Health AI : Intelligent Healthcare Assistant

Project Documentation

1. Introduction:

• Project Title : **Health Ai: Intelligent Healthcare Assistant**

• Team member : ***MadhanKumar V***

• Team member : ***Sibhi***

• Team member : ***Divya Bharathi A***

• Team member : ***Syed Aareefuddin F***

2. Project Overview:

- Purpose

The **Health AI Assistant** serves as an **informational and guidance tool** designed to help individuals understand their health conditions and gain insights into potential treatments based on their symptoms and medical history. The primary purpose of this project is to **assist with early symptom analysis** and provide **personalized treatment suggestions**, empowering users to make more informed decisions about their health. It also aims to enhance **health awareness** and **education** in a user-friendly, easily accessible format.

Specific Purposes:

1. **Symptom Analysis and Disease Prediction:**
   - **Purpose:** Help users quickly identify potential medical conditions based on the symptoms they provide.
   - This feature aids individuals in understanding what might be causing their symptoms, offering potential diagnoses in a general, non-specific manner. It emphasizes that **only a medical professional** can offer a definitive diagnosis.
2. **Personalized Treatment Recommendations:**
   - **Purpose:** Offer users tailored treatment plans based on their medical condition, age, gender, and medical history.
   - The AI generates personalized advice about home remedies, lifestyle changes, and medications, providing users with a **starting point** for their healthcare journey. However, it stresses the need for **professional consultation**.
3. **Health Education and Awareness:**
   - **Purpose:** Educate users about common health conditions, general symptoms, and available treatment options.

    - By interacting with the AI, users can gain a broader understanding of various health topics, helping them make more informed decisions about seeking medical advice or making lifestyle changes.

4. **Accessible Health Guidance:**
    - **Purpose:** Make health information **more accessible** to a larger audience, including those who may not have immediate access to healthcare professionals.
    - The platform provides a **low-barrier entry point** for health information, allowing people in remote or underserved areas to explore potential health concerns before visiting a doctor.
5. **Encourage Preventive Healthcare:**
    - **Purpose:** Promote early intervention and preventive healthcare by helping users recognize symptoms early and seek appropriate care.
    - By providing symptom analysis and suggesting possible conditions, the AI can encourage individuals to seek medical attention before conditions worsen, supporting proactive health management.

Overall Goal:

The ultimate goal of the **Health AI Assistant** is to **improve access to healthcare information**, **enhance user awareness about health issues**, and **offer personalized advice** in a responsible, ethical manner. It is intended to act as a **complementary tool** to traditional medical consultation rather than a replacement. The purpose is to make medical knowledge **more accessible, interactive, and tailored** to the needs of individuals, helping them navigate their health with greater confidence.

- *Key Features and Functionalities:*

1. **Symptom Analysis (Disease Prediction):**
    - Users can input a list of symptoms (e.g., headache, fever, cough), and the AI will generate a list of possible medical conditions associated with those symptoms.
    - It also provides general treatment suggestions such as over-the-counter medications or home remedies.
    - **Disclaimer:** The results are intended for informational purposes only and users are encouraged to consult with healthcare professionals for accurate diagnosis and treatment.
2. **Personalized Treatment Plan Generation:**
    - Users can input detailed information about their medical condition, age, gender, and medical history.
    - The assistant generates a tailored treatment plan, which includes home remedies, medications, and general lifestyle suggestions based on the provided information.

    - **Disclaimer:** Like the symptom analysis, the treatment plan is for guidance only and should not replace professional medical advice.

3. **Interactive and User-Friendly Interface:**
    - Built using **Gradio**, the app features a user-friendly interface with easy-to-use textboxes and buttons.
    - The app is organized into two main tabs: **Disease Prediction** and **Treatment Plans**, making it easy to switch between the different functionalities.
4. **AI Model Integration:**
    - The assistant uses the **Granite-3.2-2b-instruct** model by IBM, a pre-trained large language model designed to understand and generate human-like text based on prompts.
    - The model processes the input data, runs inference to predict possible medical conditions or generate treatment plans, and then presents the output in a natural, readable format.
5. **Deployment and Accessibility:**
    - The app is designed to run on a **Gradio Blocks** interface, which is accessible via a web browser.
    - It is deployed to be shared with others via a public URL, ensuring easy access for all users (even if they don't have a technical background).
    - The app is server-based and can be accessed from anywhere with an internet connection.

3. Architecture:

1. User Interface Layer (Frontend)

- **Technology**: Gradio
- **Purpose**:
    - Provides an interactive interface for users to input symptoms or medical details and receive AI-generated responses (disease predictions or treatment plans).
    - Utilizes **Gradio Blocks** to organize the layout, with **tabs** for each functionality, including:
        - **Symptom Analysis** (Disease Prediction Tab)
        - **Treatment Plan Generation** (Treatment Plan Tab)

  Components:
    - **Textboxes**: For entering symptoms, medical conditions, and personal health details (age, gender, medical history).

    - **Buttons**: To trigger disease prediction and treatment plan generation.
    - **Dropdown**: For selecting the gender.
    - **Markdown**: For displaying the AI's response (predictions or treatment plans).

---

2. Application Logic Layer (Backend)

- **Technology**: Python, Gradio, Transformers, PyTorch
- **Purpose**:
    - Handles the business logic of the application, including data preprocessing, calling the AI model for inference, and post-processing the AI's responses.

  Components:

    - **Model Loading**: The AI model (`granite-3.2-2b-instruct`) is loaded using the **Transformers library** by Hugging Face. If a GPU is available, it loads in float16 precision for faster processing.
    - **Input Handling**: Takes input from the frontend (via Gradio textboxes, dropdowns) and prepares it for the model. For example:
        - Symptom analysis takes symptom data from the user and formats it into a prompt for the model.
        - Treatment plan generation uses data like medical condition, age, gender, and medical history to generate personalized suggestions.
    - **Model Inference**: The model processes the input data and generates predictions or treatment recommendations based on pre-trained knowledge.
    - **Post-Processing**: After model inference, the output text is cleaned (e.g., removing the original prompt) and returned to the frontend as a meaningful result.

