!pip install transformers torch gradio -q

import gradio as gr

import torch

from transformers import AutoTokenizer, AutoModelForCausalLM

import datetime

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

# Load Model

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

# This part of the code loads the AI model for generating responses.

# It uses a pre-trained model from IBM for causal language modeling.

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

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

# AI Response Functions

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

def generate\_response(prompt, max\_length=1024):

"""Generates a text response from the loaded model based on the given prompt."""

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

def disease\_prediction(symptoms, history):

"""

Predicts possible medical conditions and adds the query to the history.

"""

prompt = f"""Based on the following symptoms, provide possible medical conditions

and general medication suggestions. Always emphasize the importance of consulting a doctor.

Symptoms: {symptoms}

\*IMPORTANT: This is for informational purposes only. Please consult a healthcare professional.\*

Analysis:"""

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

# Update history with the new query and response

new\_entry = {

"timestamp": datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S"),

"type": "Disease Prediction",

"query": f"Symptoms: {symptoms}",

"response": response

}

history.append(new\_entry)

return response, history

def treatment\_plan(condition, age, gender, medical\_history, history):

"""

Generates a personalized treatment plan and adds the query to the history.

"""

prompt = f"""Generate personalized treatment suggestions for the following patient:

Medical Condition: {condition}

Age: {age}

Gender: {gender}

Medical History: {medical\_history}

\*IMPORTANT: This is for informational purposes only. Please consult a healthcare professional.\*

Treatment Plan:"""

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

# Update history with the new query and response

new\_entry = {

"timestamp": datetime.datetime.now().strftime("%Y-%m-%d %H:%M:%S"),

"type": "Treatment Plan",

"query": f"Condition: {condition}, Age: {age}, Gender: {gender}, History: {medical\_history}",

"response": response

}

history.append(new\_entry)

return response, history

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

# User Authentication System

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

users = {

"admin": "1234" # your account by default

}

def login(username, password):

"""Authenticates a user and shows the main app if credentials are correct."""

if username in users and users[username] == password:

return gr.update(visible=False), gr.update(visible=False), gr.update(visible=True), f"✅ Welcome, {username}!"

else:

return gr.update(visible=True), gr.update(visible=False), gr.update(visible=False), "❌ Invalid credentials. Try again."

def signup(new\_username, new\_password, confirm\_password):

"""Handles new user registration."""

if new\_username in users:

return "⚠️ Username already exists. Try another."

if new\_password != confirm\_password:

return "⚠️ Passwords do not match. Try again."

if len(new\_username.strip()) == 0 or len(new\_password.strip()) == 0:

return "⚠️ Username/Password cannot be empty."

users[new\_username] = new\_password

return f"✅ Account created for \*\*{new\_username}\*\*! You can now log in."

def clear\_history(history):

"""Clears the in-memory chat history."""

return [], "" # Return an empty list for history and an empty string for the display

def format\_history\_display(history):

"""Formats the history list into a readable markdown string."""

if not history:

return "No history available."

history\_string = ""

for entry in history:

history\_string += f"\*\*Timestamp:\*\* {entry['timestamp']}\n"

history\_string += f"\*\*Type:\*\* {entry['type']}\n"

history\_string += f"\*\*Query:\*\* {entry['query']}\n"

history\_string += f"\*\*Response:\*\* {entry['response']}\n"

history\_string += "---\n"

return history\_string

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

# Build Gradio Interface

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

with gr.Blocks(title="HEALTH AI ASSISTANT") as app:

gr.Markdown("## 🔐 HEALTH AI ASSISTANT WITH LOGIN & SIGNUP USING GENERATIVE AI AND IBM GRANITE MODEL”)

# State to hold the chat history

history\_state = gr.State([])

# Login Section

with gr.Group(visible=True) as login\_group:

gr.Markdown("### Please Login")

username\_input = gr.Textbox(label="Username")

password\_input = gr.Textbox(label="Password", type="password")

login\_btn = gr.Button("Login")

login\_status = gr.Markdown("")

signup\_link = gr.Button("New user? Sign up here")

# Signup Section

with gr.Group(visible=False) as signup\_group:

gr.Markdown("### Create a New Account")

new\_username = gr.Textbox(label="New Username")

new\_password = gr.Textbox(label="New Password", type="password")

confirm\_password = gr.Textbox(label="Confirm Password", type="password")

signup\_btn = gr.Button("Sign Up")

signup\_status = gr.Markdown("")

back\_to\_login = gr.Button("⬅ Back to Login")

# Main App Section (hidden until login)

with gr.Group(visible=False) as main\_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():

# Home Page

with gr.TabItem("Home"):

gr.Markdown(

"""

### Welcome to the Medical AI Assistant!

This application provides you with informational content about potential medical conditions and treatment plans. It is designed to be a starting point for gathering information, not a substitute for professional medical advice.

