Health AI intelligent healthcare assistant using IBM granite

Project Documentation

1.Introduction

• Project title :Health AI intelligent healthcare assistant using IBM granite

• Team member : Yamini S

• Team member : Nandhini R

•Team member : Sathya R

• Team member : Nega G

• Team member: Parameshwari P

2.project overview

• **Purpose :**

The purpose of the Health AI project is to:

1. Enhance Clinical Decision-Making: Assist healthcare professionals with AI-driven diagnostics, personalized treatment recommendations, and predictive risk assessments.

2. Streamline Operations: Automate routine administrative tasks such as medical documentation, billing, and patient communication to reduce burnout and increase efficiency.

3. Improve Patient Outcomes: Enable early detection of diseases and better care management through data-driven insights and continuous monitoring.

4. Accelerate Research: Use AI to analyze large datasets (e.g., medical records, clinical trials, genomic data) to uncover new patterns, support drug discovery, and foster innovation.

5. Ensure Ethical and Secure AI Use: Employ IBM Granite's enterprise-grade AI models that prioritize transparency, fairness, and data privacy in all healthcare applications.

3. Architecture

HealthAI Architecture

Components

1. Frontend: User interface (UI) built with HTML, CSS, and JavaScript.

2. Backend: FastAPI handles routing, sessions, and integration with AI models.

3. AI Engine: IBM WatsonX Granite models power symptom analysis and treatment suggestions.

4. Data Layer: Stores session and patient data.

Key Technologies

1. IBM Granite Models: Power AI-driven insights and recommendations.

2. FastAPI: Enables efficient backend operations.

3. Streamlit: Provides interactive UI components.

4. Setup Instructions

1. Install Required Libraries: Install necessary libraries, such as `ibm-watson` and `streamlit`, using pip.

2. Configure IBM Granite API Keys: Set up API keys for IBM Granite services in your Python environment.

3. Create a Streamlit App: Build a Streamlit app to create a user interface for your Health AI assistant.

4. Integrate IBM Granite Services: Use IBM Granite's NLP capabilities to analyze user input and provide insights.

5. Develop ML Models: Train and deploy machine learning models to provide personalized insights and recommendations.

6. Test and Deploy: Test your Health AI assistant and deploy it on a cloud platform (e.g., IBM Cloud) or locally.

Additional Resources

1. IBM Granite Documentation: Refer to IBM Granite's documentation for detailed setup instructions and API usage.

2. Streamlit Documentation: Use Streamlit's documentation to build and customize your user interface.

3. GitHub Repositories: Explore open-source repositories (e.g., Health-ai) for example code and implementation details.

5. Folder Structure

Main Folders

1. app: Application cod

•routes: API endpoint

- models: Data models and AI integration
- services: Business logic and AI interactions
- utils: Utility functions

2. config: Configuration files
   - (link unavailable): Environment variables and settings
3. data: Data storage and management
   - patient_data: Sample patient data
4. frontend: User interface code
   - templates: HTML templates
   - static: CSS, JavaScript, and other static files
5. tests: Unit tests and integration tests

Key Files

- (link unavailable): Application entry point
- requirements.txt: Dependencies and libraries
- (link unavailable): Project documentation and setup instructions

6. Running the Application

1.Start the Streamlit App: Run the Streamlit app using the command `streamlit run your_app.py` (replace `your_app.py` with your Python file).

2. Launch the Application: Open a web browser and navigate to `http://localhost:8501` (default Streamlit port) to access the Health AI assistant.

3. Interact with the Application: Users can interact with the application by asking health-related questions, providing symptoms, or seeking advice.

Application Flow:

1. User Input: Users provide input through the application's user interface.

2. NLP Processing: IBM Granite's NLP capabilities process the input, extracting relevant information and context.

3. Insight Generation: The application generates insights, predictions, or recommendations based on the user's input and IBM Granite's analysis.