---

3. AI Model Layer (Machine Learning)

- **Technology**: Hugging Face Transformers, PyTorch
- **Purpose**:
    - The AI model (`granite-3.2-2b-instruct`) processes the input data and generates meaningful responses (disease predictions or treatment plans).

  Components:

    - **Pre-trained Model**: The `granite-3.2-2b-instruct` model is fine-tuned to understand and respond to medical queries, processing both symptoms and medical history.
    - **Tokenizer**: Converts user input into tokenized text for the model.

- **Model Inference**: Using **PyTorch**, the model performs inference to generate human-like text based on the input data.

4. Setup Instructions:

Step-by-Step Instructions for Running the Health AI Assistant

1. Install Required Libraries

Run the following commands to install the necessary dependencies:

```
pip install torch torchvision torchaudio
pip install transformers
pip install gradio
```

---

2. Set Up Python Script

- Create a new Python file (e.g., `health_ai_assistant.py`).
- Copy and paste the provided code into this file.

---

3. Run the Script

In the terminal, navigate to the directory where your Python file is saved and run:

```
python health_ai_assistant.py
```

---

4. Open the Gradio Interface

Once the script is running, access the app through:

- **Local URL**: `http://127.0.0.1:7860`
- **Public URL**: (shared by Gradio)

---

5. Interact with the Application

- **Disease Prediction**: Enter symptoms (e.g., fever, cough) and click **"Analyze Symptoms"**.
- **Treatment Plan**: Enter medical details (e.g., condition, age, medical history) and click **"Generate Treatment Plan"**.

---

6. Modify Parameters (Optional)

You can adjust:

- **Max Length**: Change `max_length` in `generate_response` to control response length.
- **Temperature**: Adjust response randomness by modifying `temperature`.

---

7. Deploy (Optional)

To make the app publicly accessible, use the **Gradio share link** or deploy it on platforms like **Heroku** or **AWS**.

---

8. End Session

Stop the server with `Ctrl + C` in the terminal when done.

Summary

1. Install libraries: `pip install torch transformers gradio`
2. Save the code as `health_ai_assistant.py`.
3. Run with: `python health_ai_assistant.py`.
4. Access via the provided URL.
5. Interact with the AI for predictions and treatment plans.

6. Folder Structure:

Here's a suggested **folder structure** for the Health AI Assistant project, keeping the code organized and scalable:

Folder Structure

```
health_ai_assistant/
│
├── assets/                       # Optional folder for images or additional
assets
│   └── logo.png                  # Example: an image logo if needed
│
├── models/                       # Folder for model files (optional if you
store locally)
│   └── granite-3.2-2b-instruct/ # Optional: Folder for saving model
checkpoints
│
├── requirements.txt              # List of dependencies for easy installation
├── health_ai_assistant.py        # Main Python script for running the app
└── README.md                     # Project documentation
```

Explanation of Each Folder/File:

1. **`health_ai_assistant.py`**:
   - The main Python script where the code resides. It contains the logic for loading the model, interacting with the user, and generating responses via Gradio.
2. **`requirements.txt`**:

- A file that lists all the required dependencies, so others can easily set up the environment by running:

```
3. pip install -r requirements.txt
```

Example `requirements.txt`:

```
torch
transformers
gradio
```

4. **`assets/`**:
   - A folder to store any extra assets, such as images or logos, that may be used in the application or interface (e.g., a logo for the Gradio interface).
5. **`models/`**:
   - An optional folder if you want to **store model checkpoints locally** for faster loading. For example, if you want to download and store the model on your local machine rather than fetching it each time.
6. **`README.md`**:
   - A file for documenting the project. This would explain how to set up, run, and use the application. It's helpful for collaborators or anyone new to the project.

How to Organize the Files

1. **Download and Store Models Locally (Optional)**:
   - If you choose to download the model to avoid fetching it from Hugging Face each time, create a `models/` folder where you can save the model and tokenizer.
2. **Place Additional Resources in `assets/`**:
   - If you need to add visual resources or configuration files for the application (such as images for the Gradio UI), put them inside the `assets/` folder.

Example Project Structure:

```
health_ai_assistant/
│
├── assets/
│   └── logo.png
│
├── models/
│   └── granite-3.2-2b-instruct/
│
├── requirements.txt
├── health_ai_assistant.py
└── README.md
```

7. Running the Application:

Follow these steps to run the Health AI Assistant application based on the provided code.

---

1. Install Required Libraries

First, you need to install all the necessary Python dependencies. Open a terminal or command prompt and run the following commands:

```
pip install torch torchvision torchaudio
pip install transformers
pip install gradio
```

These libraries are crucial for loading the AI model, handling the model's inference, and providing the user interface.

---

2. Set Up the Project Folder

Create a folder structure as follows (or use the structure I previously suggested):

```
health_ai_assistant/
│
├── assets/                    # Optional folder for images or additional
assets
│
├── models/                    # Folder for model files (optional if
storing locally)
│
├── requirements.txt           # List of dependencies for easy installation
├── health_ai_assistant.py     # Main Python script for running the app
└── README.md                  # Project documentation
```

- **`health_ai_assistant.py`**: The main Python file with the logic for loading the model, processing inputs, and generating responses.
- **`requirements.txt`**: This file lists the dependencies, which you can easily install using `pip install -r requirements.txt`.

---

3. Download and Set Up the Model

By default, the model is fetched from Hugging Face using the following line in the code:

```
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)
```

The model will be downloaded automatically the first time you run the application. If you'd prefer to store the model locally to avoid downloading it each time:

1. Create a `models/` folder in your project directory.
2. Download the model manually from Hugging Face or use the `transformers` library to cache the model in this folder.

---

4. Create `requirements.txt`

To make setting up the project easier for others (or for yourself later), create a `requirements.txt` file with the following contents:

```
torch
transformers
gradio
```

To install all dependencies in one go, run:

```
pip install -r requirements.txt
```

---

5. Run the Application

Once the dependencies are installed and your folder structure is set up:

1. Open a terminal.
2. Navigate to the folder where your `health_ai_assistant.py` file is located.
3. Run the Python script:

```
python health_ai_assistant.py
```

---

6. Access the Gradio Interface

After running the script, you'll see output similar to the following in the terminal:

```
Running on local URL:  http://127.0.0.1:7860
Running on public URL: https://<random-string>.gradio.app
```

- **Local URL**: You can open `http://127.0.0.1:7860` in your browser to view the Gradio interface locally.
- **Public URL**: Gradio will also generate a public URL that you can share with others. This URL is temporary and will expire after a session ends.

