Health AI

Project Documentation

1. Introduction

Project Title: Health AI with IBM

Team Leader: MAGIVARDHINI R

Team Members: KAMALI M

Team Members: ARCHANA J

Team Members: ABINAYA B

Team Members: AKALYA R

---

2. Project Overview

Purpose:
The Medical AI Assistant is designed to provide quick and informative health-related suggestions based on user-provided symptoms and medical conditions. It emphasizes the importance of consulting a doctor while offering general recommendations, home remedies, and guidance.

Features:

Disease Prediction: Suggests possible conditions based on symptoms.

Treatment Plans: Provides home remedies and general medication guidelines.

Patient Personalization: Considers age, gender, and medical history.

Conversational Interface: Easy-to-use chatbot powered by Gradio.

Disclaimer Integration: Ensures users understand this is not a substitute for professional medical advice.

---

3. Architecture

Frontend (Gradio):
Provides a clean interface with tabs for "Disease Prediction" and "Treatment Plans".

Backend (Python + Hugging Face Transformers):
Uses IBM Granite LLM models for generating responses.

Model Integration:

Model Name: ibm-granite/granite-3.2-2b-instruct

Loaded using Hugging Face Transformers library.

Hardware Support:

Runs on CPU or GPU (optimized with CUDA if available).

---

4. Setup Instructions

Prerequisites

Python 3.8+

Google Colab / Jupyter Notebook / Local Python environment

Hugging Face Transformers

Gradio

Steps

1. Install dependencies:

pip install gradio torch transformers

2. Load IBM Granite model from Hugging Face.

3. Run the Gradio app code.

4. Launch the app and open the link provided in the console.

---

5. Folder Structure

```
medical_ai/
│
├── app/                # Backend logic
├── ui/                 # Gradio UI files
├── medical_ai.py       # Main application file
├── model_loader.py     # Handles IBM Granite model integration
└── requirements.txt    # Dependencies
```

---

6. Running the Application

1. Run the Python script in Colab or your local environment.

2. The Gradio app launches with two tabs:

Disease Prediction Tab: Enter symptoms → Get possible conditions.

Treatment Plan Tab: Enter condition + age + gender + history → Get personalized plan.

3. A shareable link will be generated for testing.

---

7. API Documentation (Internal Functions)

disease_prediction(symptoms)

Input: Symptoms string

Output: Possible conditions + recommendations

treatment_plan(condition, age, gender, medical_history)

Input: Condition + Patient info

Output: Personalized treatment plan with home remedies + guidelines

---

8. Authentication

Current demo runs in open mode.

Future deployments can integrate:

API Keys

OAuth2 authentication

Role-based access for doctors/patients

---

9. User Interface

Tabs:

Disease Prediction

Treatment Plans

Inputs: Textboxes, Number input, Dropdowns

Outputs: Text responses with medical recommendations

Disclaimer: Displayed at the top to ensure users consult professionals

---

10. Testing

Unit Testing:
Checked response generation with sample inputs.

Edge Case Handling:

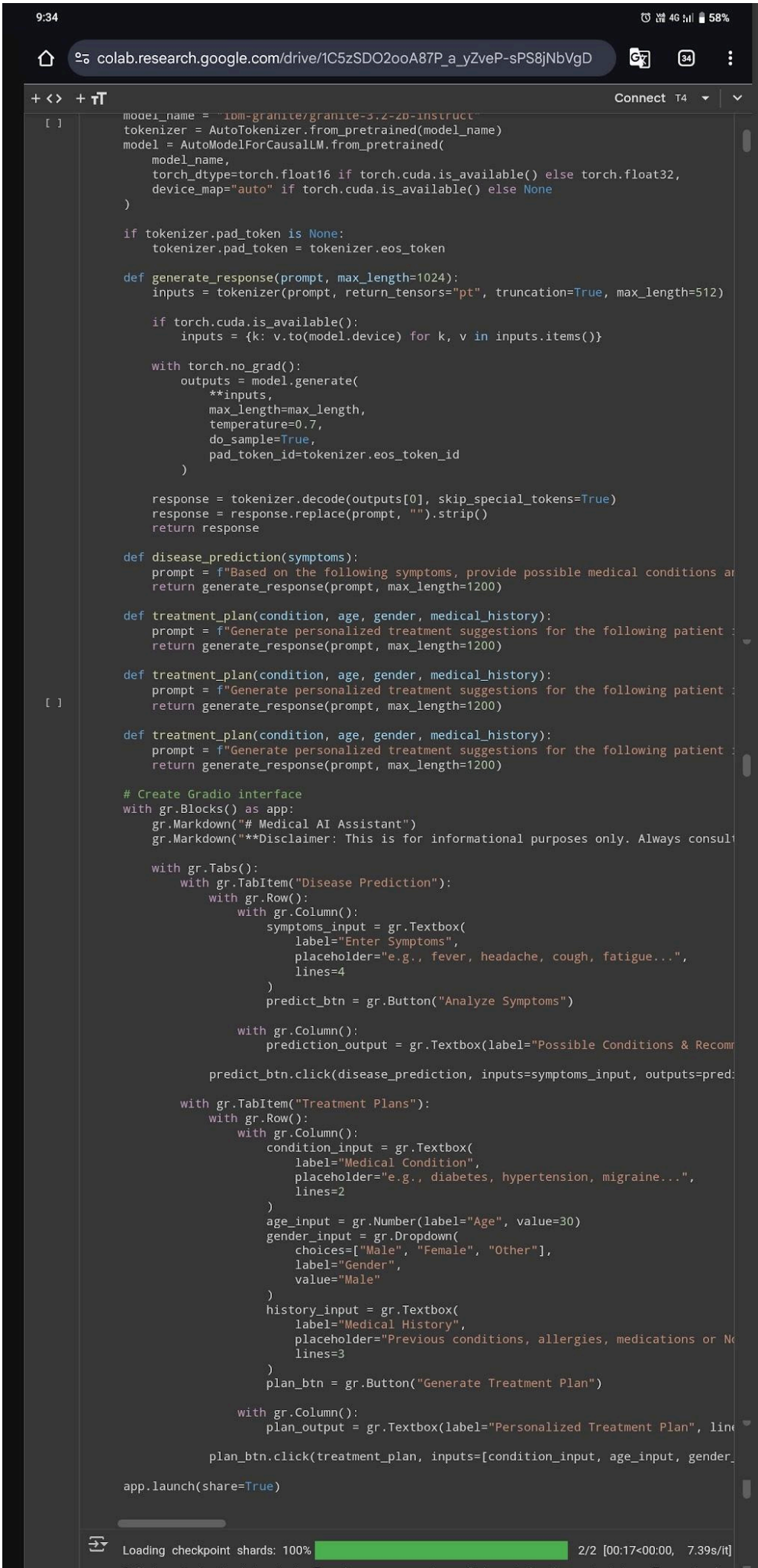
Empty input fields

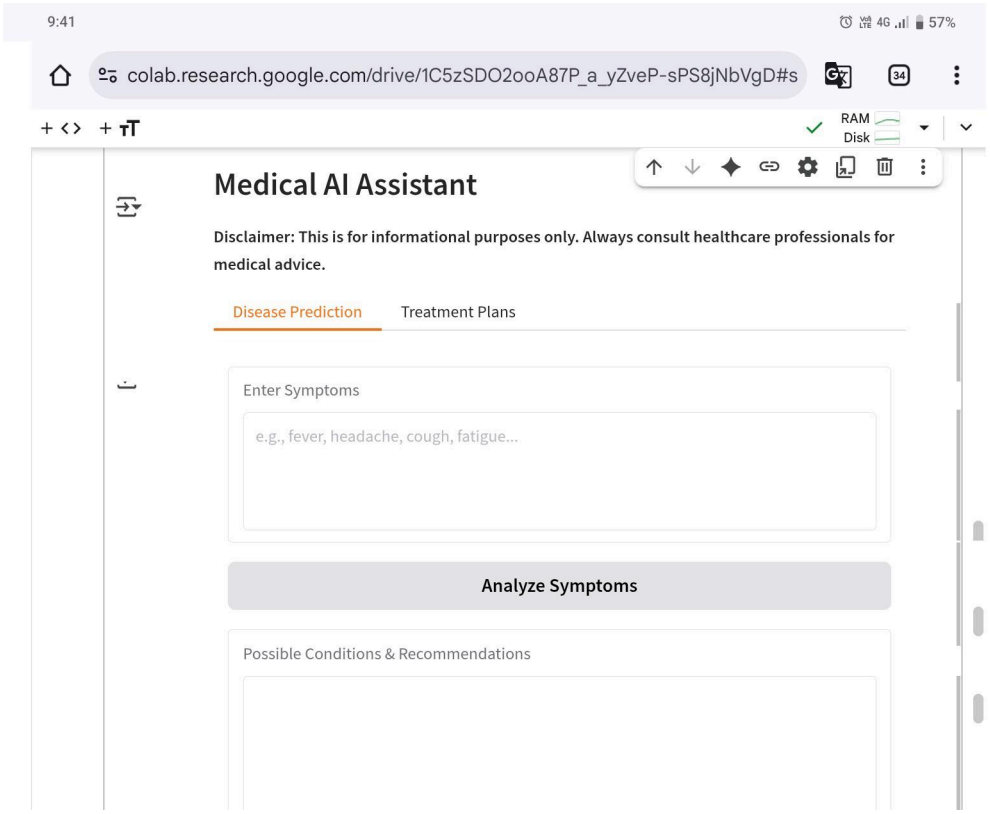
Non-medical text

Long symptom lists

Manual Testing:
Tested interface usability with Gradio demo links.

11.Screenshots


9:34
58%
colab.research.google.com/drive/1C5zSDO2ooA87P_a_yZveP-sPS8jNbVgD
Connect T4
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,
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 ar
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 :
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 :
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 :
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
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 & Recomm
predict_btn.click(disease_prediction, inputs=symptoms_input, outputs=predi
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 No
lines=3
)
plan_btn = gr.Button("Generate Treatment Plan")
with gr.Column():
plan_output = gr.Textbox(label="Personalized Treatment Plan", line
plan_btn.click(treatment_plan, inputs=[condition_input, age_input, gender_
app.launch(share=True)
Loading checkpoint shards: 100%
2/2 [00:17<00:00, 7.39s/it]



12. Known Issues

LLM responses may sometimes be generic.

Requires internet to fetch model from Hugging Face.

Not a substitute for certified medical systems.

---

13. Future Enhancements

Multi-language support (Tamil, Hindi, etc.).

Integration with medical databases for verified information.

Cloud deployment for large-scale usage.

Add voice input and speech output.

Dashboard for doctors to review aggregated patient queries.