4. Output: The application presents the output to the user, providing accurate and personalized healthcare insights.

7. API Documentation

1. Patient Chat: `/api/patient-chat`
    - POST: Send patient query and receive AI response
2. Disease Identifier: `/api/disease-identifier`
    - POST: Send symptoms and receive predicted disease
3. Treatment Plan Generator: `/api/treatment-plan`
    - POST: Send patient data and receive personalized treatment plan

API Request/Response Format

1. JSON: API requests and responses in JSON format
2. Authentication: API key or token-based authentication

API Documentation Tools

1. Swagger: API documentation and testing tool
2. API Blueprint: API documentation format

8. Authentication

1. MFA + Strong Identity

Use multi-factor authentication (something you know + something you have, e.g. SMS, hardware token) whenever a user accesses any PHI or sensitive query UI.

Use Single Sign On (SSO) where possible, integrated with hospital identity provider.

2. Zero Trust Architecture

Assume that no actor within the network is automatically trusted. Every access request should be checked.

Network segmentation: AI inference servers should be isolated; least privilinterfa

3. API Access Authentication

If other systems call Granite or your AI service via API (e.g. external systems, mobile apps), use OAuth tokens, signed JWTs, possibly mTLS (mutual TLS).

Rate-limiting, monitoring for unusual request patterns.

4. Prompt Sanitization / Whitelisting

Avoid letting users directly insert uncontrolled content into prompts that might override safety constraints. Maybe predefine templates, filter or clean inputs.

9. User interface

The User Interface (UI) for a Health AI Intelligent Healthcare Assistant using IBM Granite is the platform through which users interact with the system. Here's what it entails:

Key Features:

1. User-Friendly Design: An intuitive and easy-to-use interface that allows users to navigate and access various features.

2. Input Fields: Text input fields where users can ask health-related questions, provide symptoms, or describe their concerns.

3. Chatbot Interface: A conversational interface that simulates a human-like conversation, providing users with a natural way to interact with the system.

4. Results Display: A clear and concise display of results, including diagnoses, treatment options, and recommendations.

5. Personalized Insights: The UI can display personalized insights and recommendations based on the user's input and medical history.

Design Considerations:

1. Simple and Intuitive: The UI should be easy to use, even for those who are not tech-savvy.

2. Clear and Concise Language: The UI should use clear and concise language, avoiding technical jargon whenever possible.

3. Visual Hierarchy: The UI should use a clear visual hierarchy, making it easy for users to navigate and find the information they need.

4. Accessibility: The UI should be designed with accessibility in mind, ensuring that it can be used by users with disabilities.

Technologies Used:

1. Streamlit: A popular Python library for building data-driven web applications.

2. Front-end Frameworks: Frameworks like React, Angular, or Vue.js can be used to build a custom UI.

3. HTML/CSS: Standard web development technologies for building and styling the UI

10. Testing

1. Unit Testing

Test individual components (auth, patient search, AI query function).

2. Integration Testing

Check IBM Granite ↔ Hospital EHR ↔ UI flows.

Example: Doctor selects patient → Granite fetches + summarizes data.

3. System Testing

Full hospital workflow simulation (admit → treatment → discharge).

4. User Acceptance Testing (UAT)

Real clinicians try the system in a controlled setting.