---

7. Interact with the Application

1. **Disease Prediction**:
    - In the **"Disease Prediction"** tab, type in symptoms (e.g., "fever, headache, cough") and click **"Analyze Symptoms"**.

   - The AI will process the input and provide a response with possible conditions and medication suggestions.
2. **Treatment Plan**:
   - In the **"Treatment Plan"** tab, input medical details like:
     - **Medical Condition**: e.g., "Diabetes"
     - **Age**: e.g., 45
     - **Gender**: Male
     - **Medical History**: e.g., "None"
   - Click **"Generate Treatment Plan"** to receive a personalized treatment suggestion.

---

8. Modify Parameters (Optional)

You can customize certain aspects of the application:

- **Change Max Length**: Modify the response length by changing the `max_length` parameter in the `generate_response` function.
- **Adjust Temperature**: Change the `temperature` setting in the response generation to control randomness (higher = more varied responses).

---

9. Stop the Application

Once you are done, you can stop the Gradio interface by pressing **Ctrl + C** in the terminal.

Summary of Steps:

1. Install dependencies: `pip install torch transformers gradio`
2. Set up the project folder and `requirements.txt`.
3. Run the script: `python health_ai_assistant.py`.
4. Open the local or public URL provided by Gradio.
5. Input symptoms or medical details to interact with the AI assistant.
6. (Optional) Modify response parameters or deploy the app.
7. End the session by stopping the server with **Ctrl + C**.

7. API Documentation:

This documentation provides an overview of the API used in the **Health AI Assistant** application, which predicts possible diseases based on symptoms and generates personalized treatment plans for various medical conditions.

The application is built with **Gradio** for the UI, **Transformers** for the AI model, and **PyTorch** for model inference.

API Endpoints

The Health AI Assistant has two primary functionalities:

1. **Disease Prediction** based on provided symptoms.
2. **Treatment Plan Generation** based on medical conditions, age, gender, and medical history.

Both functionalities are wrapped inside the Gradio interface, making the interaction seamless for the user. The Gradio app serves as the frontend that connects to the backend functions.

---

1. Disease Prediction API

Endpoint: `/predict_disease`
Method: `POST`

Description: Predicts possible medical conditions based on the provided symptoms.

Request Parameters:

- **`symptoms`** (string, required): A comma-separated list of symptoms experienced by the user (e.g., "fever, headache, cough, fatigue").

Request Example:

```
{
  "symptoms": "fever, headache, cough, fatigue"
}
```

Response:

- **`possible_conditions`** (string): A list of possible medical conditions that could be related to the provided symptoms.
- **`recommendations`** (string): General recommendations for treatment, such as rest, hydration, or over-the-counter medication.

Response Example:

```
{
  "possible_conditions": "Common cold, Flu, COVID-19",
  "recommendations": "Rest, Hydration, Over-the-counter medications for fever
and cough. Consult a healthcare provider for further diagnosis."
}
```

---

2. Treatment Plan API

Endpoint: `/generate_treatment_plan`
Method: `POST`

Description: Generates a personalized treatment plan for the user based on their medical condition, age, gender, and medical history.

Request Parameters:

- **`condition`** (string, required): The medical condition for which the treatment plan is requested (e.g., "diabetes", "hypertension").

- **`age`** (integer, required): The age of the patient (e.g., 45).
- **`gender`** (string, required): The gender of the patient (e.g., "Male", "Female", "Other").
- **`medical_history`** (string, required): Any relevant medical history such as "hypertension", "allergies", or "None".

Request Example:

```
{
  "condition": "Diabetes",
  "age": 45,
  "gender": "Male",
  "medical_history": "None"
}
```

Response:

- **`treatment_plan`** (string): A detailed treatment plan that includes both home remedies and medication guidelines based on the provided condition and patient information.

Response Example:

```
{
  "treatment_plan": "For Diabetes management: 1. Follow a low-carb,
high-fiber diet. 2. Exercise regularly. 3. Take prescribed medication like
metformin as advised by a healthcare professional. 4. Monitor blood sugar
levels daily. 5. Consult your doctor for further advice and possible insulin
therapy."
}
```

8. Authentication:

To add authentication to the **Health AI Assistant** app, we can implement simple token-based authentication to secure access to the application. This ensures that only authorized users can interact with the app, especially if it's deployed in a public environment.

In this case, we'll use a basic API key authentication system where the user has to provide a valid API key to access the app's functionality. Below are the steps for implementing this.

Steps for Adding Authentication

---

1. Create an API Key

Before implementing the authentication, you need to create an API key. For simplicity, let's assume a hardcoded API key. In a real-world application, you might want to store this key securely in a database or environment variable.

For example:

```
API_KEY = "your-secret-api-key-here"
```

---

2. Modify the Backend Code to Check the API Key

We will modify the existing code to check if a valid API key is provided before processing any requests. For simplicity, we'll add the authentication as part of the `Gradio` UI.

We'll update the existing Gradio blocks to accept an API key from the user and check if it matches the expected key before allowing them to use the app.

3. Adding API Key Authentication to the Gradio Interface

Here's how you can modify the code:

Step 1: Define the API Key

At the beginning of your script, define the API key you want to use for authentication.

```
# Define a hardcoded API key for authentication
API_KEY = "your-secret-api-key-here"
```

Step 2: Authentication Function

Create a function that checks whether the API key provided by the user matches the hardcoded one.

Step 3: Add API Key Field in Gradio Interface

Add an input field for the user to enter their API key in the UI.

9. User Interface:

To create an intuitive and user-friendly interface for the **Health AI Assistant**, we'll use **Gradio** to build an interactive web-based UI. The provided code already includes a basic framework for this, but we'll enhance it by adding some UI elements like input fields, buttons, tabs, and output containers.

