Health AI – Intelligent Healthcare Assistant

1. Introduction:

● Project Title:Health Ai-intelligent healthcare assistant

● Team Member 1:Evangline.s

● Team Member 2:Maythini.R

● Team Member 3:Narmadha sri.k

● Team Member 4:Lavanya.k

2.Project overview:

Health AI is an intelligent healthcare assistant powered by IBM Granite models. It is designed to assist patients and healthcare providers by offering:

- Patient Chat: A conversational interface for patients to describe symptoms and get initial guidance.
- Disease Prediction: AI-driven predictions based on patient inputs and health data.
- Treatment Plan Suggestions: Providing treatment guidance and lifestyle recommendations.
- Extensibility: Developers can add new functionalities, like medicine reminders, medical history analysis, or integration with EHR systems.

The project leverages Google Colab for deployment, ensuring that even students or researchers without high-end machines can access GPU resources.By using IBM Granite LLMs from Hugging Face, the system combines language intelligence with healthcare applications, making medical information more accessible, understandable, and interactive.

3.Architecture

The architecture of Health AI follows a simple but efficient flow:

1. User Input Layer:

- Patients enter symptoms or queries via a Gradio interface.
- Input may include text like “I have fever, cough, and fatigue.”

2. Processing Layer:

- The system sends the input to an IBM Granite model hosted on Hugging Face.
- Pre-processing ensures text is cleaned and standardized.

3. Model Inference Layer:

- The Granite model generates a predicted response, which could be:
- Probable disease condition
- Suggested treatment approach
- Follow-up advice

4. Output Layer:

- Results are displayed back to the user in the Gradio interface.
- Users can interact in real-time, like chatting with a virtual doctor.

Tech Stack:

- Frontend: Gradio (for interactive UI)
- Backend: Python + Hugging Face (IBM Granite LLMs)
- Deployment: Google Colab (T4 GPU runtime)
- Version Control: GitHub

4.Setup Instructions:

- Pre-requisites
- Before running the project, ensure:
- Basic knowledge of Python and Gradio.
- Hugging Face account (for accessing IBM Granite models).
- Google Colab account (for running the notebook with GPU).
- GitHub account (for uploading project files).

Step-by-Step Setup

Step-1.Open Google Colab.

- Visit Google Colab.
- Create a new notebook.

Step 2: Change Runtime

- Go to Runtime → Change Runtime Type.
- Select GPU (T4) and save.

Step 3: Install Dependencies

- !pip install transformers torch gradio -q

Step 4: Load IBM Granite Model

- Login to Hugging Face.
- Select IBM Granite model (e.g., granite-3.2-2b-instruct).

Step 5: Run Application

- Execute the notebook cells.
- Gradio will generate a link → open in a new tab.

Step 6: Upload to GitHub

- Download notebook as .py or .ipynb.
- Push to GitHub repository.

5. Folder Structure

A well-defined folder structure ensures clarity, easy collaboration, and better project maintenance. The proposed structure for the HealthAI project is organized as follows:

- HealthAI/
  The root directory of the project, containing all essential files and subfolders.
- Notebooks/
  Contains Jupyter or Colab notebooks used for experimentation, prototyping, and testing.
- HealthAI.ipynb: The main notebook where initial model development, data exploration, or demonstrations are carried out.
- Src/This is the source code directory containing the main logic of the application.
- App.py: Defines the Gradio interface and application flow.
- Model.py: Handles model loading, initialization, and inference functions.
- Utils.py: Includes helper or utility functions to support preprocessing, evaluation, or other repetitive tasks.
- Models/Stores model-related files such as weights, checkpoints, or configuration details.
- Config.json: Contains model configuration parameters, hyperparameters, or setup details.
- Docs/Contains project-related documentation.
- ProjectReport.md: Extended documentation of the project, including design, architecture, and usage details.
- Screenshots/Stores visual assets such as interface previews, diagrams, or workflow images.
- Ui.png: Screenshot of the application's user interface.
- Requirements.txt
  A list of all Python dependencies required to run the project (e.g., Gradio, Transformers, etc.).
- README.md

Provides a short overview of the project, setup instructions, and quick usage guidelines.

6.Running the Application

1. Open Google Colab Notebook → HealthAI.ipynb.

2. Install dependencies.

3. Load IBM Granite model from Hugging Face.

4. Start Gradio app → copy the generated URL.

5. Open in browser → interact with the chatbot.

7. API Documentation

Currently, the app is designed as an interactive tool, but can also act like an API.

- Endpoint 1 – Chat
- Input: Patient text (symptoms/query).
- Output: AI-generated conversational reply.

Endpoint 2 – Disease Prediction

- Input: Symptom description.
- Output: Possible diseases.

Endpoint 3 – Treatment Suggestion

- Input: Disease or symptom query.
- Output: Suggested treatments & lifestyle tips.

