Medical AI Assistant Project Documentation

Team ID : NM2025TMID00796

Team Members:

*SWEETHA E

*PRIYANGA D

*PRITHIGA R

1. Introduction

Project Title: Health AI – Intelligent Healthcare Assistant Using IBM Granite

This project is a collaborative effort designed to explore how Artificial Intelligence can support healthcare by providing intelligent recommendations. The assistant is powered by IBM Granite and aims to simplify medical consultations by offering predictive analysis based on user-provided symptoms.

2. Project Overview

The Medical AI Assistant provides a supportive digital healthcare experience. It takes symptoms entered by the user, analyzes them with the help of IBM Granite, and generates possible health conditions. Additionally, it offers general treatment guidelines and personalized suggestions based on age, gender, and medical history.

The primary aim is not to replace doctors but to serve as a first-level assistant that helps individuals understand their symptoms better and motivates them to seek professional help.

3. Architecture

The architecture of the system is modular and divided into different components:

- Frontend (Gradio): Provides an interactive web interface for disease prediction and treatment planning.
- Backend (Python + Transformers): Handles AI model loading, prompt engineering, and response generation.
- LLM Integration (IBM Granite 3.2 2B): Processes user queries and generates meaningful results.
- Hosting: The application can be run locally or shared using Gradio's hosting feature.

4. Setup Instructions

Prerequisites:

- Python 3.8 or above
- pip package manager
- Virtual environment
- IBM Watsonx API key

Installation Process:

1. Clone the repository or copy the project files.
2. Create and activate a virtual environment.
3. Install required dependencies using requirements.txt.
4. Run the script using: python app.py
5. Access the Gradio app through the provided local or share URL.

5. Folder Structure

The folder structure is organized as follows:

```
project-root/
├── app/ – FastAPI backend logic including chat, prediction, and analytics modules
├── ui/ – Streamlit frontend components for dashboards and health visualization
├── app.py – Entry script to run the main interface
├── granite_llm.py – Manages IBM Granite model interactions
├── prediction_engine.py – Contains disease prediction logic
├── treatment_planner.py – Creates treatment recommendations
└── health_dashboard.py – Displays health data and insights
```

6. Running the Application

1.Open terminal in the project directory.
2.Run: python app.py
3.Gradio will generate a local URL and optional public share link.
4.Use the interface to input symptoms or request a treatment plan.
5.AI-generated results will be displayed instantly.

7. API Documentation

This project can expose backend functions as APIs using FastAPI or Flask.

Available Functions:

- disease_prediction(symptoms): Returns possible conditions and general advice.
- treatment_plan(condition, age, gender, history): Provides a basic treatment plan customized to user input.

8. Authentication

Currently, authentication is not included since this is a prototype. For production-level applications, JWT authentication or API key validation must be integrated for security.

- QTab-based navigation for user convenience.
- Symptom input field for disease prediction.
- Inputs for condition, age, gender, and medical history.
- Generated AI responses displayed in a text box.
- Disclaimer emphasizing professional medical advice.

9. User Interface

- Tab-based navigation for user convenience.
- Symptom input field for disease prediction.
- Inputs for condition, age, gender, and medical history.
- Generated AI responses displayed in a text box.
- Disclaimer emphasizing professional medical advice.

10. Testing

- Unit Testing: Verifies AI functions and responses.
- Manual Testing: Confirms UI components work correctly in Gradio.
- Edge Case Testing: Evaluates system behavior with incomplete or unusual inputs.

