Medical AI Assistant – Project Documentation

1. Project Overview

The Medical AI Assistant is a web application that uses a Large Language Model (LLM) to:

Suggest possible medical conditions based on user-entered symptoms.

Provide general treatment guidance and home-remedy suggestions for a given condition.

> Important: This tool is for informational purposes only and is not a substitute for professional medical advice, diagnosis, or treatment. The app displays a disclaimer prominently to remind users to consult a healthcare professional.

---

2. Key Features

Disease Prediction Tab

Users enter symptoms, and the app outputs likely conditions with general advice.

Treatment Plan Tab

Users provide a medical condition, age, gender, and medical history to receive a basic, generic treatment plan (home remedies, medication guidelines).

Interactive Web Interface

Built with Gradio, providing an easy, no-code interface for end users.

---

3. Technology Stack

Component	Description
Python	Primary programming language
Gradio	Builds the interactive web UI
Transformers (Hugging Face)	Loads and interacts with the Granite language model
PyTorch	Backend deep-learning framework for running the model
Model	ibm-granite/granite-3.2-2b-instruct – open-source LLM

---

4. Code Walkthrough

4.1 Importing Dependencies

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

gradio: For the web UI.

torch: Handles GPU/CPU execution.

transformers: Provides pretrained tokenizer and model loading.

4.2 Loading the 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
)
```

Downloads the Granite-3.2-2B Instruct model and tokenizer.

Automatically uses GPU (float16) if available, else falls back to CPU.

if tokenizer.pad_token is None:

    tokenizer.pad_token = tokenizer.eos_token

Ensures a padding token is set, preventing errors during generation.

4.3 Core Generation Function

def generate_response(prompt, max_length=1024):

    inputs = tokenizer(prompt, return_tensors="pt", truncation=True, max_length=512)

    ...

    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

Encodes the user’s prompt and feeds it to the model.

Uses temperature=0.7 for creative but coherent outputs.

Decodes the generated tokens and cleans them.

4.4 Application Logic

Disease Prediction

```
def disease_prediction(symptoms):
    prompt = f"Based on the following symptoms... {symptoms} ..."
    return generate_response(prompt, max_length=1200)
```

Builds a prompt that requests possible conditions and general medication suggestions with a safety disclaimer.

Treatment Plan

```
def treatment_plan(condition, age, gender, medical_history):
    prompt = f"Generate personalized treatment suggestions... {condition}..."
    return generate_response(prompt, max_length=1200)
```

Takes patient details and returns a generic treatment plan.

4.5 Gradio Interface

```
with gr.Blocks() as app:
    gr.Markdown("# Medical AI Assistant")
    ...
    with gr.Tabs():
        with gr.TabItem("Disease Prediction"):
            ...
            predict_btn.click(disease_prediction, inputs=symptoms_input, outputs=prediction_output)
        with gr.TabItem("Treatment Plans"):
            ...
            plan_btn.click(treatment_plan, inputs=[condition_input, age_input, gender_input,
history_input], outputs=plan_output)
```

Blocks layout organizes the UI.

Two tabs: Disease Prediction and Treatment Plans.

click events bind buttons to the respective functions.

Finally:

app.launch(share=True)

Launches a public, shareable Gradio web app.

---

5. Setup Instructions

1. Install Requirements

pip install gradio torch transformers

2. Save the Script

e.g., medical_ai_app.py

3. Run the App

python medical_ai_app.py

A local URL (and a temporary public URL if share=True) will appear in the terminal.

---

6. Usage Guidelines & Limitations

Provide clear, concise symptoms or condition details for best results.

The model provides generalized suggestions, not verified medical facts.

Internet connection is required to download the model the first time.

GPU is recommended for faster responses.

---

7. Future Improvements

Add a database of verified medical information for cross-checking outputs.

Include multi-language support.

Provide optional PDF export of the generated advice.

Disclaimer

This application is not a medical device. It is intended for educational and informational purposes only. Always seek the advice of qualified health providers with any questions regarding a medical condition.

End of Document