8.Authentication

- Hugging Face API Key required to download IBM Granite models.
- Store API keys securely (never upload to GitHub).

Use environment variables in Colab:

Import os os.environ["HF_TOKEN"] = "your_huggingface_token"

9. User Interface

Built with Gradio, which provides a simple UI:

- Text box for inputting patient symptoms.

- Chat window for conversational interaction.
- Results panel for displaying predictions and treatment plans.
- Mobile-friendly and lightweight.

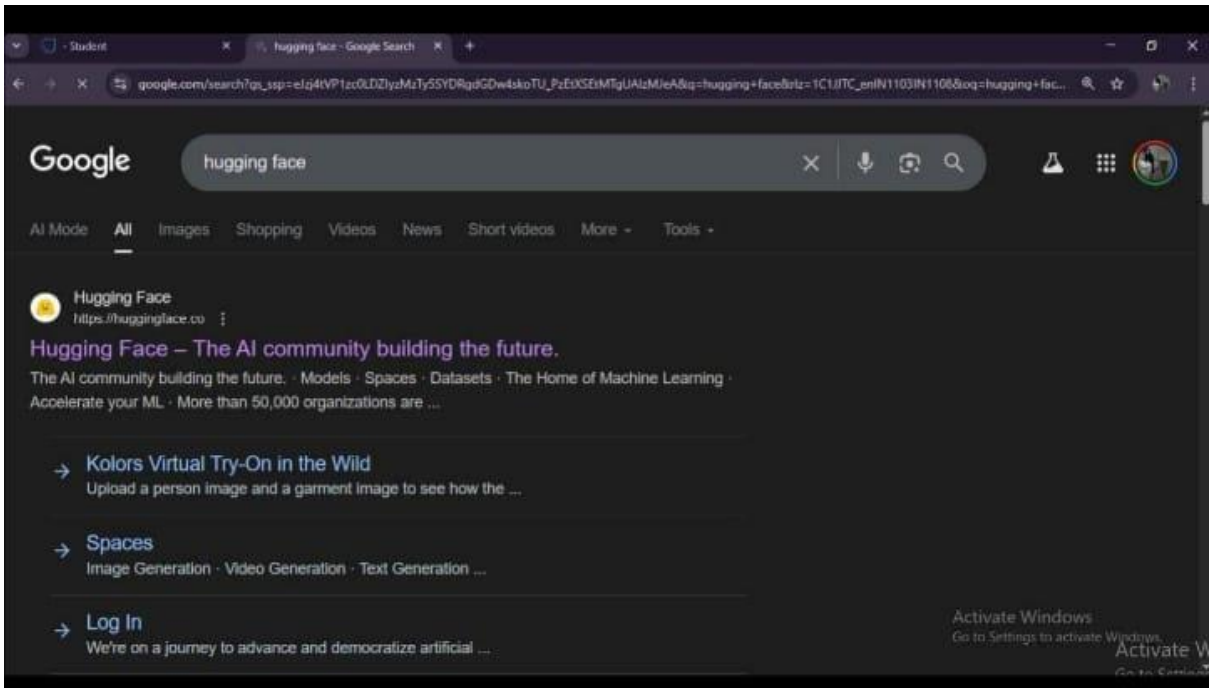
10. Testing

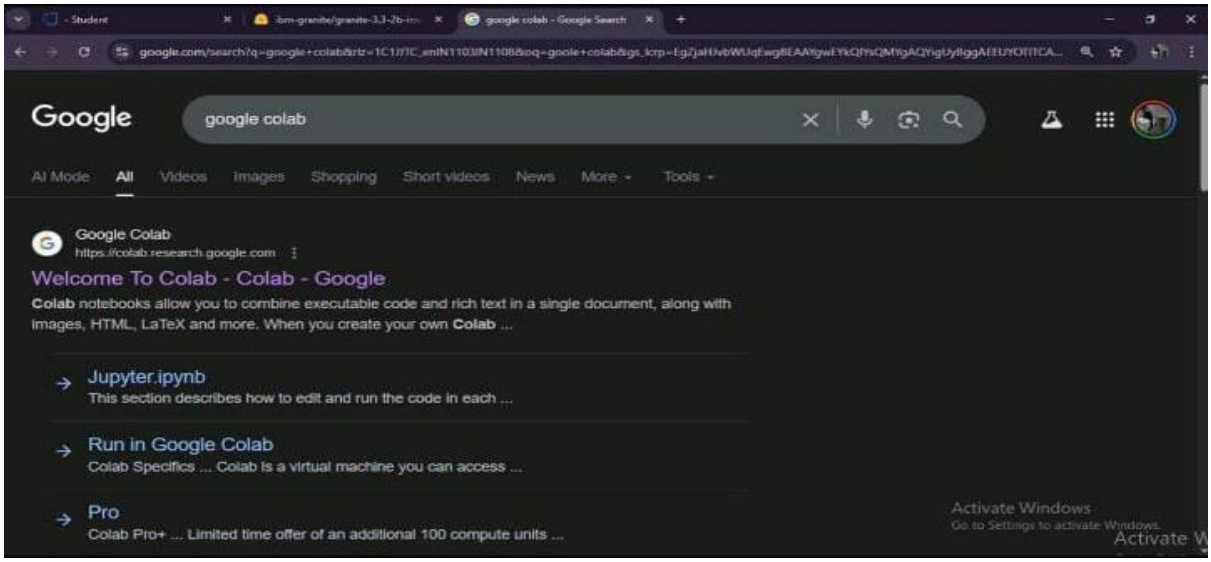
- Unit Testing: Validate each function in utils.py.
- Integration Testing: Ensure Hugging Face API works with Colab.
- User Testing: Try queries like:

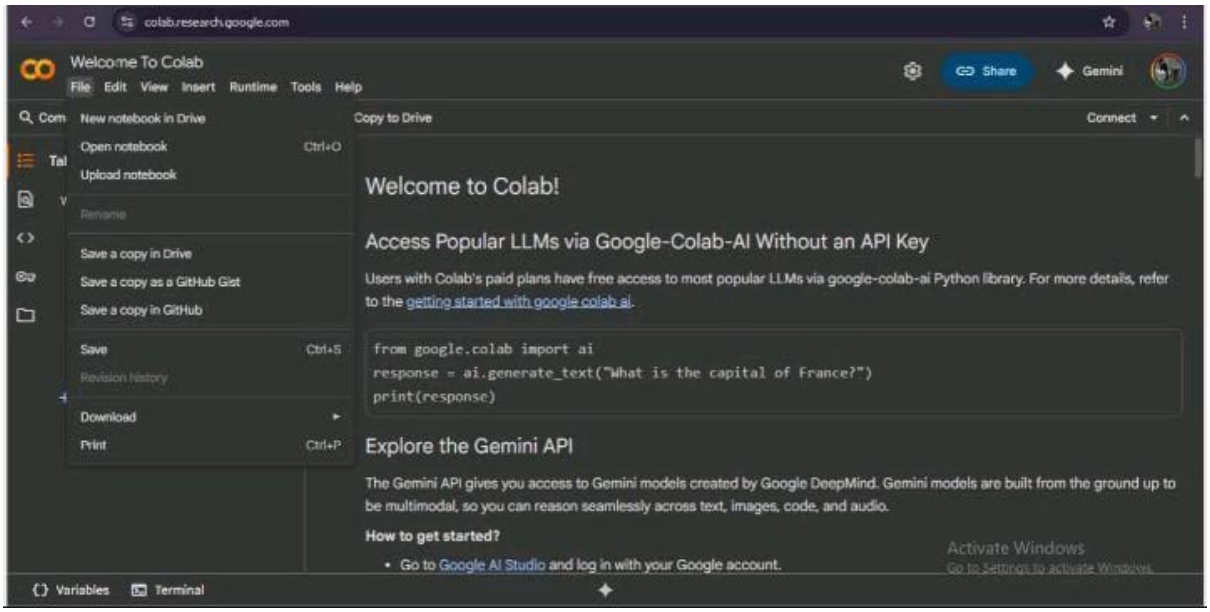
“I have cough and fever” → Predict flu or cold.

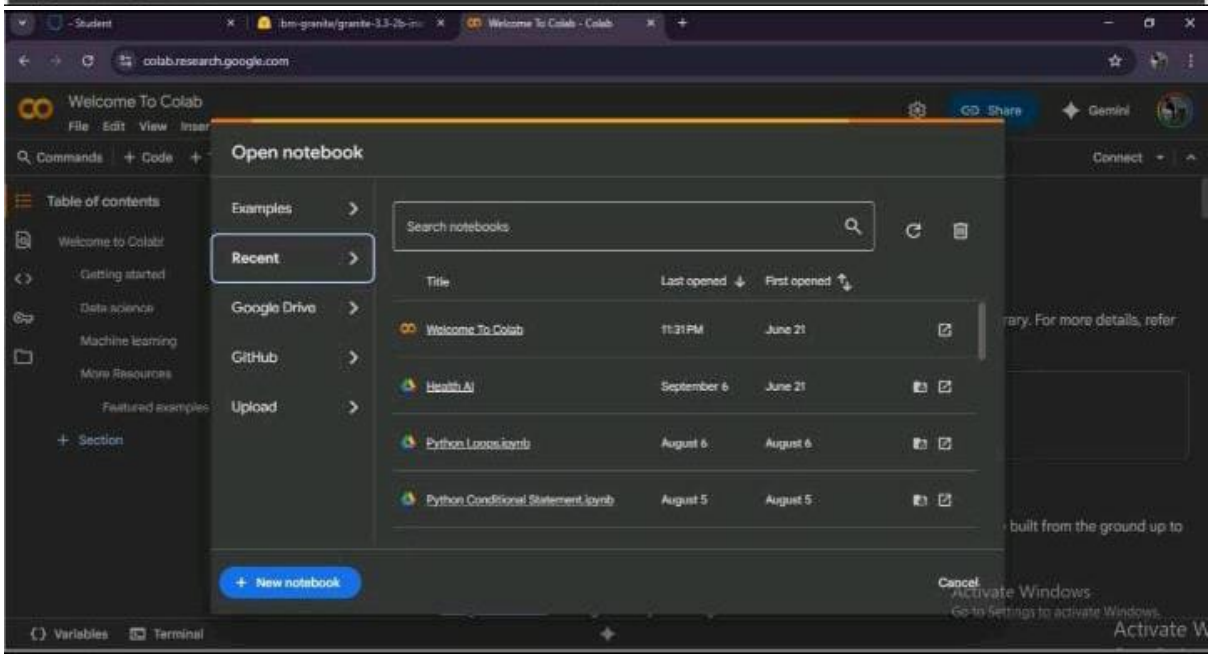
“What is the treatment for diabetes?” → Suggest lifestyle + medication advice.

