HealthAI: Intelligent Healthcare Assistant Using IBM Granite

1.Introduction:

- **Project title :** HealthAI
- **Team member :** SHREE VISHAKHA K
- **Team member :** PRIYANKA K
- **Team member :** CAROLINA V N
- **Team member :** SHARULATHA MA

2. Project Overview:

Purpose:

The purpose of HealthAI is to democratize healthcare access by providing instant, AI-driven medical insights to patients anywhere, anytime. With IBM Granite LLM powering its backend, the system is capable of understanding medical prompts and generating contextual responses.

HealthAI focuses on two critical functions as implemented in the current version of the code:

1. Disease Prediction – Based on symptoms, it identifies possible conditions and provides general medication suggestions.

2. Treatment Plan Generator – Provides personalized treatment advice tailored to the patient's demographics and history.

Goals of the Project:

- Assist patients in understanding their symptoms before consulting a doctor.
- Support healthcare providers with AI-generated treatment ideas.
- Reduce dependency on long hospital visits for basic medical guidance.
- Enhance patient awareness of home remedies and preventive care.

Features:

1. **Conversational Interface**
    - Function: Natural health Q&A using plain language.
    - Example*:* Patient asks, “What could be the cause of persistent cough?” and receives a possible list of conditions.
2. **Disease Prediction**
    - Function: Symptom analysis leading to possible medical conditions.
    - Example: User enters “fever, cough, fatigue”, and AI suggests possible flu or viral infection with cautionary notes.
3. **Treatment Plan Generator**
    - Function: Creates personalized treatment suggestions based on patient details.
    - Example: Input: Diabetes, Age 50, Male, History of hypertension.

- Output: Lifestyle suggestions, dietary guidelines, and standard treatment options.

3. Architecture

The architecture of HealthAI is structured around simplicity, scalability, and modularity.

Frontend (Gradio):

- Provides an interactive UI with two tabs:
  - *Disease Prediction Tab* – accepts symptom input and generates predictions.
  - *Treatment Plan Tab* – accepts condition, age, gender, and history to create a treatment plan.
- Output displayed in clear, multi-line text boxes.
- Easy integration with additional features (feedback, reports, KPI monitoring).

Backend (Python):

- Written in Python with Hugging Face Transformers and PyTorch.
- Includes modular functions for disease prediction and treatment plan generation.

4. Setup Instructions

Prerequisites:

- Python 3.9+
- pip package manager
- Internet access (required to download IBM Granite model)
- GPU (optional, for faster response generation)

Installation Process:

1. Install dependencies:

   “pip install transformers accelerate gradio torch”

2. Clone repository and open app.py.
3. Run application:

   python app.py

4. Access via Gradio local server:
   - Local URL: http://127.0.0.1:7860
   - Public URL (Colab): Shared link generated by app.launch(share=True)

5. Folder Structure:

1. **health_ai.py**

- The main script of the project.
- Contains model initialization, disease prediction function, treatment plan function, and Gradio UI in one file.

2. **requirements.txt**

- Lists dependencies: torch, transformers, accelerate, and gradio.
- Ensures consistent environment setup.

3. **README.md**

- Provides an overview of the project.
- Explains installation steps, how to run the app, and usage guidelines.

6. Running the Application

Steps:

1. Start the script in Google Colab or local environment.
2. Load IBM Granite model automatically.
3. Select Disease Prediction Tab → Enter symptoms → Get conditions.
4. Select Treatment Plans Tab → Enter details → Get treatment suggestions.

Example Use Case:

- Input: Symptoms → “headache, nausea, blurred vision”
- Output: Possible conditions include migraine, dehydration, or high blood pressure. Please consult a doctor.

7. API Documentation

- POST /predict-disease – Accepts user-input symptoms in JSON format and returns a list of possible conditions along with recommendations.

- POST /treatment-plan – Accepts patient details such as condition, age, gender, and medical history, and responds with a personalized treatment plan.

- POST /upload-history – Allows uploading past health records to enhance the accuracy of recommendations.

8. Authentication

The current system is open for demo use. For real-world deployment:

- JWT Authentication for user access.
- Role-Based Access Control (RBAC):
    - Patient – symptom entry, treatment viewing.
    - Doctor – view reports, override AI output.
    - Admin – manage application and records.

9. User Interface

- Tabbed Navigation**:** Disease Prediction & Treatment Plan.
- Textbox Inputs**:** For symptoms, condition, and medical history.
- Multi-line Outputs**:** For clear AI responses.
- Accessibility**:** Designed for ease of use by non-technical patients.