Here's a complete breakdown of the **User Interface (UI)** for the application, as implemented with **Gradio**:

Health AI Assistant User Interface (UI)

Overview:

- The app will have two main functionalities available through tabs:
    1. **Disease Prediction**: Allows the user to enter symptoms and get possible medical conditions.
    2. **Treatment Plan**: Allows the user to input medical condition, age, gender, and medical history to generate a personalized treatment plan.

UI Flow:

1. **API Key Input (Authentication)**
2. **Main Interface (After API Key Validation)**
    - Disease Prediction Tab
    - Treatment Plan Tab
3. **Results Display**

1. Authentication Screen:

- **API Key Input**: Before the main interface, the user will be asked to input a valid API key to access the app. If the API key is incorrect, they will see an error message.

UI Elements:

- **Textbox**: To input the API key.
- **Button**: To submit the API key.
- **Markdown Output**: To show error messages or confirm valid access.

2. Main Interface (After Authentication)

Once the correct API key is entered, the user will be presented with the following two main features in a **Gradio Tabs** layout.

UI Elements:

- **Tabs**: To switch between Disease Prediction and Treatment Plan sections.
- **Textboxes**: For users to input symptoms, medical conditions, age, gender, and medical history.
- **Buttons**: For submitting the input and generating results.
- **Markdown Output**: To display the generated results (disease prediction or treatment plan).

10. Testing:

1. Unit Tests

Use **pytest** to directly test `disease_prediction` and `treatment_plan`.

- Verify outputs are strings.
- Check disclaimer is always present.
- Handle empty inputs gracefully.
- Validate history-based treatment plans return results without crashing.

2. API Tests

Use **requests** to test the Gradio API endpoints.

- Confirm server responds with status `200`.
- Disease prediction endpoint returns output including disclaimer.
- Treatment plan endpoint responds with structured text and disclaimer.

---

3. Manual Testing

Checklist for human testers:

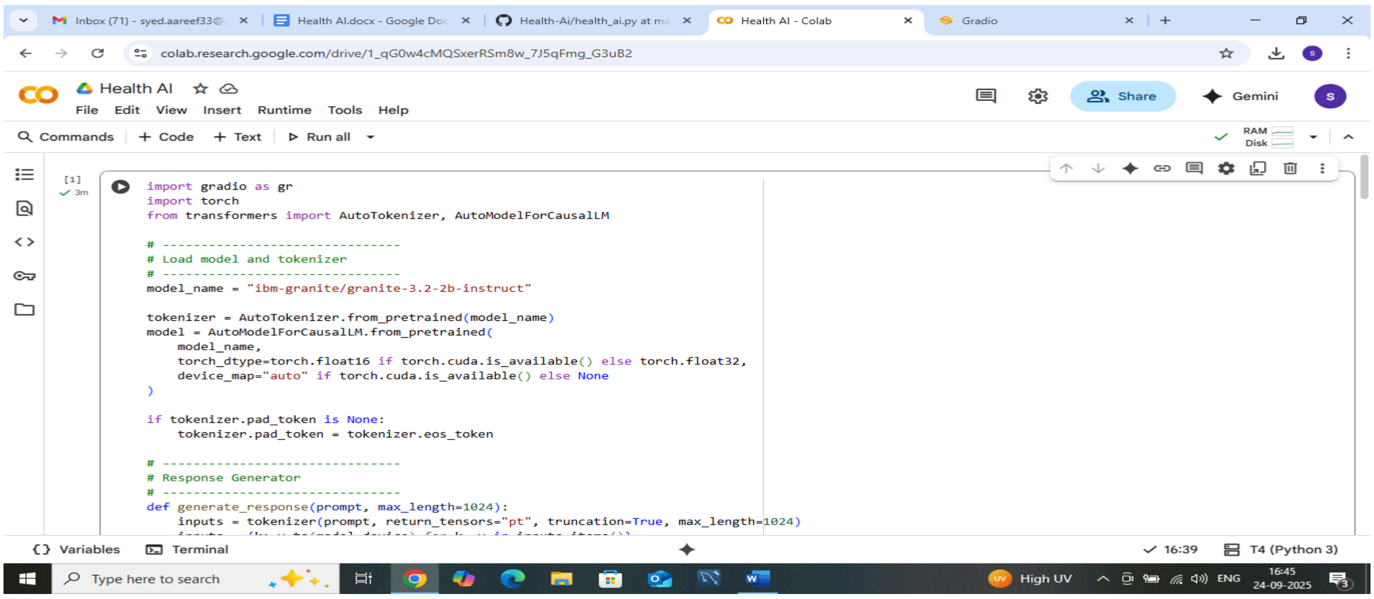
- Enter basic symptoms → expect conditions + disclaimer.
- Enter condition + demographics → expect plan + disclaimer.
- Empty input → should return safe message, not crash.
- Special characters or gibberish → should still respond safely.
- Very long input → app truncates but responds meaningfully.