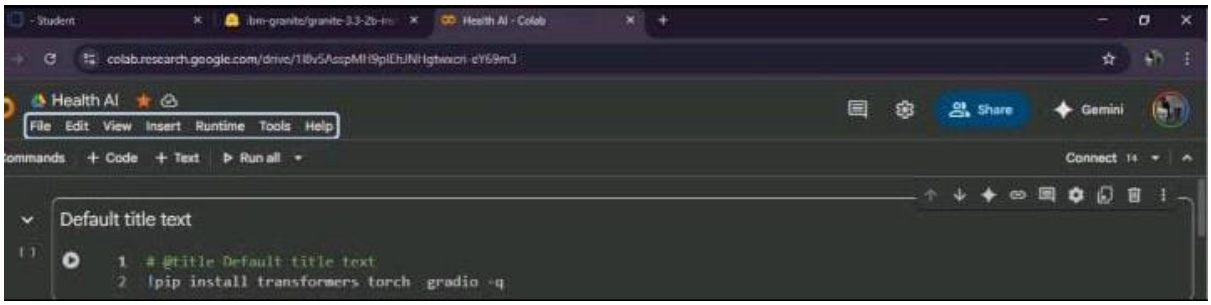
11. Screenshots/demo

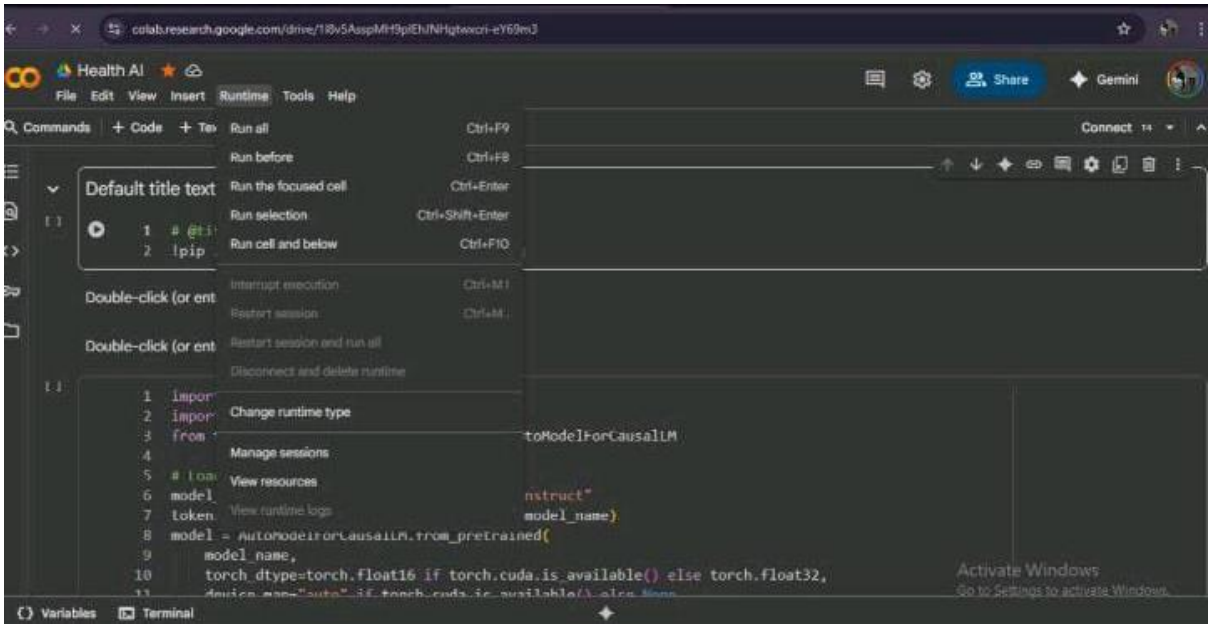


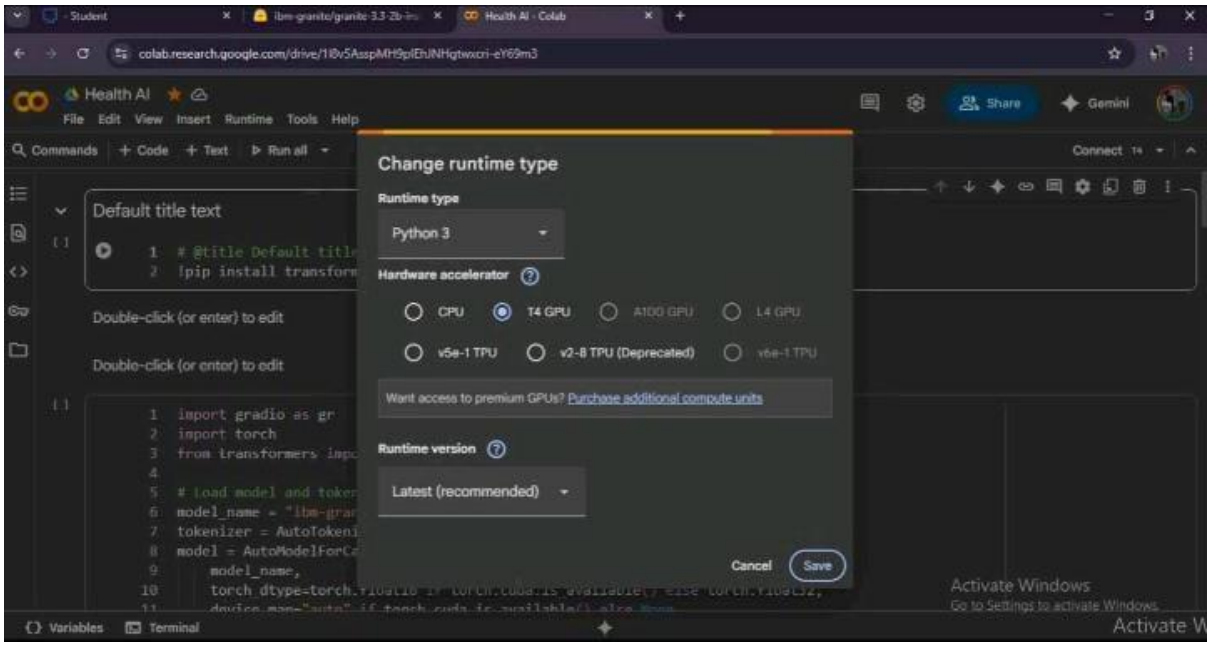

Google
google colab
AI Mode
All
Videos
Images
Shopping
Short videos
News
More
Tools
Google Colab
https://colab.research.google.com
Welcome To Colab - Colab - Google
Colab notebooks allow you to combine executable code and rich text in a single document, along with images, HTML, LaTeX and more. When you create your own Colab ...
Jupyter.ipynb
This section describes how to edit and run the code in each ...
Run in Google Colab
Colab Specifics ... Colab is a virtual machine you can access ...
Pro
Colab Pro+ ... Limited time offer of an additional 100 compute units ...
Activate Windows