10. Testing

Phases of Testing:

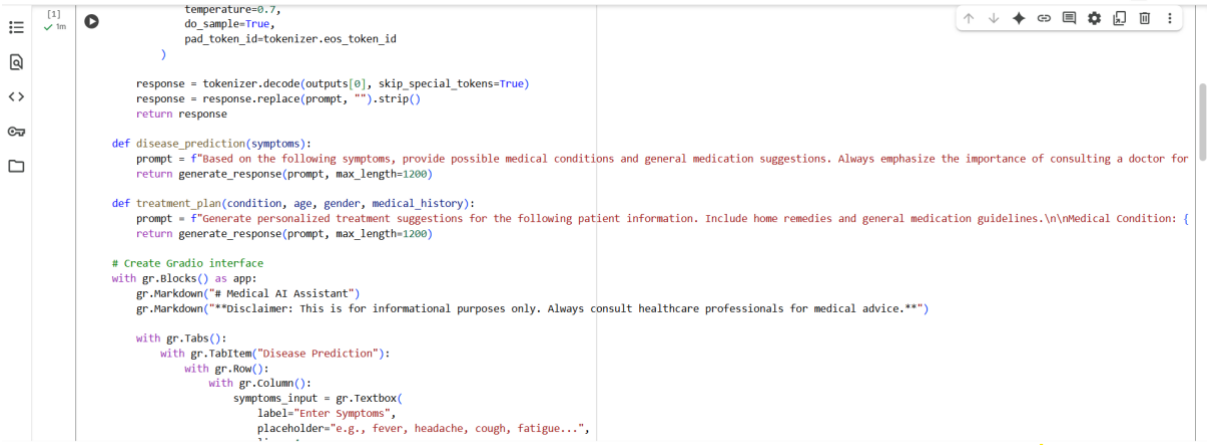
1. **Unit Testing:**
    - Checked response formatting, prompt handling.
2. **Manual Testing:**
    - Symptom entry tested with real-world conditions.
3. **API Testing (Future):**
    - Swagger/Postman for endpoint validation.

4. **Edge Case Handling:**
    - Empty input → AI returns “Please enter valid symptoms.”
    - Long inputs → Handled with tokenizer truncation.

11.Output

Coding

```
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,
```



```
            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 of consulting a doctor for
    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.\n\nMedical Condition: {
    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.TabItem("Disease Prediction"):
            with gr.Row():
                with gr.Column():
                    symptoms_input = gr.Textbox(
                        label="Enter Symptoms",
                        placeholder="e.g., fever, headache, cough, fatigue...",
```

```
                )
                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
                )
                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")
```

```
                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)

app.launch(share=True)
```

```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in your Google Colab and restart your ses
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
`torch_dtype` is deprecated! Use `dtype` instead!
```

Commands + Code + Text ▷ Run all

RAM Disk

```
Fetching 2 files: 100% 2/2 [04:04<00:00, 244.84s/it]
model-00001-of-00002.safetensors: 100% 5.00G/5.00G [04:03<00:00, 24.4MB/s]
model-00002-of-00002.safetensors: 100% 67.1M/67.1M [00:01<00:00, 45.1MB/s]
Loading checkpoint shards: 100% 2/2 [00:32<00:00, 13.45s/it]
generation_config.json: 100% 137/137 [00:00<00:00, 13.0kB/s]
Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://9c5d388d6874dbe8ca.gradio.live

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 Hugging Face Spaces (
```