11. Known Issues

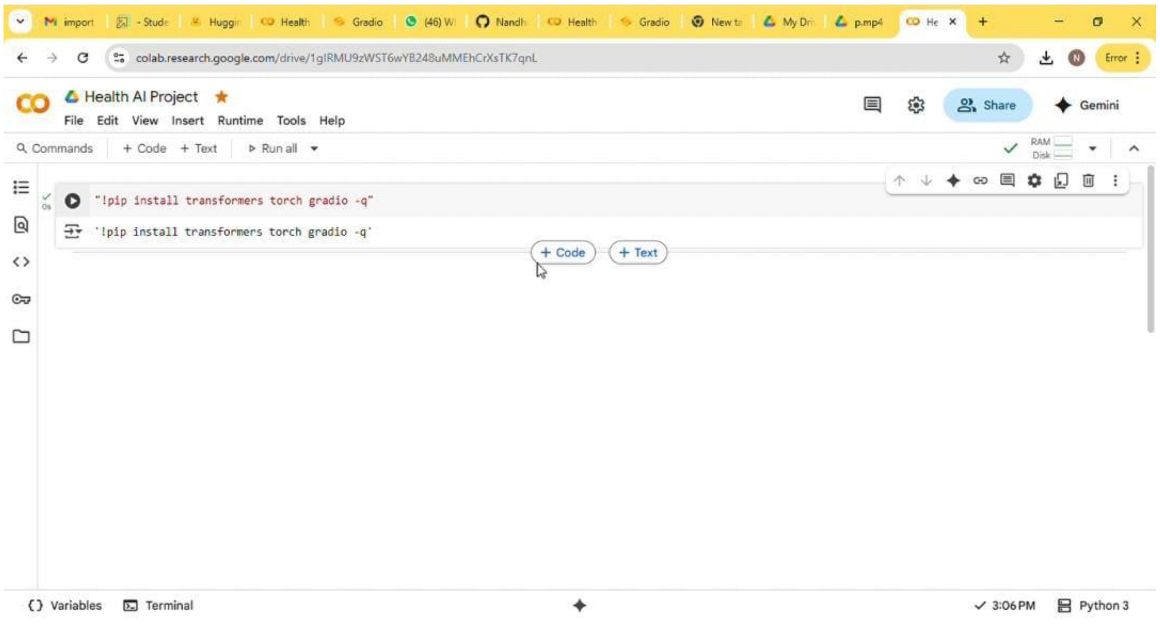
- Limited medical condition coverage due to model constraints.
- Response accuracy depends on phrasing of symptoms.
- Requires a stable internet connection.
- Not a replacement for professional healthcare advice.

12. Future Enhancements

- API deployment using FastAPI or Flask.
- Expanding the disease prediction knowledge base.
- Multi-language support for wider accessibility.
- Secure authentication methods.
- Improved accuracy with fine-tuned healthcare datasets.

13. Screenshots

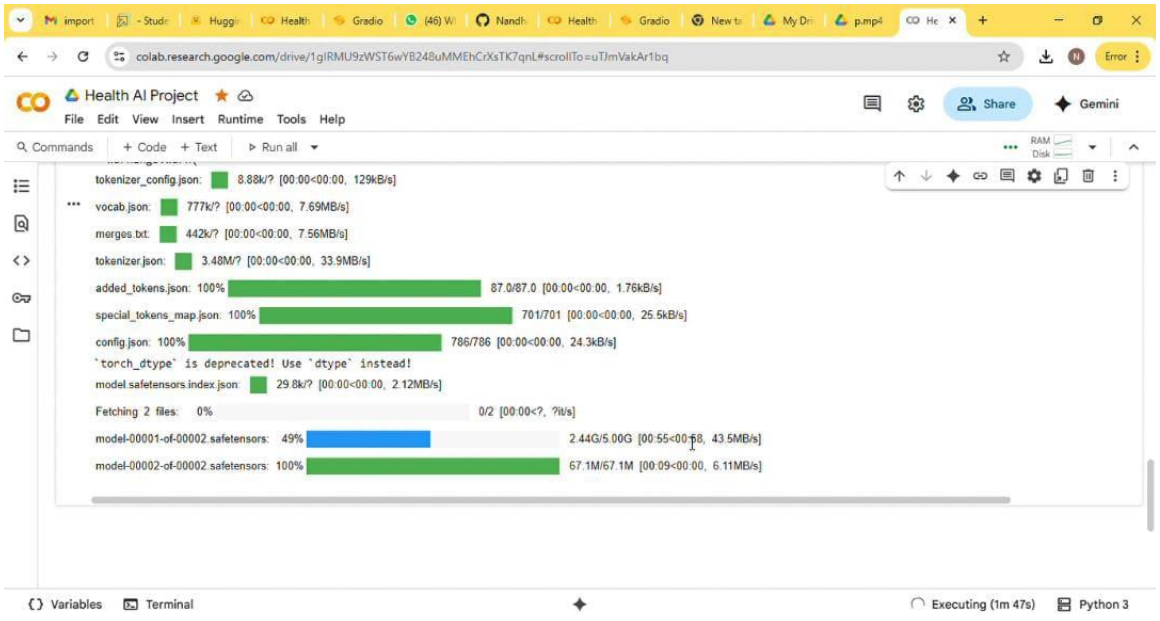
- Screen 1:


colab.research.google.com/drive/1gIRMU9zWST6wYB248uMMEhCrXsTK7qnL
Health AI Project
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands
+ Code + Text
Run all
Variables
Terminal
3:06 PM
Python 3

```
"!pip install transformers torch gradio -q"
```