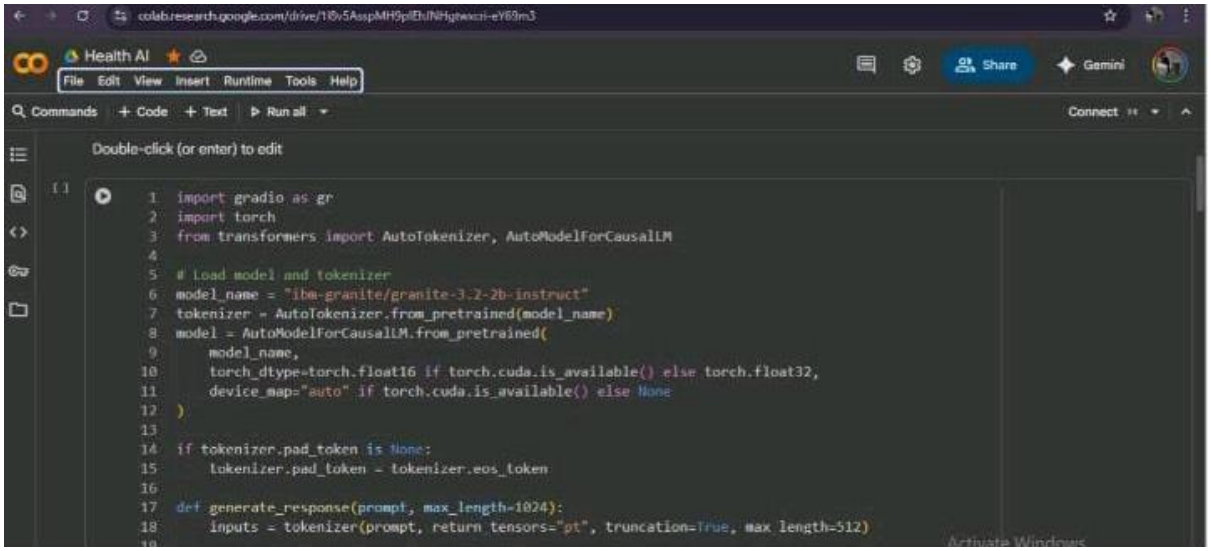

colab.research.google.com
Welcome To Colab
File Edit View Insert Runtime Tools Help
Share
Gemini
Connect
New notebook in Drive
Open notebook Ctrl+O
Upload notebook
Rename
Save a copy in Drive
Save a copy as a GitHub Gist
Save a copy in GitHub
Save Ctrl+S
Revision history
Download
Print Ctrl+P
Copy to Drive
Welcome to Colab!
Access Popular LLMs via Google-Colab-AI Without an API Key
Users with Colab's paid plans have free access to most popular LLMs via google-colab-ai Python library. For more details, refer to the getting started with google colab ai.
from google.colab import ai
response = ai.generate_text("What is the capital of France?")
print(response)
Explore the Gemini API
The Gemini API gives you access to Gemini models created by Google DeepMind. Gemini models are built from the ground up to be multimodal, so you can reason seamlessly across text, images, code, and audio.
How to get started?
Go to Google AI Studio and log in with your Google account.
Variables
Terminal
Activate Windows


colab.research.google.com
Welcome To Colab
Open notebook
Examples
Recent
Google Drive
GitHub
Upload
Search notebooks
Title
Last opened
First opened
Welcome To Colab
11:31 PM
June 21
Health AI
September 6
June 21
Python Loops.ipynb
August 6
August 6
Python Conditional Statement.ipynb
August 5
August 5
New notebook
Cancel
Table of contents
Welcome to Colab!
Getting started
Data science
Machine learning
More Resources
Featured examples
Section
Variables
Terminal
Activate Windows

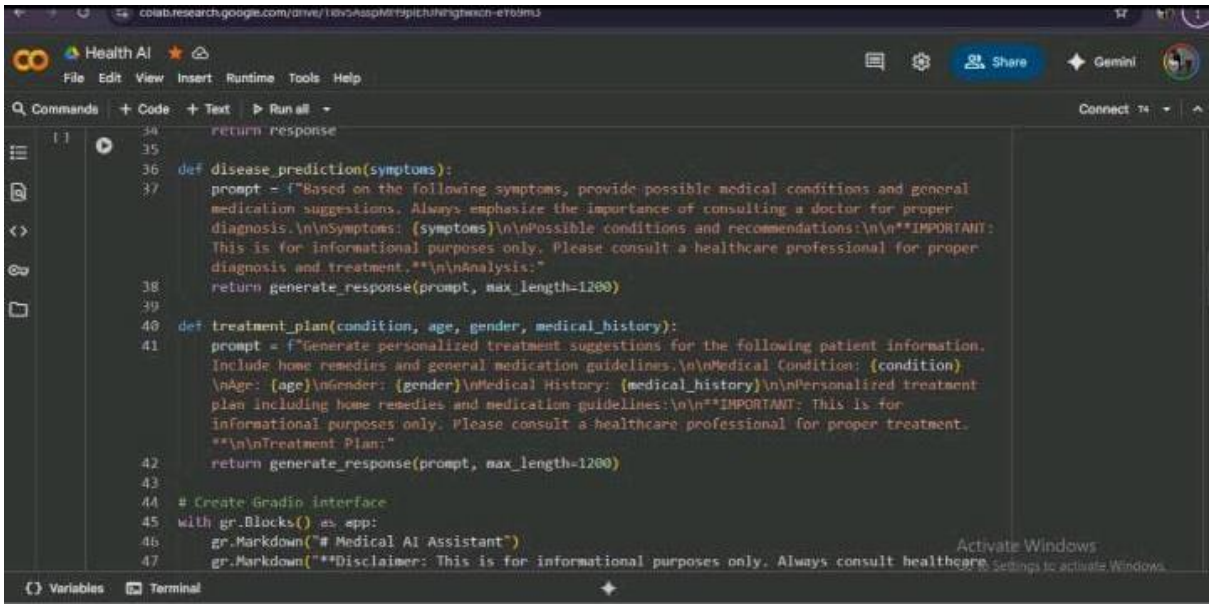

Health AI
File Edit View Insert Runtime Tools Help
Share
Gemini
Connect
Default title text
@title Default title text
!pip install transformers torch gradio -q

