these keys to the .env file.
12. **Start the Application**
13. npm run start
14. **Launch the Mobile App**
    - For **iOS**: Open the HealthAi.xcodeproj in Xcode and run.
    - For **Android**: Open the project in Android Studio and run.

---

4. FOLDER STRUCTURE

The **HealthAi** project follows a modular structure for better scalability and maintainability. Here's a breakdown of the folder structure:

```
HealthAi/
|
├── backend/                # Backend code and APIs
|   ├── controllers/        # API Controllers
|   ├── models/             # Database models
|   ├── routes/             # API routes
|   ├── services/           # Business logic
|   └── utils/              # Helper functions
|
├── frontend/               # Frontend code for mobile app
|   ├── components/         # Reusable UI components
|   ├── screens/            # Different screens in the app
|   ├── utils/              # Utility functions and hooks
|   └── assets/             # Images and fonts
|
├── docs/                   # Documentation files
├── config/                 # Configuration files (database, API keys)
├── scripts/                # Automation scripts (build, deploy, etc.)
└── .env                    # Environment variables (API keys, secrets)
```

---

5. RUNNING THE APPLICATION

After setting up the environment as described in the **Set Up Instructions**, you can run the **HealthAi** application using the following steps:

5.1. Running the Back-End

To run the back-end server:

```
cd backend

npm start
```

This will start the API server at http://localhost:3000.

5.2. Running the Front-End (Mobile App)

- For **iOS**:
  Open the HealthAi.xcodeproj file in Xcode and run the app.
- For **Android**:
  Open the project in Android Studio and run the app on an emulator or device.

5.3. Running the Full Application

To run the full system with Docker containers:

docker-compose up

This will launch all necessary services (backend, frontend, database, etc.) in separate containers.

---

6. API DOCUMENTATION

The **HealthAi** back-end provides a RESTful API for communication between the mobile app, web portal, and external services. Below are the key API endpoints:

6.1. Authentication

- **POST** /api/auth/login: Logs in a user.
- **POST** /api/auth/register: Registers a new user.
- **POST** /api/auth/logout: Logs out a user.

6.2. Health Data

- **GET** /api/vitals: Retrieves the latest health vitals (e.g., heart rate, blood pressure).
- **POST** /api/vitals: Submit new health data (e.g., from a wearable).
- **GET** /api/medications: Retrieves a list of medication reminders for the user.
- **POST** /api/medications: Adds a new medication schedule.

6.3. Symptom Checker

- **POST** /api/symptoms: Analyze entered symptoms and return possible diagnoses.

6.4. Emergency Services

- **GET** /api/emergency: Get emergency contact information based on the user's location.

---

7. USER INTERFACE

The user interface (UI) of **HealthAi** is designed to be simple, intuitive, and accessible. Key features of the UI include:

7.1. Mobile App UI

- **Dashboard**: Displays user health stats (e.g., vitals, exercise, medication).
- **Symptom Checker**: A user-friendly form for entering symptoms.
- **Health Insights**: Graphs and reports visualizing trends in health data.

7.2. Web Portal UI for Professionals

- **Patient Dashboard**: Doctors can view patients' health metrics, trends, and historical data.
- **Consultation Screen**: Virtual consultation options for patient interaction.
- **Notifications**: Alerts for critical health data or patient requests.

---

8. TESTING

Testing ensures that **HealthAi** is robust, secure, and performs well under different scenarios. The project includes:

8.1. Unit Testing

- **Back-End**: Jest and Mocha are used for testing API routes, business logic, and utility functions.
- **Front-End**: Jest and React Testing Library are used for testing UI components and functionality.

8.2. Integration Testing

- Tests cover end-to-end interactions between the back-end and front-end, ensuring that data flows correctly between the user interface and server.

8.3. Manual Testing

- **Mobile App**: Manual testing on various devices and OS versions.
- **Back-End**: API testing using Postman.