---

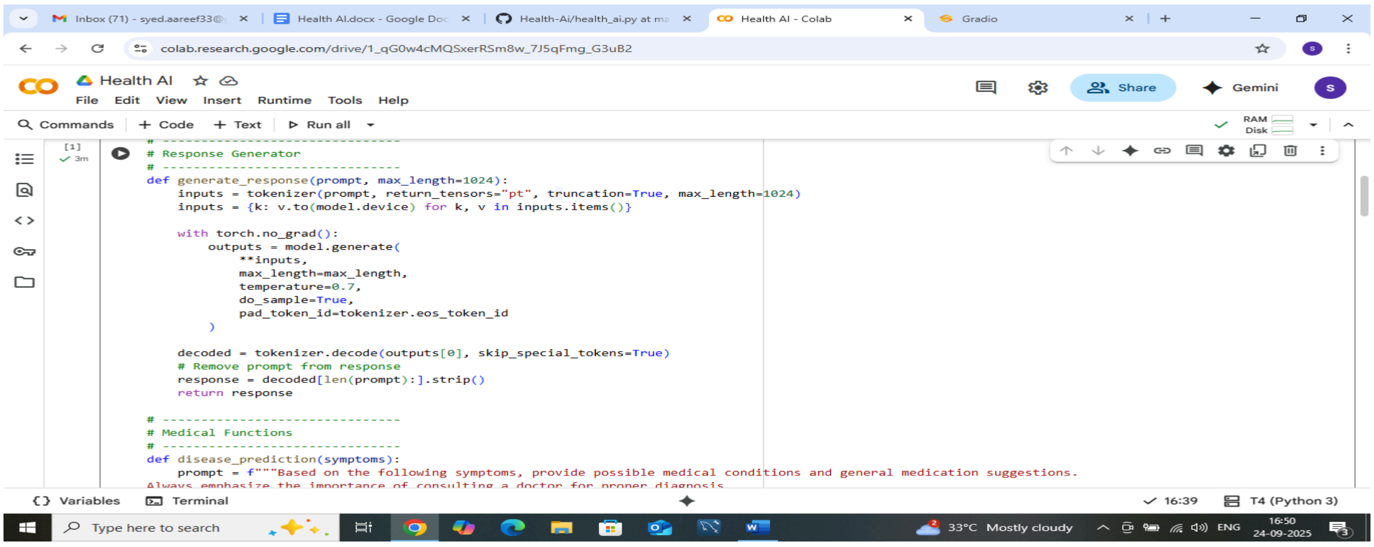
4. Edge Case Tests

- **Empty symptoms** → return disclaimer only.
- **Gibberish input** → return disclaimer, no crash.
- **Very long input** → handle truncation safely.
- **Unusual gender** (`Other`) → accepted gracefully.
- **Boundary age values** (0, 120) → safe response, no errors.
- **Malicious text** (`DROP TABLE users;`) → treated as plain text.
- **Multiple conditions** → provide broad advice with disclaimer.

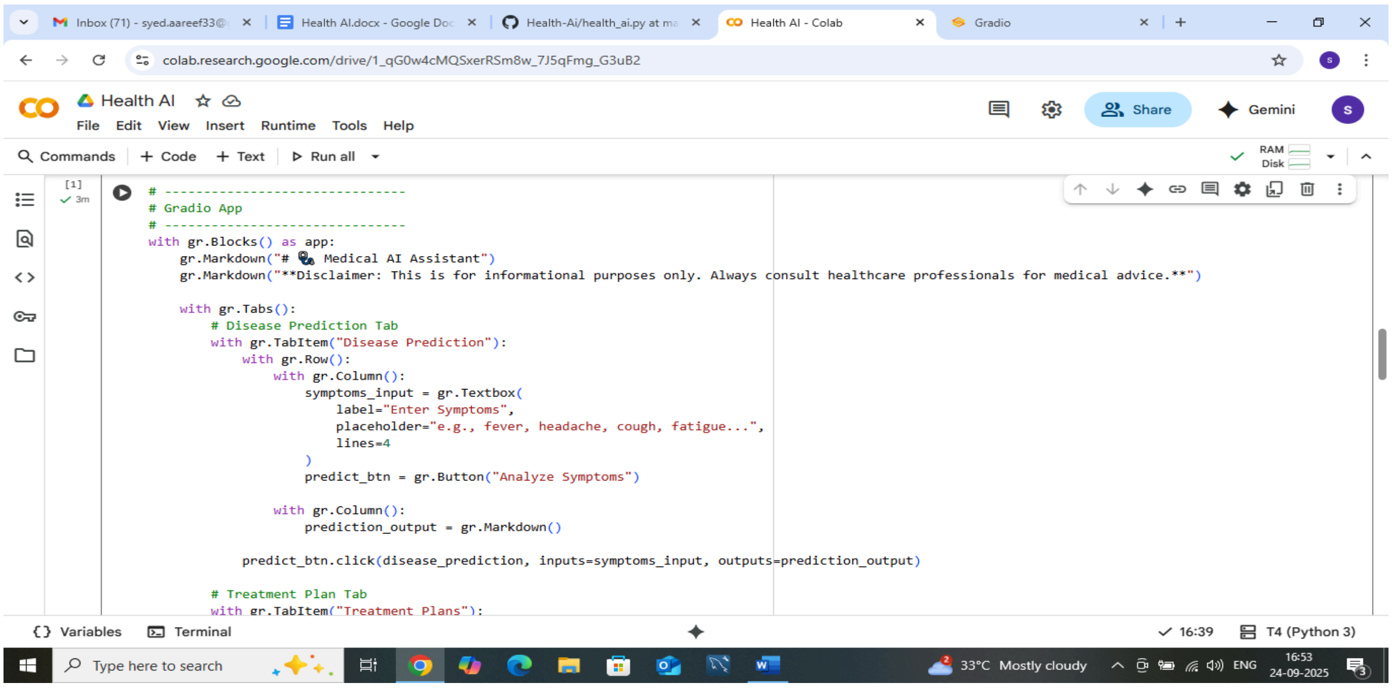
11.Screen Shots:



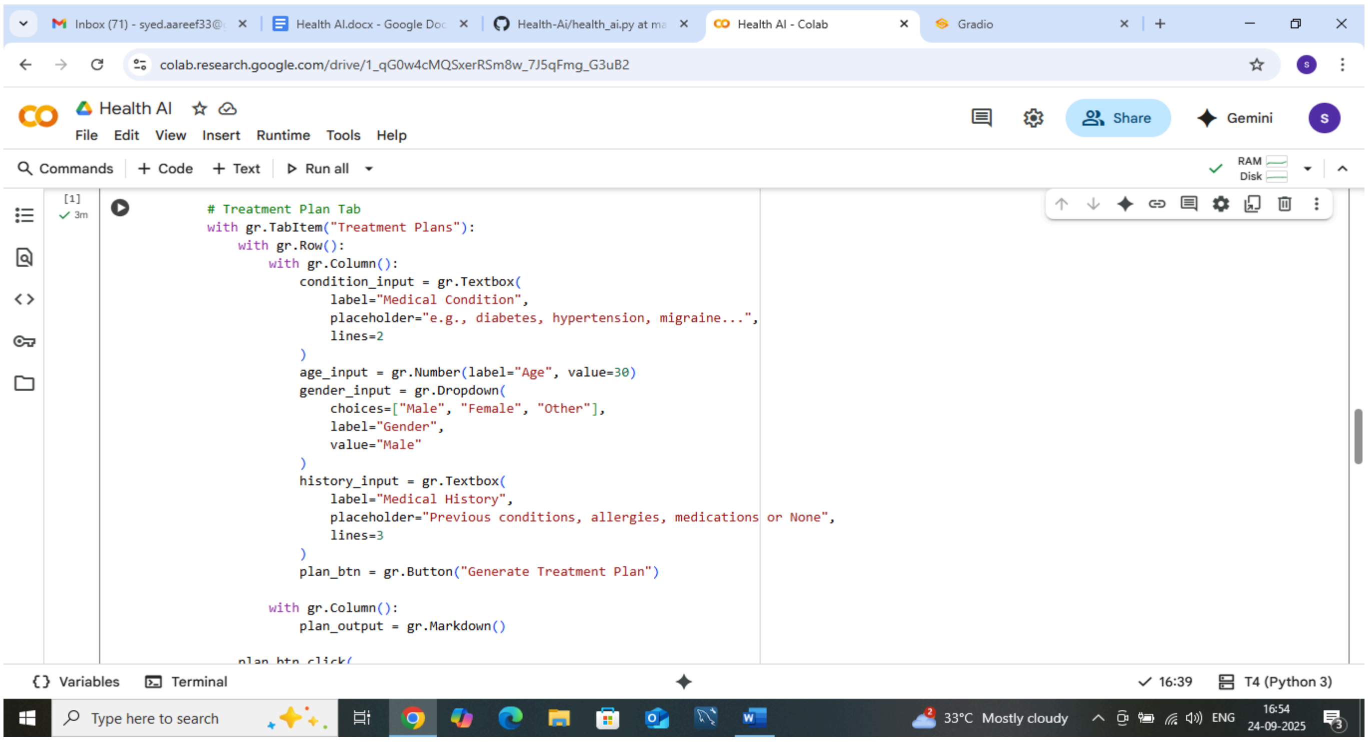
```python
import gradio as gr
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM

# ------------------------------
# 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

# ------------------------------
# Response Generator
# ------------------------------
def generate_response(prompt, max_length=1024):
    inputs = tokenizer(prompt, return_tensors="pt", truncation=True, max_length=1024)
```



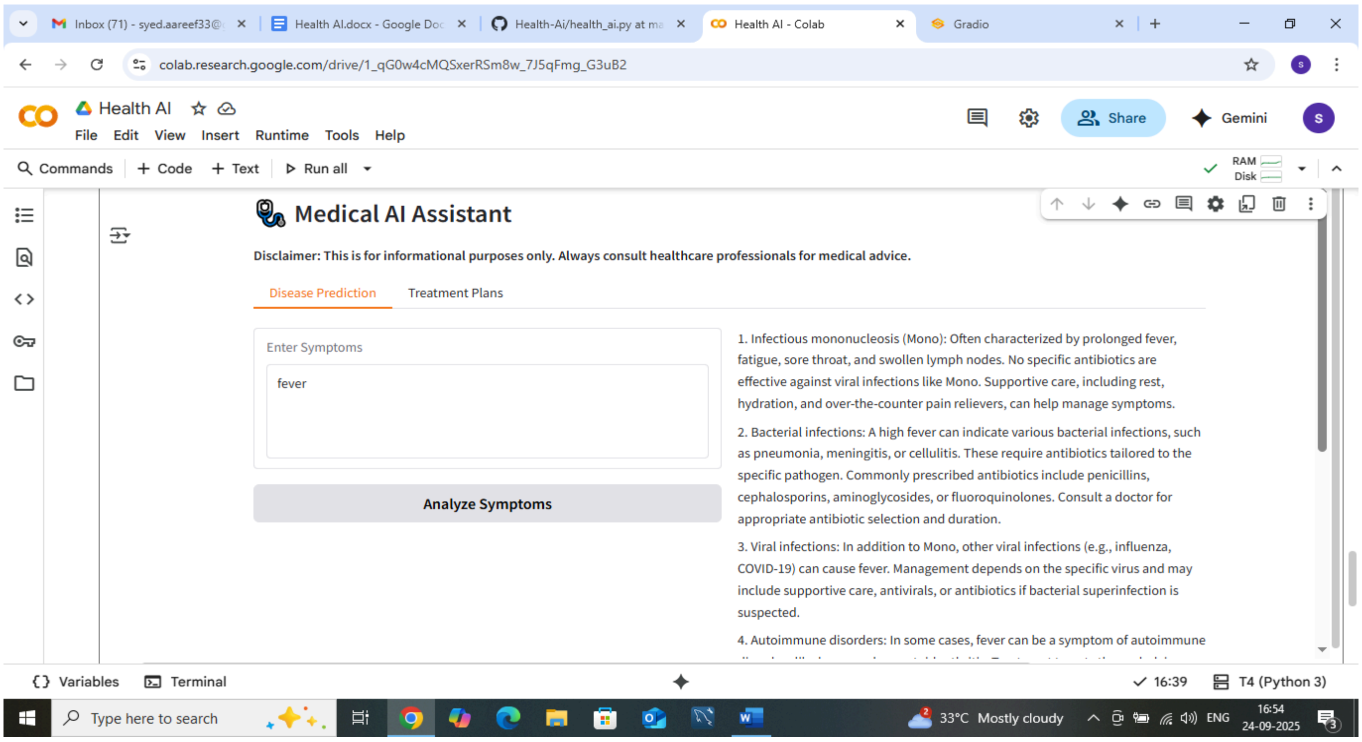
```python
# Response Generator
# ------------------------------
def generate_response(prompt, max_length=1024):
    inputs = tokenizer(prompt, return_tensors="pt", truncation=True, max_length=1024)
    inputs = {k: v.to(model.device) for k, v in inputs.items()}

    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
        )

    decoded = tokenizer.decode(outputs[0], skip_special_tokens=True)
    # Remove prompt from response
    response = decoded[len(prompt):].strip()
    return response

# ------------------------------
# Medical Functions
# ------------------------------
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 doctor for proper diagnosis.
```