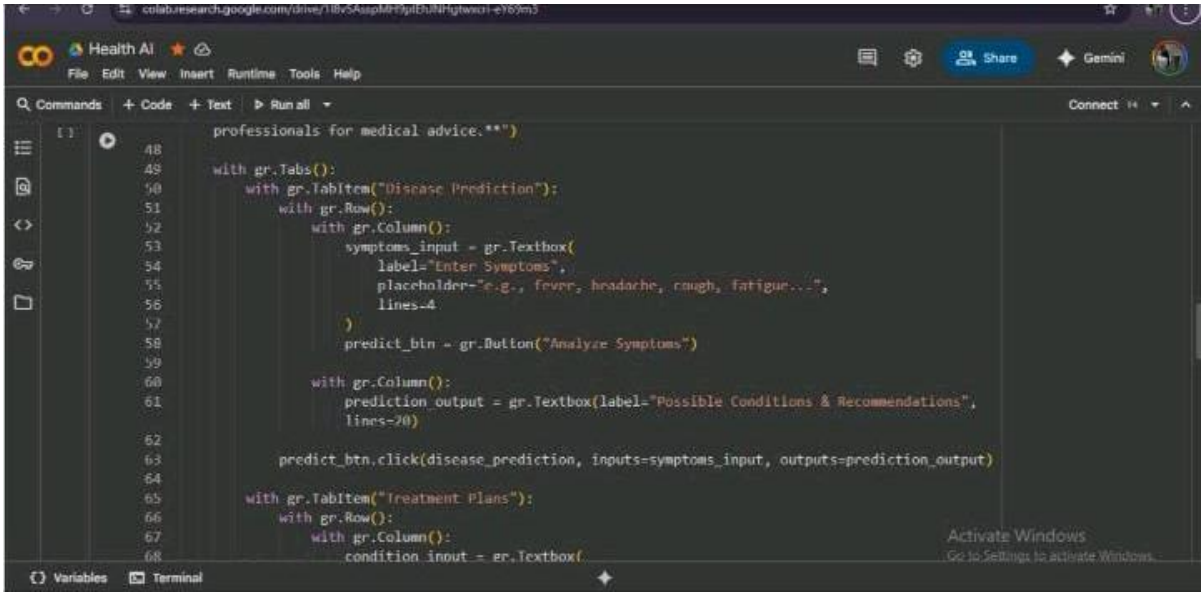

Health AI
File Edit View Insert Runtime Tools Help
Share
Gemini
Connect
Run all Ctrl+F9
Run before Ctrl+F8
Run the focused cell Ctrl+Enter
Run selection Ctrl+Shift+Enter
Run cell and below Ctrl+F10
Interrupt execution
Restart session
Restart session and run all
Disconnect and delete runtime
Change runtime type
Manage sessions
View resources
View runtime logs
Default title text
Double-click (or ent
Double-click (or ent
torch_dtype=torch.float16 if torch.cuda.is_available() else torch.float32,
Activate Windows
Variables
Terminal


Health AI
File Edit View Insert Runtime Tools Help
Share
Gemini
Connect
Change runtime type
Runtime type
Python 3
Hardware accelerator
CPU
T4 GPU
A100 GPU
L4 GPU
v5e-1 TPU
v2-8 TPU (Deprecated)
v6e-1 TPU
Want access to premium GPUs? Purchase additional compute units
Runtime version
Latest (recommended)
Cancel
Save
Default title text
Double-click (or enter) to edit
Double-click (or enter) to edit
import gradio as gr
import torch
Activate Windows
Variables
Terminal

