Health AI: Intelligent Healthcare Assistant

|  |  |
| --- | --- |
| DATE |  |
| TEAM ID | NM2025TMID00651 |
| PROJECT NAME | Health AI : Intelligent Healthcare Assistant |
| MAXIMUM MARKS |  |

**Introduction**

The **Health AI: Intelligent Healthcare Assistant** is a conversational AI tool built to assist in disease prediction and personalized treatment plans. It uses advanced natural language processing (NLP) models to analyze user-inputted symptoms and medical information, and generate possible medical conditions along with treatment suggestions. The tool is designed to be an informational assistant, providing recommendations while encouraging users to consult healthcare professionals for proper diagnosis and treatment.

**Key Features**

* **Disease Prediction**: Analyzes user-reported symptoms and provides possible medical conditions.
* **Treatment Plan Generation**: Generates personalized treatment plans based on medical conditions, age, gender, and medical history.
* **Gradio Interface**: The app is integrated with a user-friendly Gradio interface for easy interaction.

**Disclaimer**

The suggestions provided by this assistant are for informational purposes only and should not be considered as professional medical advice. Always consult a healthcare professional for accurate diagnosis and treatment.

**Getting Started**

**Prerequisites**

* **Python 3.x** (recommended version: 3.8 or higher)
* **PyTorch** (with CUDA support for GPU acceleration, optional but recommended)
* **Gradio** (for building the web interface)
* **Transformers library** (for loading the AI model and tokenizer)

**Installation Steps**

To get started with the project, ensure that all dependencies are installed. You can use the following commands to install the required libraries:

pip install torch gradio transformers

**How It Works**

**Model Loading**

The model used in the assistant is a **Causal Language Model** from the ibm-granite family. This model is fine-tuned to understand medical language and generate appropriate responses. The model is loaded using the **Transformers** library by Hugging Face.

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 **AutoTokenizer** and **AutoModelForCausalLM** classes are used to load the tokenizer and model. The model is set to use GPU if available, otherwise, it defaults to CPU.

**Response Generation**

The generate\_response() function takes the user’s prompt and processes it through the model to generate a response. The model is set to **temperature = 0.7**, which controls the randomness of the output, and the **max\_length** parameter is used to ensure the response does not exceed a certain length.

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

**Disease Prediction**

The **disease\_prediction()** function takes the user-inputted symptoms and generates a list of possible medical conditions. It then provides general recommendations on managing these conditions.

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.\n\nSymptoms: {symptoms}\n\nPossible conditions and recommendations:\n\n\*\*IMPORTANT: This is for informational purposes only. Please consult a healthcare professional for proper diagnosis and treatment.\*\*\n\nAnalysis:"

return generate\_response(prompt, max\_length=1200)

**Inputs:**

* **Symptoms**: A string describing the symptoms (e.g., fever, headache, fatigue).

**Output:**

* A list of possible medical conditions and recommendations for further steps.

**Treatment Plan**

The **treatment\_plan()** function generates a personalized treatment plan for a given medical condition, based on the user’s age, gender, and medical history. The plan includes medication guidelines and home remedies.

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: {condition}\nAge: {age}\nGender: {gender}\nMedical History: {medical\_history}\n\nPersonalized treatment plan including home remedies and medication guidelines:\n\n\*\*IMPORTANT: This is for informational purposes only. Please consult a healthcare professional for proper treatment.\*\*\n\nTreatment Plan:"

return generate\_response(prompt, max\_length=1200)

**Inputs:**

* **Medical Condition**: The name of the condition (e.g., diabetes, hypertension).
* **Age**: The age of the user.
* **Gender**: The gender of the user.
* **Medical History**: A brief description of the user's medical history (e.g., allergies, chronic conditions).

**Output:**

* A personalized treatment plan with recommendations for medication and lifestyle changes.

**Gradio Interface**

The app uses the **Gradio** library to create a user-friendly interface. Users can interact with the system through a simple web interface, where they can input their symptoms or medical information, and receive disease predictions and treatment plans.

**Tabs and Inputs**

The Gradio interface consists of two main tabs:

1. **Disease Prediction Tab**:
   * Users can input their symptoms and receive predictions for possible conditions and treatment recommendations.
2. **Treatment Plan Tab**:
   * Users can input their medical condition, age, gender, and medical history to receive a personalized treatment plan.

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...",

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

)

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)

**Conclusion**

This **Health AI Assistant** serves as an informative tool for disease prediction and treatment suggestions. While it uses advanced AI models to provide helpful insights based on input data, it is essential to note that this tool should **not** replace professional medical advice. Always consult a healthcare professional for proper diagnosis and treatment.