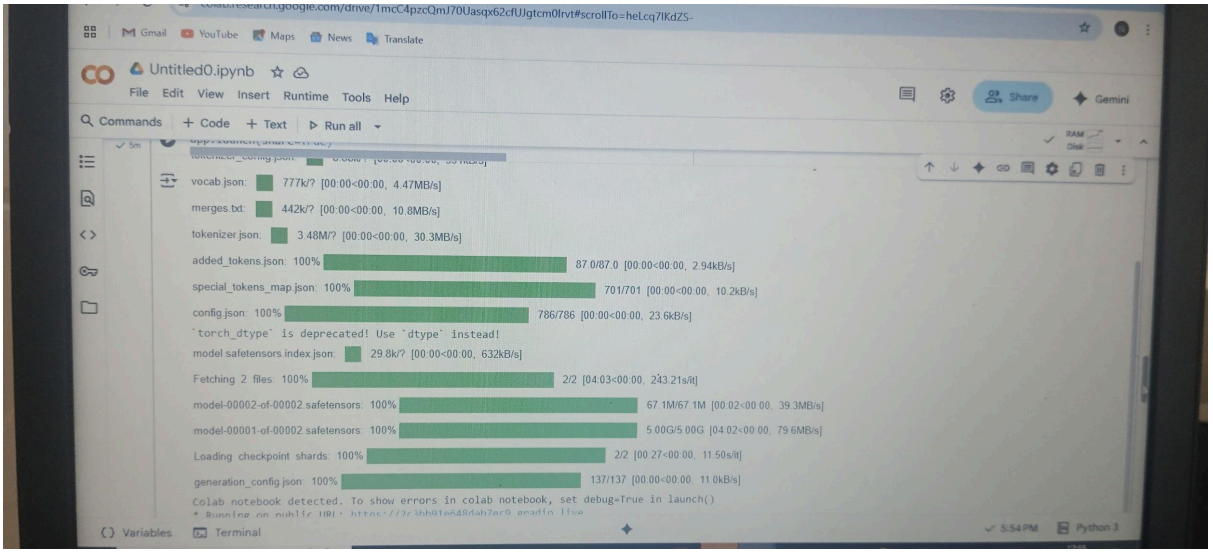
Collect feedback on accuracy, usability, trust.