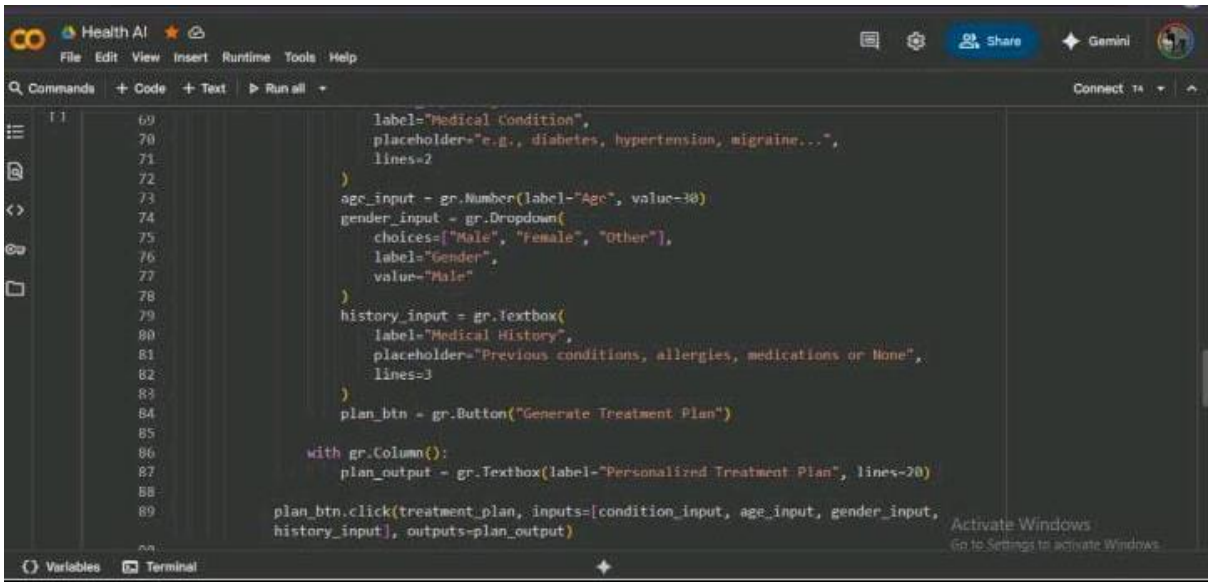

Health AI
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands + Code + Text Run all
Connect
Double-click (or enter) to edit
import gradio as gr
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM
# Load model and tokenizer
model_name = "ibm-granite/granite-3.2-2b-instruct"
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)
Activate Windows


Health AI
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands + Code + Text Run all
Connect
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 and general
medication suggestions. Always emphasize the importance of consulting a doctor for proper
Activate Windows
Go to Settings to activate Windows.

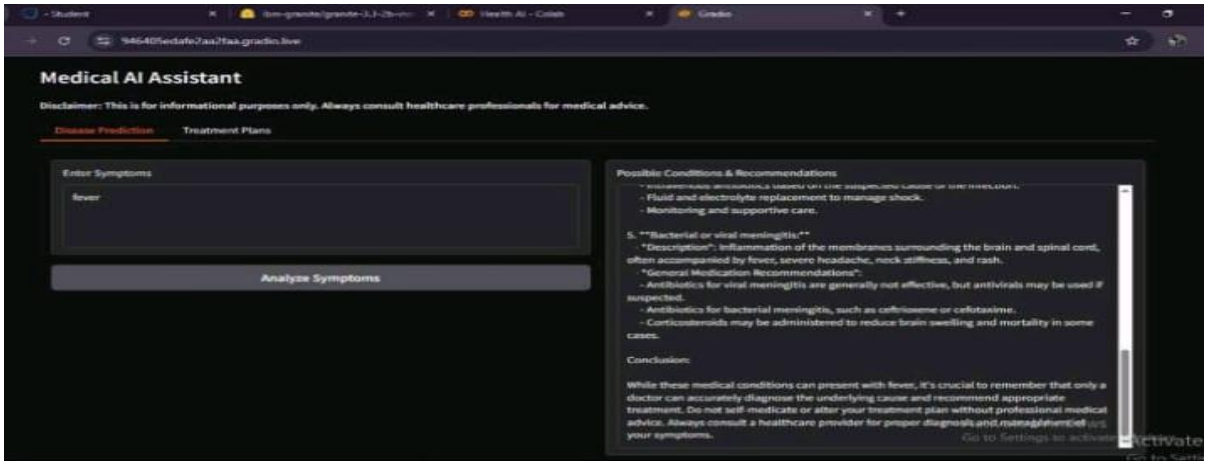

Health AI
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands + Code + Text Run all
Connect
return response
def disease_prediction(symptoms):
prompt = f"Based on the following symptoms, provide possible medical conditions and general
medication suggestions. Always emphasize the importance 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)
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)
Create Gradio interface
with gr.Blocks() as app:
gr.Markdown("# Medical AI Assistant")
gr.Markdown("**Disclaimer: This is for informational purposes only. Always consult healthcare
Activate Windows
Go to Settings to activate Windows.
Variables
Terminal



```
Health AI
File Edit View Insert Runtime Tools Help
Commands + Code + Text ▷ Run all
Connect
        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(
Activate Windows
Go to Settings to activate Windows.
Variables Terminal
```



```
Health AI
File Edit View Insert Runtime Tools Help
Commands + Code + Text ▷ Run all
Connect
                    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)
Activate Windows
Go to Settings to activate Windows.
Variables Terminal
```



12. Known Issues

- Dependency on Google Colab GPU (sessions may timeout).
- Hugging Face model load time is slow initially.
- Limited to text-only input (no speech/image support yet).
- Predictions are not medical diagnoses – only guidance.

13. Future Enhancements

- Deploy on IBM Cloud for better scalability.
- Voice-based assistant for accessibility.
- Integration with wearables (smartwatch, fitness tracker).
- Multilingual support for rural/remote users.
- Larger medical dataset for better disease coverage.