Inbox (71) - syed.aareef33@
Health AI.docx - Google Doc
Health-Ai/health_ai.py at ma
Health AI - Colab
Gradio
colab.research.google.com/drive/1_qG0w4cMQSxerRSm8w_7J5qFmg_G3uB2
Health AI
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands + Code + Text Run all
RAM Disk
[1] 3m
Variables Terminal
16:39 T4 (Python 3)
Type here to search
33°C Mostly cloudy ENG 16:53 24-09-2025

```python
# -------------------------------
# Gradio App
# -------------------------------
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():
        # Disease Prediction Tab
        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.Markdown()

            predict_btn.click(disease_prediction, inputs=symptoms_input, outputs=prediction_output)

        # Treatment Plan Tab
        with gr.TabItem("Treatment Plans"):
```


Inbox (71) - syed.aareef33@
Health AI.docx - Google Doc
Health-Ai/health_ai.py at ma
Health AI - Colab
Gradio
colab.research.google.com/drive/1_qG0w4cMQSxerRSm8w_7J5qFmg_G3uB2
Health AI
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands + Code + Text Run all
RAM Disk
[1] 3m
Variables Terminal
16:39 T4 (Python 3)
Type here to search
33°C Mostly cloudy ENG 16:54 24-09-2025

```python
        # Treatment Plan Tab
        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
                    )
                    plan_btn = gr.Button("Generate Treatment Plan")

                with gr.Column():
                    plan_output = gr.Markdown()

            plan_btn.click(
```



Inbox (71) - syed.aareef33@
Health AI.docx - Google Doc
Health-Ai/health_ai.py at ma
Health AI - Colab
Gradio
colab.research.google.com/drive/1_qG0w4cMQSxerRSm8w_7J5qFmg_G3uB2
Health AI
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands
+ Code
+ Text
Run all
RAM
Disk
Variables
Terminal
16:39
T4 (Python 3)
Type here to search
33°C Mostly cloudy
ENG
16:54
24-09-2025

Medical AI Assistant

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

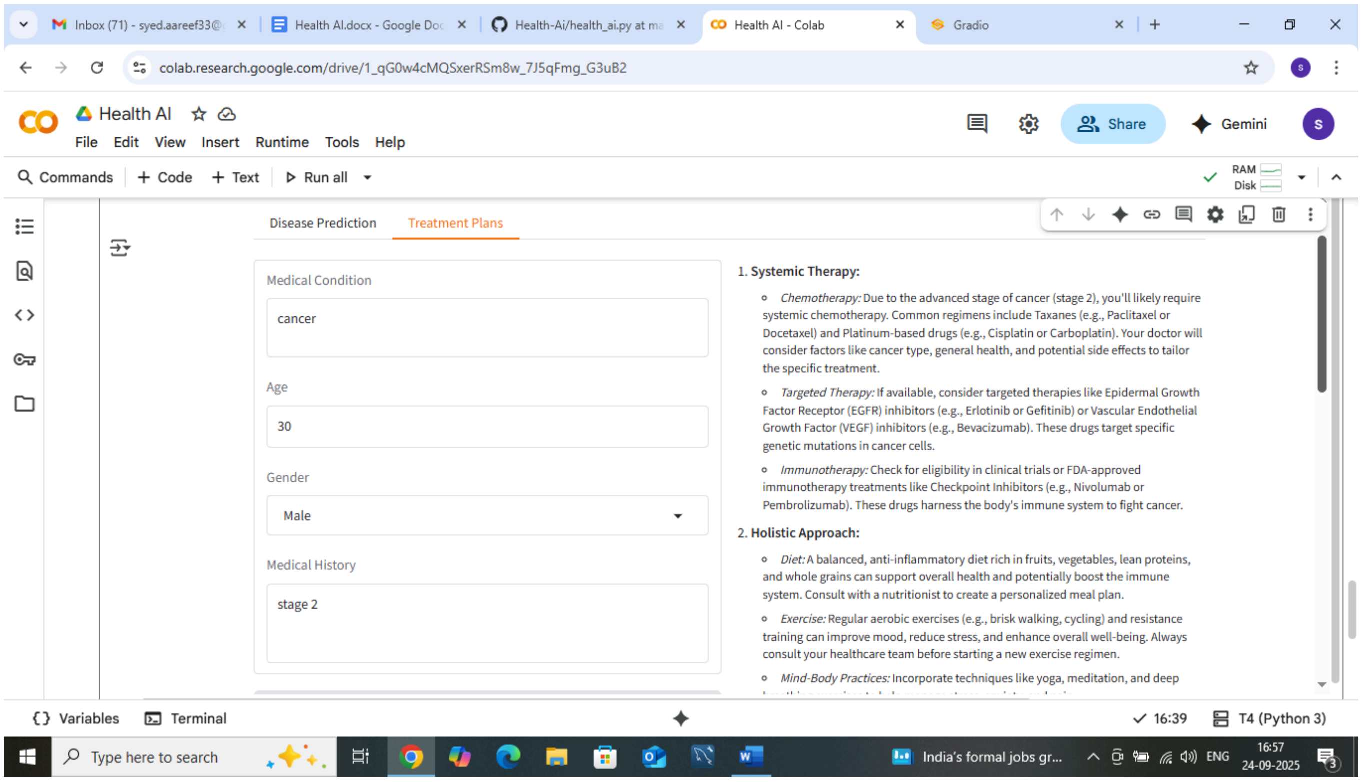
Disease Prediction | Treatment Plans

Enter Symptoms

fever

Analyze Symptoms

1. Infectious mononucleosis (Mono): Often characterized by prolonged fever, fatigue, sore throat, and swollen lymph nodes. No specific antibiotics are effective against viral infections like Mono. Supportive care, including rest, hydration, and over-the-counter pain relievers, can help manage symptoms.
2. Bacterial infections: A high fever can indicate various bacterial infections, such as pneumonia, meningitis, or cellulitis. These require antibiotics tailored to the specific pathogen. Commonly prescribed antibiotics include penicillins, cephalosporins, aminoglycosides, or fluoroquinolones. Consult a doctor for appropriate antibiotic selection and duration.
3. Viral infections: In addition to Mono, other viral infections (e.g., influenza, COVID-19) can cause fever. Management depends on the specific virus and may include supportive care, antivirals, or antibiotics if bacterial superinfection is suspected.
4. Autoimmune disorders: In some cases, fever can be a symptom of autoimmune