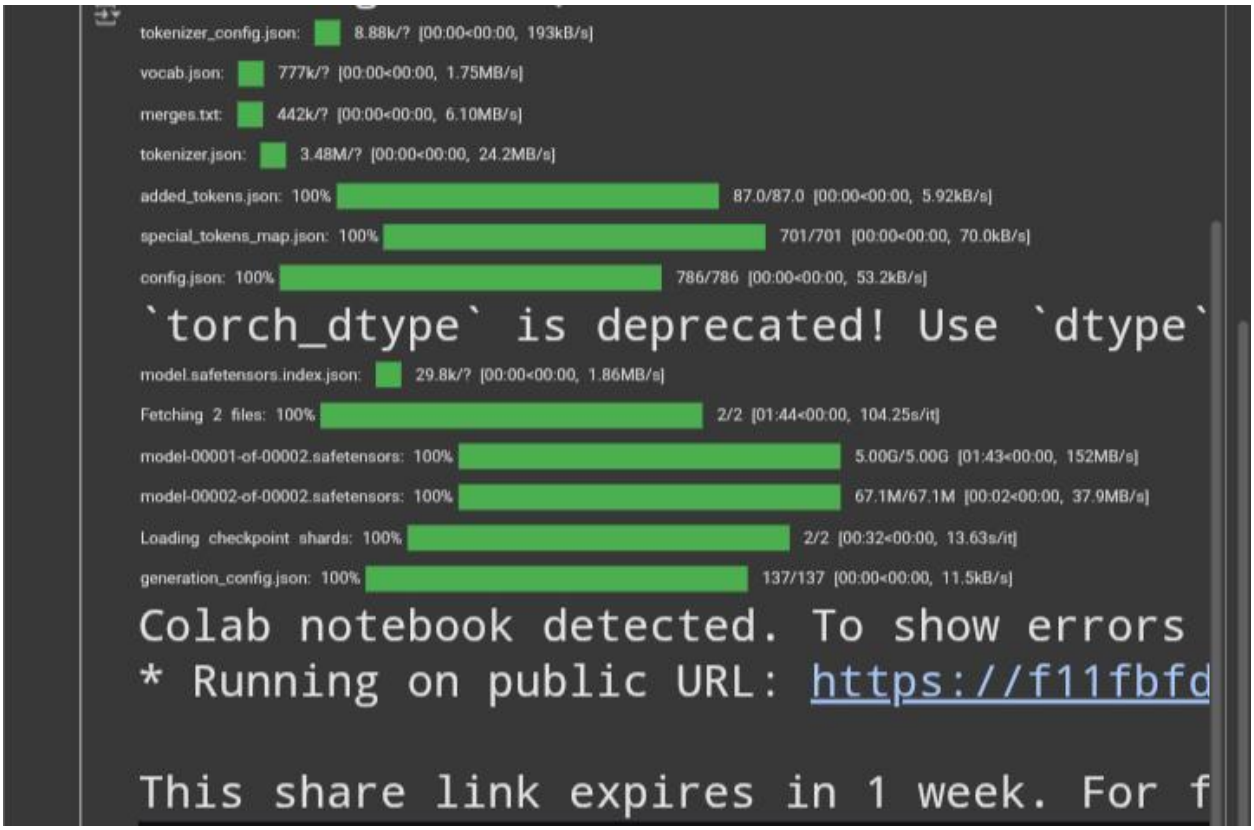
Source code

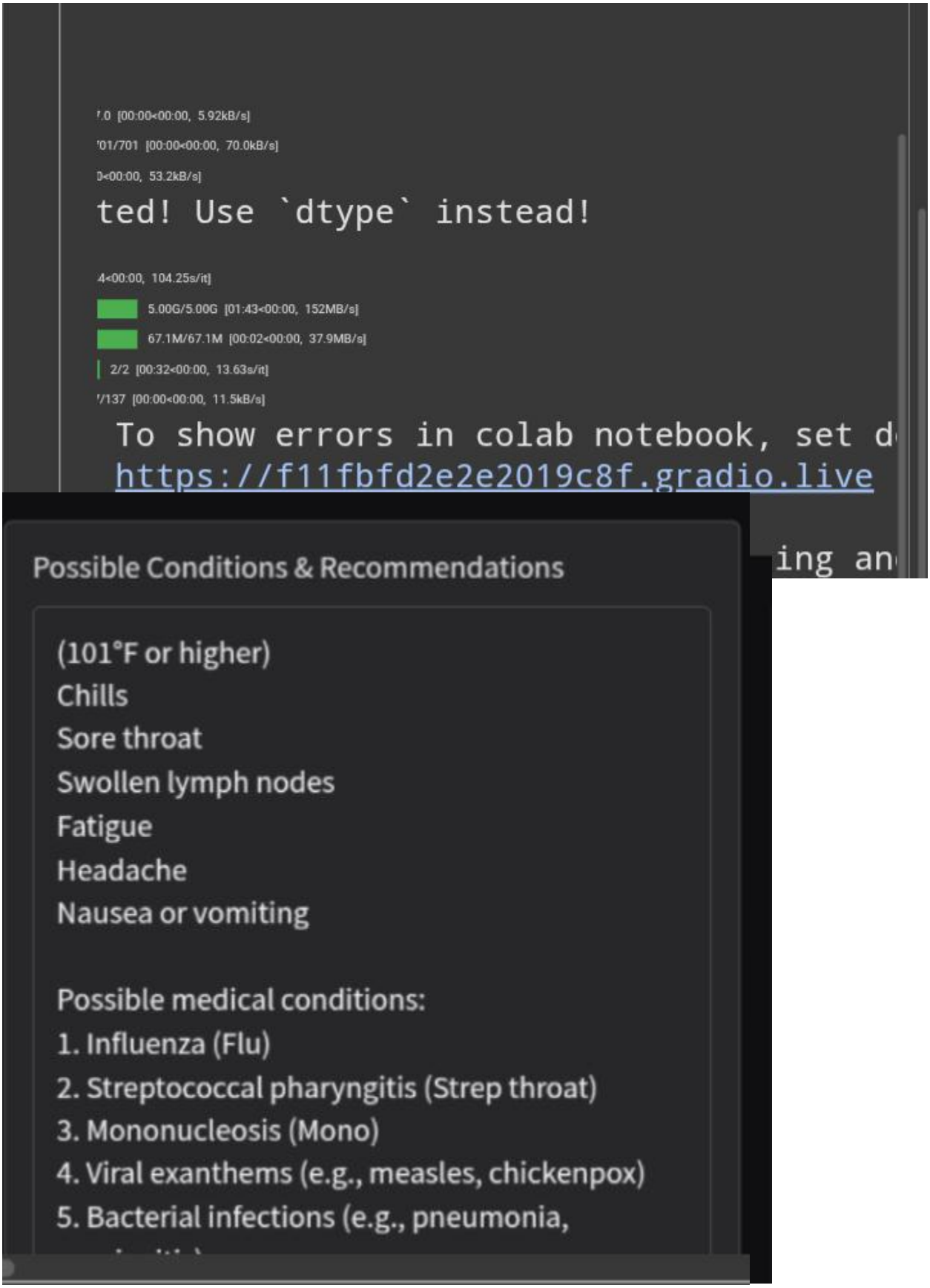
```
import gradio as gr
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM

# 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
)

# Set pad token if not already set
if tokenizer.pad_token is None:
    tokenizer.pad_token = tokenizer.eos_token

# Define response generation function
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(
```

```
    with torch.no_grad():
        outputs = model.generate(
            **inputs,
            max_length=max_length,
            pad_token_id=tokenizer.pad_token_id
        )

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

# Define Gradio interface functions
def disease_prediction(symptoms):
    prompt = f"Given the following symptoms, provide possible medical conditions and general medication suggestions. Always emphasize the importance of consulting a healthcare professional:\n{symptoms}"
    return generate_response(prompt, max_length=1200)

def treatment_plan(condition, age, gender, history):
    prompt = (
        f"Generate personalized treatment suggestions for the following patient information. "
        f"Include home remedies and general medication guidelines.\n"
        f"Condition: {condition}\nAge: {age}\nGender: {gender}\nMedical History: {history}"
    )
    return generate_response(prompt, max_length=1300)

# Build 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.Row():
            with gr.Column():
                symptom_input = gr.Textbox(
                    placeholder="e.g., fever, headache, cough, fatigue...",
                    label="Enter Symptoms",
                    lines=4
                )
                predict_btn = gr.Button("Analyze Symptoms")
            with gr.Column():