Medical AI Assistant

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

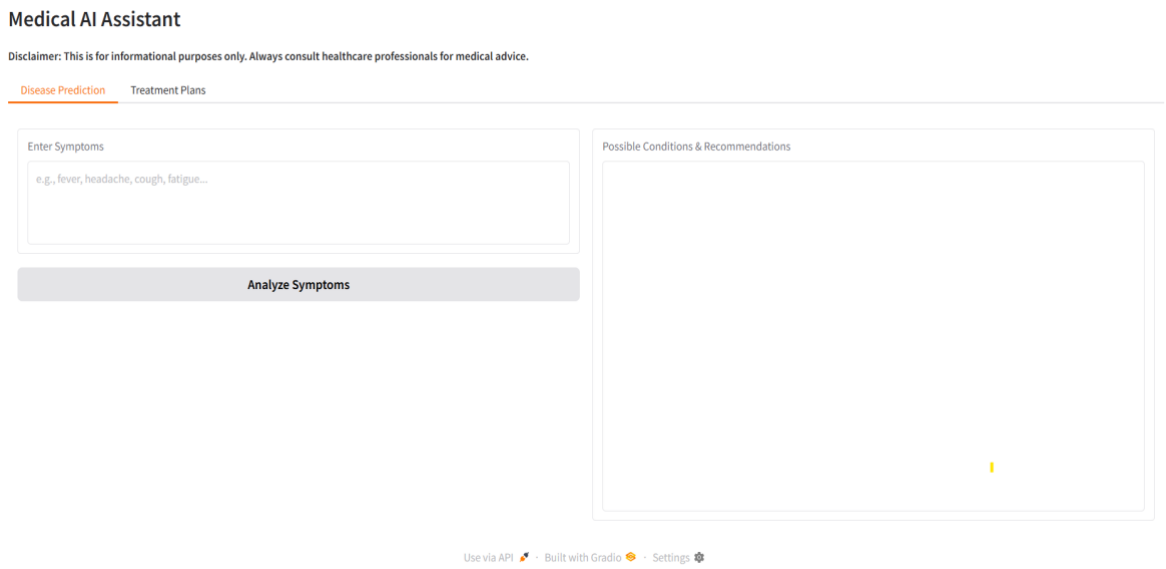
Disease Prediction | Treatment Plans

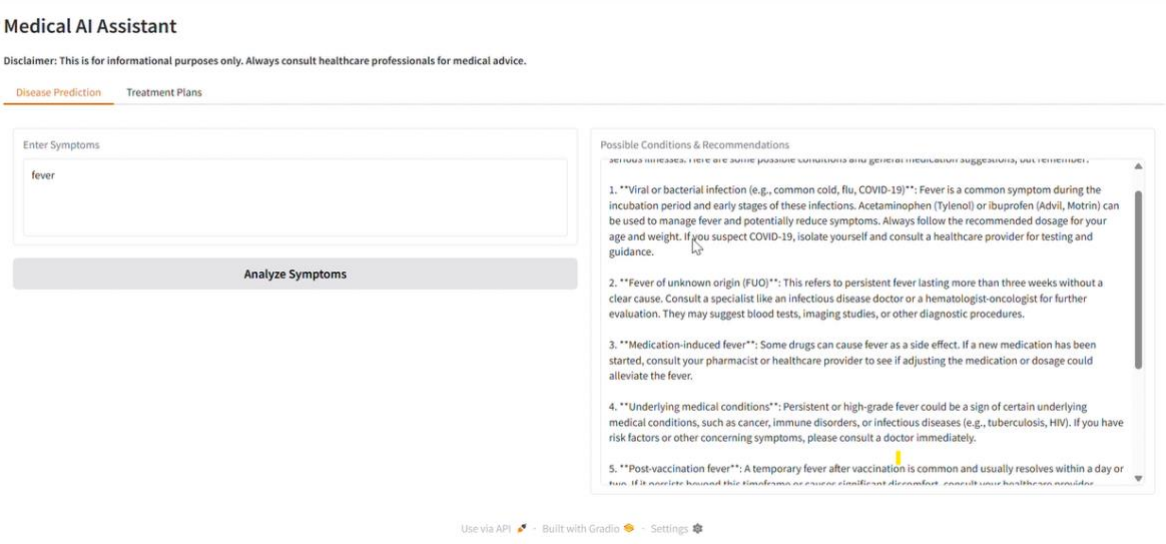
Enter Symptoms

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

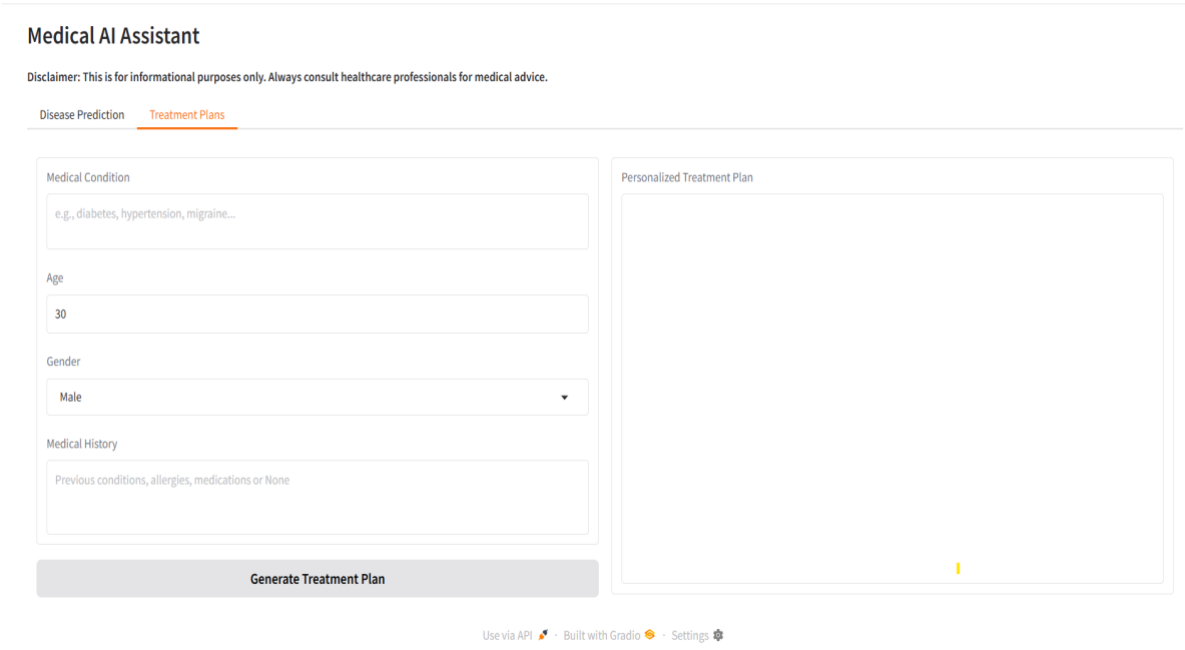
Possible Conditions & Recommendations

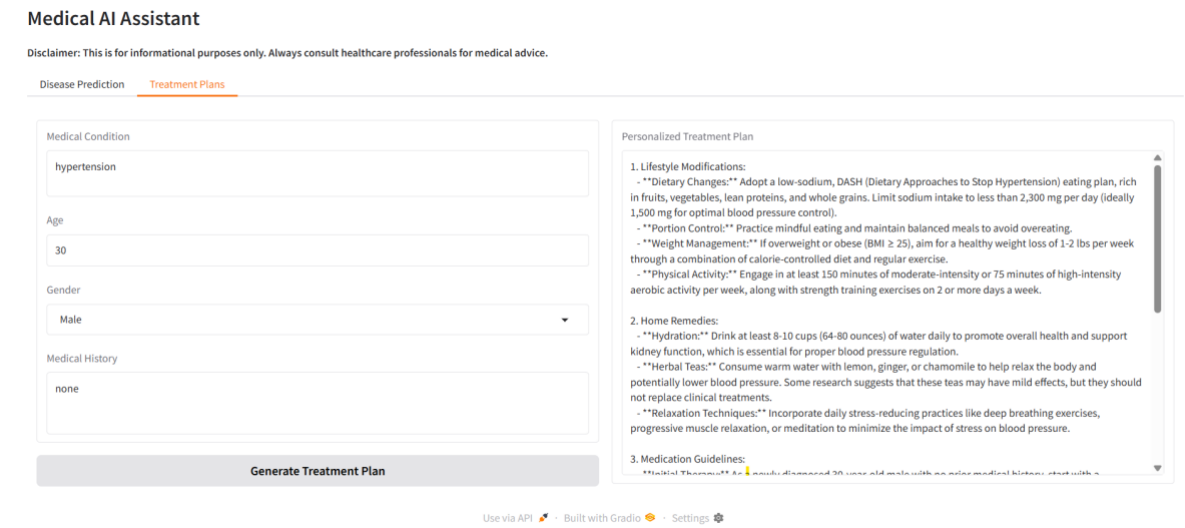
Disease prediction





Treatment plan





12. Known Issues

- Requires stable internet for IBM Granite model.
- Outputs are probabilistic and may vary.
- Does not replace real medical consultation.
- Limited to two major features (prediction & treatment) in current version.

13. Future Enhancements

The current version of HealthAI focuses on symptom-based disease prediction and personalized treatment suggestions. Future improvements based on the existing code framework may include:

- **Medical Report Summarization** – Allow users to upload PDFs or text-based medical reports, which the model can summarize into simple insights.
- **Extended Treatment Plans** – Incorporate additional patient details (diet, lifestyle habits) to generate more comprehensive treatment guidance.
- **Feedback Collection** – Add a feedback tab where patients can share responses, enabling continuous refinement of AI suggestions.
- **Health Data Forecasting** – Introduce forecasting for patient health metrics (e.g., recurring symptoms, predicted recovery timeline).
- **Multilingual Responses** – Enable the model to respond in multiple languages to increase accessibility for non-English-speaking users.