---

### How to Use This Tool

\* \*\*Disease Prediction:\*\* Navigate to the "Disease Prediction" tab and enter a list of symptoms you are experiencing. The AI will provide a list of possible conditions and general recommendations.

\* \*\*Treatment Plans:\*\* In the "Treatment Plans" tab, you can input a medical condition along with personal details like age, gender, and medical history. The AI will generate a personalized treatment plan for informational purposes.

\* \*\*History:\*\* The "History" tab keeps a record of all your queries and the AI's responses, allowing you to review past interactions.

---

### Important Disclaimer

The information and suggestions provided by this AI assistant are based on a large language model and should be considered for \*\*informational purposes only\*\*. They are not intended to be a substitute for a professional medical diagnosis or treatment. \*\*Always consult a qualified healthcare provider for medical advice.\*\*

"""

)

# 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.Textbox(label="Possible Conditions & Recommendations", lines=20)

predict\_btn.click(

disease\_prediction,

inputs=[symptoms\_input, history\_state],

outputs=[prediction\_output, history\_state]

).then(

format\_history\_display,

inputs=history\_state,

outputs=gr.Textbox(label="History Display", lines=10) # Dummy output for history tab refresh

)

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

plan\_btn.click(

treatment\_plan,

inputs=[condition\_input, age\_input, gender\_input, history\_input, history\_state],

outputs=[plan\_output, history\_state]

).then(

format\_history\_display,

inputs=history\_state,

outputs=gr.Textbox(label="History Display", lines=10) # Dummy output for history tab refresh

)

# History Tab

with gr.TabItem("History"):

with gr.Column():

gr.Markdown("### Your Query History")

history\_display = gr.Textbox(label="Past Queries and Responses", lines=20, interactive=False)

clear\_history\_btn = gr.Button("Clear History")

# Initial display of history when the page loads

app.load(format\_history\_display, inputs=history\_state, outputs=history\_display)

clear\_history\_btn.click(

clear\_history,

inputs=history\_state,

outputs=[history\_state, history\_display]

)

# About Tab

with gr.TabItem("About"):

gr.Markdown(

"""

### About This Application

This AI-powered medical assistant is designed to provide you with general, informational content about possible medical conditions and treatment options.

### Technology Stack

This application is built using several powerful open-source libraries:

\* \*\*Gradio:\*\* The entire user interface is built with Gradio, which allows us to create interactive web applications from Python scripts with minimal effort. This is what you see and interact with in your browser.

\* \*\*Hugging Face Transformers:\*\* This library provides access to the pre-trained `ibm-granite/granite-3.2-2b-instruct` model, which is the core intelligence behind the assistant. The library handles the complex tasks of loading the model and tokenizer, and generating text.

\* \*\*PyTorch:\*\* The underlying deep learning framework that powers the model, handling all the computational heavy lifting for text generation.

### Purpose and Limitations

The primary purpose of this tool is to demonstrate the potential of large language models in a real-world application. It can be a helpful starting point for general information and a way to organize your thoughts before consulting a professional.

\*\*Crucial Caveat:\*\* The information provided by this assistant is not a substitute for professional medical advice, diagnosis, or treatment. It is a generative model and its responses are based on patterns in its training data, not on the complex, nuanced understanding of a medical professional. Always consult a qualified healthcare provider for any health-related concerns.

"""

)

# Connect Login

login\_btn.click(

login,

inputs=[username\_input, password\_input],

outputs=[login\_group, signup\_group, main\_app, login\_status]

)

# Connect Signup

signup\_btn.click(

signup,

inputs=[new\_username, new\_password, confirm\_password],

outputs=[signup\_status]

)

# Toggle between Login & Signup

signup\_link.click(

lambda: (gr.update(visible=False), gr.update(visible=True)),

inputs=[],

outputs=[login\_group, signup\_group]

)

back\_to\_login.click(

lambda: (gr.update(visible=True), gr.update(visible=False)),

inputs=[],

outputs=[login\_group, signup\_group]

)

app.launch()