                prediction_output = gr.Textbox(
                    label="Possible Conditions & Recommendations",
                    lines=20
                )
        predict_btn.click(fn=disease_prediction,
inputs=symptom_input, outputs=prediction_output)


    with gr.TabItem("Treatment Plan"):
        with gr.Row():
            with gr.Column():
                condition_input = gr.Textbox(label="Medical Condition",
placeholder="e.g., diabetes, hypertension...", lines=2)
                age_input = gr.Number(label="Age", value=30)
                gender_input = gr.Radio(choices=["Male", "Female",
"Other"], label="Gender")
                history_input = gr.Textbox(label="Medical History",
placeholder="Previous conditions, allergies, medications...",
lines=4)
                plan_btn = gr.Button("Generate Treatment Plan")
            with gr.Column():
                plan_output = gr.Textbox(label="Personalized Treatment
Plan", lines=20)
        plan_btn.click(fn=treatment_plan, inputs=[condition_input,
age_input, gender_input, history_input], outputs=plan_output)


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

Output


tokenizer_config.json: 8.88k/? [00:00<00:00, 193kB/s]
vocab.json: 777k/? [00:00<00:00, 1.75MB/s]
merges.txt: 442k/? [00:00<00:00, 6.10MB/s]
tokenizer.json: 3.48M/? [00:00<00:00, 24.2MB/s]
added_tokens.json: 100% 87.0/87.0 [00:00<00:00, 5.92kB/s]
special_tokens_map.json: 100% 701/701 [00:00<00:00, 70.0kB/s]
config.json: 100% 786/786 [00:00<00:00, 53.2kB/s]
`torch_dtype` is deprecated! Use `dtype`
model.safetensors.index.json: 29.8k/? [00:00<00:00, 1.86MB/s]
Fetching 2 files: 100% 2/2 [01:44<00:00, 104.25s/it]
model-00001-of-00002.safetensors: 100% 5.00G/5.00G [01:43<00:00, 152MB/s]
model-00002-of-00002.safetensors: 100% 67.1M/67.1M [00:02<00:00, 37.9MB/s]