Inbox (71) - syed.aareef33@
Health AI.docx - Google Doc
Health-Ai/health_ai.py at ma
Health AI - Colab
Gradio
colab.research.google.com/drive/1_qG0w4cMQSxerRSm8w_7J5qFmg_G3uB2
Health AI
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands
+ Code
+ Text
Run all
RAM
Disk
Variables
Terminal
16:39
T4 (Python 3)
Type here to search
India's formal jobs gr...
ENG
16:57
24-09-2025

Disease Prediction | Treatment Plans

Medical Condition

cancer

Age

30

Gender

Male

Medical History

stage 2

1. **Systemic Therapy:**
   - *Chemotherapy:* Due to the advanced stage of cancer (stage 2), you'll likely require systemic chemotherapy. Common regimens include Taxanes (e.g., Paclitaxel or Docetaxel) and Platinum-based drugs (e.g., Cisplatin or Carboplatin). Your doctor will consider factors like cancer type, general health, and potential side effects to tailor the specific treatment.
   - *Targeted Therapy:* If available, consider targeted therapies like Epidermal Growth Factor Receptor (EGFR) inhibitors (e.g., Erlotinib or Gefitinib) or Vascular Endothelial Growth Factor (VEGF) inhibitors (e.g., Bevacizumab). These drugs target specific genetic mutations in cancer cells.
   - *Immunotherapy:* Check for eligibility in clinical trials or FDA-approved immunotherapy treatments like Checkpoint Inhibitors (e.g., Nivolumab or Pembrolizumab). These drugs harness the body's immune system to fight cancer.
2. **Holistic Approach:**
   - *Diet:* A balanced, anti-inflammatory diet rich in fruits, vegetables, lean proteins, and whole grains can support overall health and potentially boost the immune system. Consult with a nutritionist to create a personalized meal plan.
   - *Exercise:* Regular aerobic exercises (e.g., brisk walking, cycling) and resistance training can improve mood, reduce stress, and enhance overall well-being. Always consult your healthcare team before starting a new exercise regimen.
   - *Mind-Body Practices:* Incorporate techniques like yoga, meditation, and deep

12. Known Issues:

1. Medical Accuracy & Safety

- Model (`ibm-granite-3.2-2b-instruct`) is **not trained for clinical accuracy** → outputs may be misleading.
- Risk of **hallucinated medical conditions or treatments**.
- No validation against real-world medical guidelines (e.g., WHO, ICMR, FDA).
- Can produce **unsafe medication suggestions** without dosage context.

---

2. Ethical & Legal Concerns

- Even with disclaimers, users might **misuse it as medical advice**.
- Non-compliant with **healthcare regulations** (HIPAA, GDPR, ICMR guidelines).
- No **data privacy guarantees** → sensitive patient info is handled as plain text.

---

3. Technical Limitations

- **Token limit (1024 / 1200)** → long symptom histories may get truncated.
- **Prompt-based outputs** → inconsistent results across runs (sampling + temperature=0.7).
- **Latency** → model inference may be slow on CPU.
- No **caching or logging** for repeated queries.

---

4. User Input Issues

- Accepts **free text only** → prone to spelling errors, vague inputs, or irrelevant data.
- No validation for **age ranges** (e.g., negative numbers, unrealistic values).
- Limited **gender options** (Male/Female/Other) without inclusivity for real-world variations.
- Cannot handle **multi-language symptoms** (English only).

---

5. UI/UX Issues

- Basic Gradio UI → no structured output (tables, charts, severity scoring).
- No **feedback loop** (e.g., “Was this answer helpful?”).
- No **history** → users cannot track past responses.
- Disclaimer present, but could be **overlooked by users**.

---

6. Edge Case Failures

- **Empty input** → returns generic disclaimer, may confuse users.
- **Gibberish input** → still tries to respond meaningfully, may mislead.
- **Malicious text input** (like SQL injection) → not dangerous in this app, but shows lack of sanitization.
- **Extremely long text** → gets truncated, response may lose context.

13. Future enhancement:

1. Medical Accuracy & Safety

- Integrate **medically fine-tuned models** (e.g., PubMed, BioGPT, MedPalm-like models).

- Add **knowledge grounding** with verified medical datasets or APIs (WHO, CDC, ICMR).
- Introduce a **safety filter** to block unsafe treatment/medicine recommendations.
- Provide **probability or confidence scores** with each prediction.

---

2. Regulatory & Compliance

- Add **data anonymization** to protect patient information.
- Ensure compliance with **HIPAA / GDPR / ICMR** depending on deployment region.
- Explicit **consent mechanism** before user inputs sensitive health data.

---

3. Input Handling

- Support **structured input forms** (checkboxes, dropdowns for common symptoms, medications).
- Add **multilingual support** (Hindi, Tamil, etc.) for inclusivity.
- Implement **spell correction / NLP preprocessing** for symptom entry.
- Validate **age ranges, gender inputs, and medical history** fields.

---

4. Output Improvements

- Present responses in **structured format** (tables, bullet points, severity categories).
- Include **“Red flag” alerts** when symptoms suggest emergency conditions.
- Provide **links to verified resources** (Mayo Clinic, WHO, NHS).
- Enable **export to PDF** so users can share reports with doctors.

---

5. User Experience (UX)

- Add **chat history** so users can revisit previous queries.
- Introduce **voice input and text-to-speech output** for accessibility.
- Add a **feedback system** (“Was this helpful?”) to refine results.
- Dark mode / mobile-friendly UI for wider usability.

---

6. Technical Enhancements

- Implement **response caching** to improve performance.
- Add **logging & monitoring** for tracking usage patterns and issues.
- Deploy with **GPU acceleration** (if available) for faster inference.
- Provide a **REST API or FastAPI wrapper** so it can be integrated into other systems.

---

7. Edge Case & Error Handling

- Gracefully handle **empty, invalid, or excessively long inputs**.
- Add **rate limiting** to prevent abuse.
- Implement **fallback responses** when model confidence is low.
- Explicitly refuse queries unrelated to health (e.g., “How to hack wifi”).