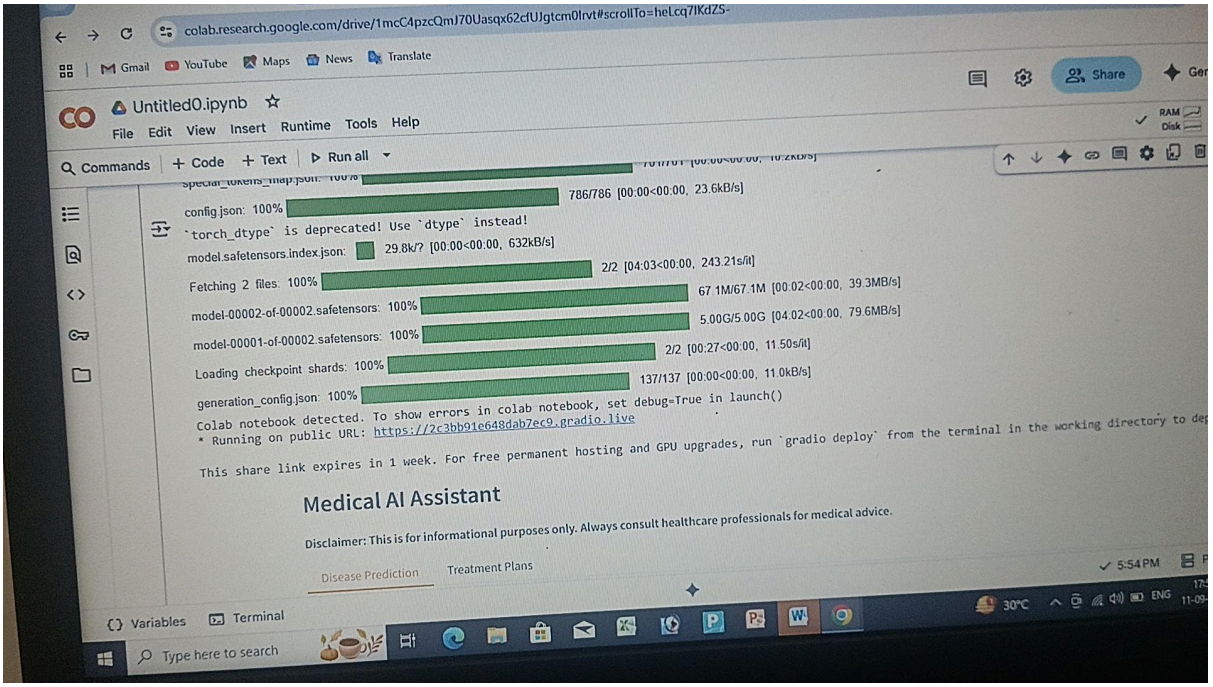
5. Regulatory Testing

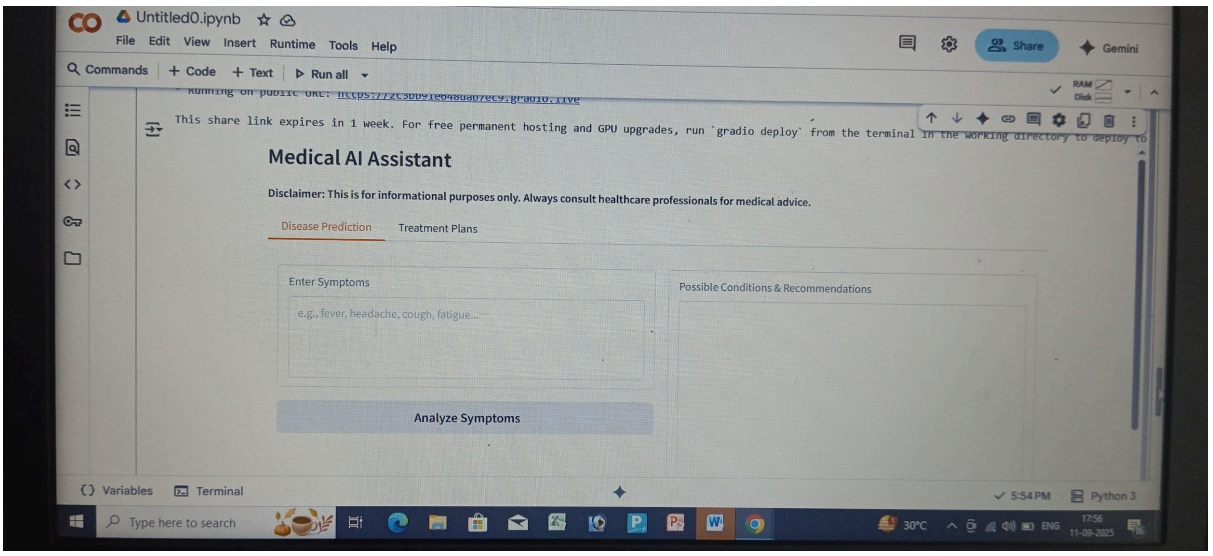
Ensure compliance with HIPAA, GDPR, CDSCO (India).

Document validation results for audits.