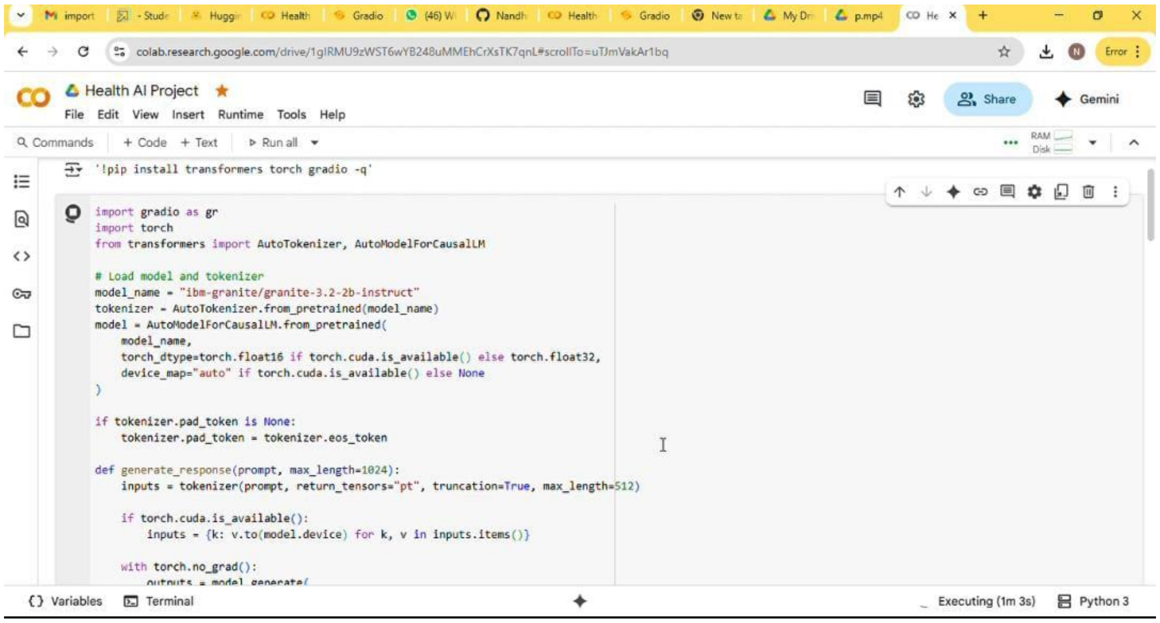
```
'!pip install transformers torch gradio -q'
```

- Screen 2:



```
tokenizer_config.json: 8.88k/? [00:00<00:00, 129kB/s]
vocab.json: 777k/? [00:00<00:00, 7.69MB/s]
merges.txt: 442k/? [00:00<00:00, 7.56MB/s]
tokenizer.json: 3.48M/? [00:00<00:00, 33.9MB/s]
added_tokens.json: 100% 87.0/87.0 [00:00<00:00, 1.76kB/s]
special_tokens_map.json: 100% 701/701 [00:00<00:00, 25.5kB/s]
config.json: 100% 786/786 [00:00<00:00, 24.3kB/s]
`torch_dtype` is deprecated! Use `dtype` instead!
model.safetensors.index.json: 29.8k/? [00:00<00:00, 2.12MB/s]
Fetching 2 files: 0% 0/2 [00:00<?, ?it/s]
model-00001-of-00002.safetensors: 49% 2.44G/5.00G [00:55<00:58, 43.5MB/s]
model-00002-of-00002.safetensors: 100% 67.1M/67.1M [00:09<00:00, 6.11MB/s]
```

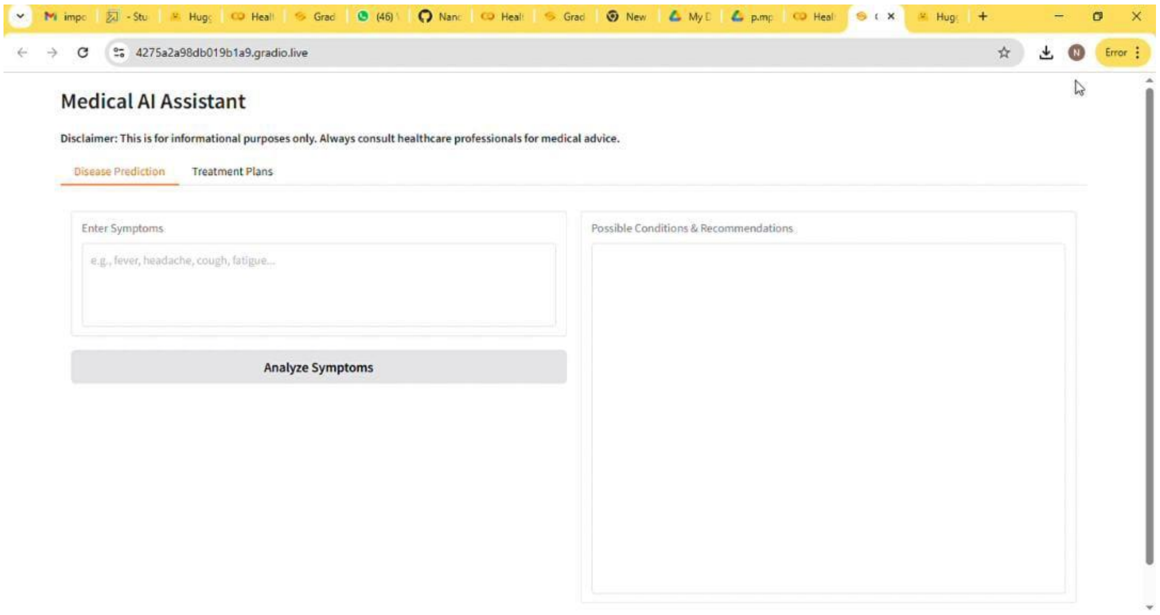
- Screen 3:



```
'!pip install transformers torch gradio -q'
```

```python
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(
```