Loading checkpoint shards: 100% 2/2 [00:32<00:00, 13.63s/it]
generation_config.json: 100% 137/137 [00:00<00:00, 11.5kB/s]
Colab notebook detected. To show errors
* Running on public URL: https://f11fbfd
This share link expires in 1 week. For f


'.0 [00:00<00:00, 5.92kB/s]
'01/701 [00:00<00:00, 70.0kB/s]
0<00:00, 53.2kB/s]
ted! Use `dtype` instead!
4<00:00, 104.25s/it]
5.00G/5.00G [01:43<00:00, 152MB/s]
67.1M/67.1M [00:02<00:00, 37.9MB/s]
2/2 [00:32<00:00, 13.63s/it]
'/137 [00:00<00:00, 11.5kB/s]
To show errors in colab notebook, set d
https://f11fbfd2e2e2019c8f.gradio.live
ing an
Possible Conditions & Recommendations
(101°F or higher)
Chills
Sore throat
Swollen lymph nodes
Fatigue
Headache
Nausea or vomiting
Possible medical conditions:
1. Influenza (Flu)
2. Streptococcal pharyngitis (Strep throat)
3. Mononucleosis (Mono)
4. Viral exanthems (e.g., measles, chickenpox)
5. Bacterial infections (e.g., pneumonia,

Possible Conditions & Recommendations

meningitis)
6. Allergic reactions
7. Viral hepatitis
8. Malaria
9. Dengue fever
10. Lyme disease

General medication suggestions:
1. Over-the-counter pain relievers (e.g., acetaminophen, ibuprofen) for fever, headache, and muscle aches.
2. Antihistamines (e.g., diphenhydramine) for allergic reactions or runny nose.
3. Decongestants (e.g., pseudoephedrine) for

Possible Conditions & Recommendations

bacterial infection is suspected, based on a healthcare professional's evaluation.

5. Antiviral medications (e.g., oseltamivir, famciclovir) for influenza or other viral exanthems, if prescribed by a healthcare professional.

6. Hydration and electrolyte replacement (e.g., oral rehydration solutions) for nausea, vomiting, or diarrhea.

7. Rest and sleep to manage fatigue.

Important note:

- Only a healthcare professional can accurately diagnose and treat these conditions.