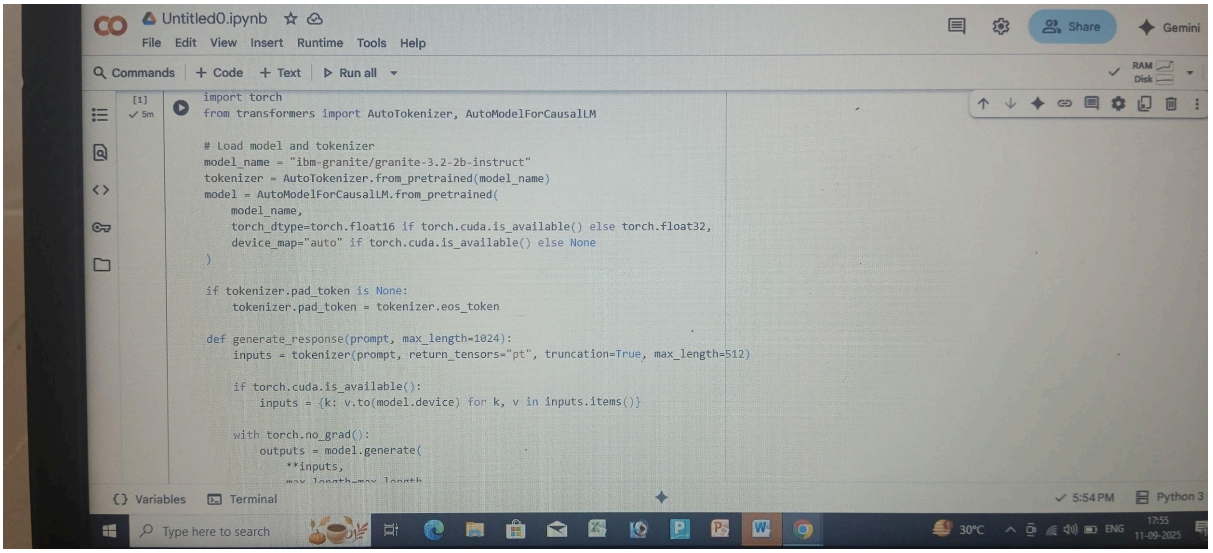
11. Screen shot

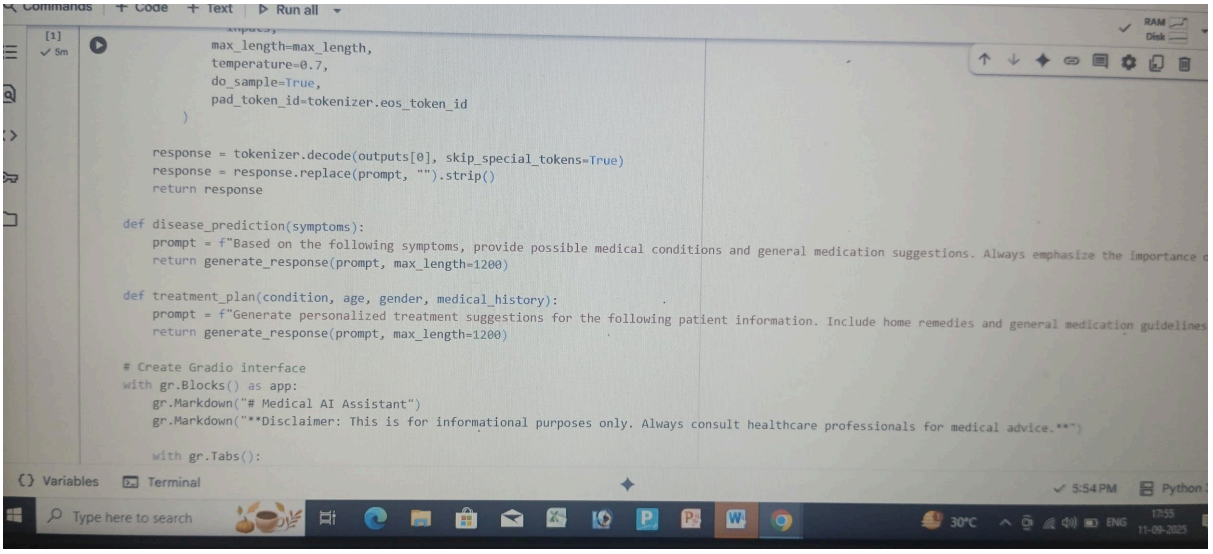




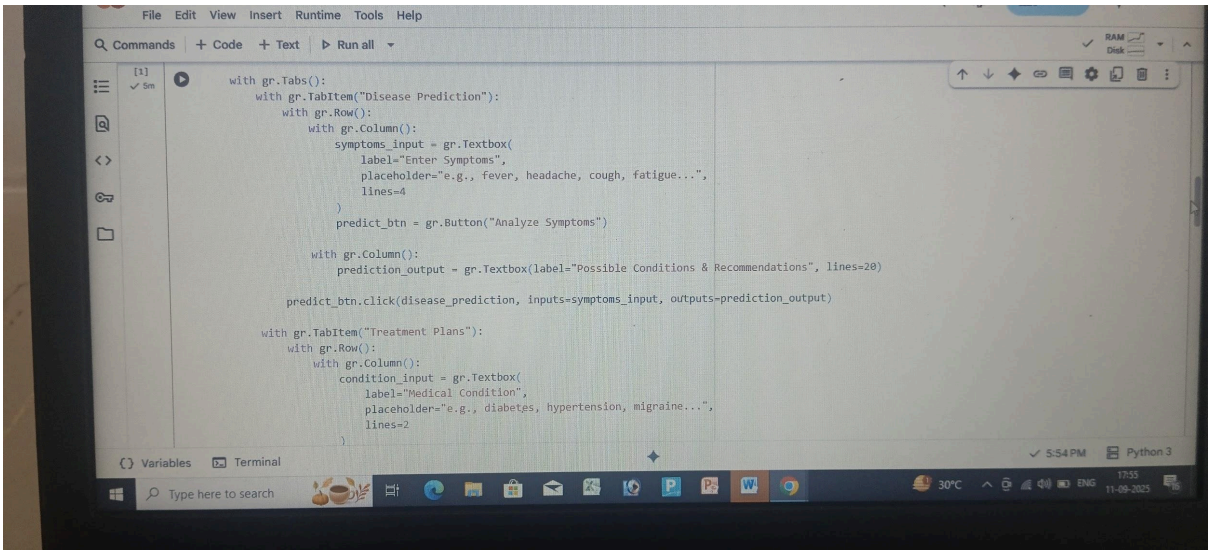

Untitled0.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands
+ Code
+ Text
Run all
This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the working directory to deploy to
Medical AI Assistant
Disclaimer: This is for informational purposes only. Always consult healthcare professionals for medical advice.
Disease Prediction
Treatment Plans
Enter Symptoms
e.g., fever, headache, cough, fatigue...
Possible Conditions & Recommendations
Analyze Symptoms
Variables
Terminal
5:54 PM
Python 3
Type here to search
30°C
ENG
17:56
11-09-2025

12.coding

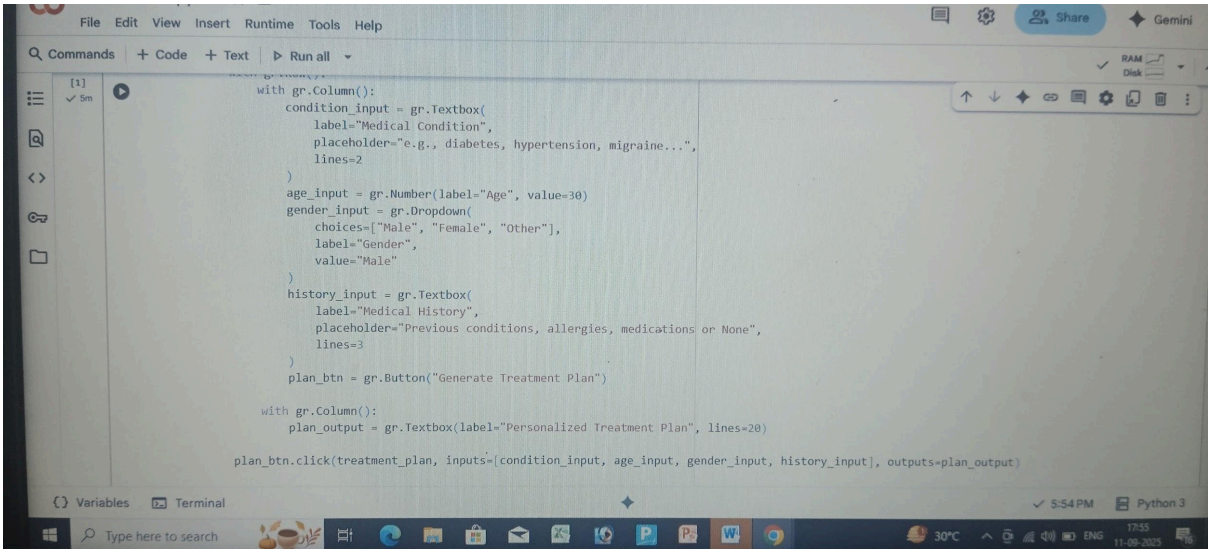

Untitled0.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands
+ Code
+ Text
Run all
[1]
5m
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM
# Load model and tokenizer
model_name = "ibm-granite/granite-3.2-2b-instruct"
tokenizer = AutoTokenizer.from_pretrained(model_name)
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,
Variables
Terminal
5:54 PM
Python 3
Type here to search
30°C
ENG
17:55
11-09-2025



```
                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 o
        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
        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.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 & 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
                    )
```



```
                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.Textbox(label="Personalized Treatment Plan", lines=20)

            plan_btn.click(treatment_plan, inputs=[condition_input, age_input, gender_input, history_input], outputs=plan_output)
```