- Screen 4:



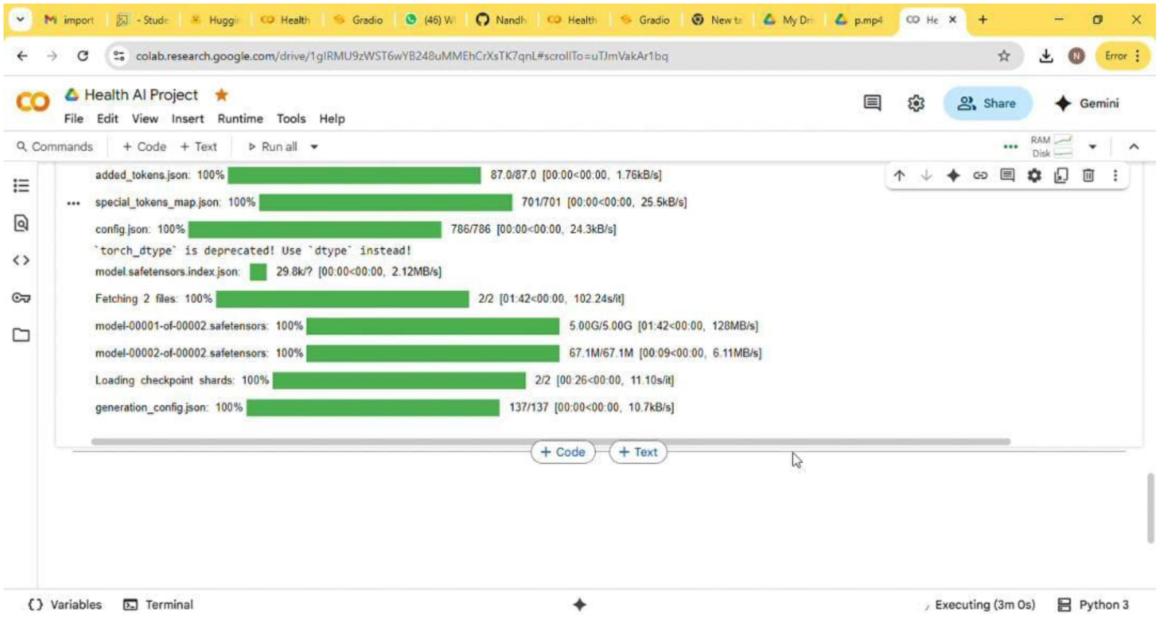
4275a2a98db019b1a9.gradio.live

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...
Analyze Symptoms
Possible Conditions & Recommendations

- Screen 5:



colab.research.google.com/drive/1gIRMU9zWST6wYB248uMMEhCrXsTK7qnL#scrollTo=uTJmVakAr1bq

```
added_tokens.json: 100% 87.0/87.0 [00:00<00:00, 1.76kB/s]
special_tokens_map.json: 100% 701/701 [00:00<00:00, 25.5kB/s]
config.json: 100% 786/786 [00:00<00:00, 24.3kB/s]
`torch_dtype` is deprecated! Use `dtype` instead!
model.safetensors.index.json: 29.8k/? [00:00<00:00, 2.12MB/s]
Fetching 2 files: 100% 2/2 [01:42<00:00, 102.24s/it]
model-00001-of-00002.safetensors: 100% 5.00G/5.00G [01:42<00:00, 128MB/s]
model-00002-of-00002.safetensors: 100% 67.1M/67.1M [00:09<00:00, 6.11MB/s]
Loading checkpoint shards: 100% 2/2 [00:26<00:00, 11.10s/it]
generation_config.json: 100% 137/137 [00:00<00:00, 10.7kB/s]
```