Citizen AI Intelligent Citizen Engagement Platform

1. Introduction

- **Project title** : Citizen AI Platform
- **Team member** : MOHAMED IRFAN.M
- **Team member** : MATHESH.N
- **Team membe**r : MOHAMED ASIK .M
- **Team membe**r : MEENAKSHI
  KRISHNA P

2. Project overview

- **Purpose:**

To build a Generative AI-based Citizen Engagement assistant using IBM Granite, This AI assistant for urban safety insights and civic engagement, helping citizens stay informed and enabling governments to provide accessible, AI-driven public support.

- **Fearture:**

1. City Analysis

Accepts a city name as input.
Generates a detailed AI-based analysis including:
Crime index and safety statistics.
Accident rates and traffic safety data.
Overall safety and livability assessment.
Provides results in a clear text format for quick understanding.

2. Citizen Services Assistant

Accepts citizen queries related to public services, policies, or civic issues.
Responds with helpful, government-style answers, making information more accessible.
Covers multiple topics such as health services, transport, education policies, etc.

3. AI-Powered Responses

Uses the IBM Granite language model to generate accurate and human-like responses.
Supports context-aware answers instead of static information.

4. User-Friendly Interface

Built with Gradio, offering a simple web-based interface.

Organized into two tabs:
City Analysis
Citizen Services
Easy input fields and clickable buttons for smooth user interaction.

5. Cross-Platform Support

Can run on CPU or GPU depending on availability.
Shareable web interface (share=True) so users can access it remotely.

6. Customizable&Scalable

Developers can modify prompts to adapt the system for:
City planning.
Public safety dashboards.
Digital government services.
Research and policy-making.

3. Architecture

1.User Interface Layer (Frontend)

Built with Gradio Blocks&Tabs.
Provides two interactive sections:

City Analysis Tab → Input city name, get safety analysis.

Citizen Services Tab → Input query, get government-style response.

Outputs are displayed in text boxes for readability.

2. Application Layer (Backend Logic)

Python Functions handle core logic:

city_analysis(city_name) → Generates structured prompt for city safety analysis.

citizen_interaction(query) → Generates prompt for government/civic queries.

generate_response(prompt) → Calls AI model, processes input, and returns response.

3. AI Model Layer

Uses IBM Granite 3.2-2B Instruct (LLM) from Hugging Face.

Tokenizer processes input text into model-ready tokens.

Causal Language Model (AutoModelForCausalLM) generates human-like responses.

Supports GPU acceleration if available (with torch_dtype=torch.float16&device_map="auto").

4. Setup Instructions

1. Prerequisites

Before running the program, ensure you have:
Python 3.9+ installed.
pip (Python package manager).
A system with GPU (CUDA) for faster inference (optional, CPU also works).
Internet connection (to download the AI model from Hugging Face).

2. Install Required Packages

Open a terminal (or Google Colab cell) and run:
pip install torch transformers gradio -q
torch → Deep learning framework for running models.
transformers → Hugging Face library to load IBM Granite model.
gradio → To create the web-based interface.

5. Folder Structure

app.py → Main entry point that ties everything together and launches Gradio app.

requirements.txt → Keeps track of Python dependencies.

config/ → Stores configurations like model name, max token length, temperature, etc.

models/ → Code for loading the IBM Granite model&tokenizer.

services/ → Business logic split into modules: city analysis&citizen services.

utils/ → Helper functions for building prompts and cleaning AI responses.

tests/ → Unit tests for ensuring correctness of features.

docs/ → Contains documentation (architecture, features, setup).

6. Running the Application

1.User Interface Layer (Frontend)

Built with Gradio Blocks&Tabs.
Provides two interactive sections:
City Analysis Tab → Input city name, get safety analysis.

Citizen Services Tab → Input query, get government-style response.

Outputs are displayed in text boxes for readability.

2. Application Layer (Backend Logic)

Python Functions handle core logic:

city_analysis(city_name) → Generates structured prompt for city safety analysis.

citizen_interaction(query) → Generates prompt for government/civic queries.

generate_response(prompt) → Calls AI model, processes input, and returns response.

7. API Documentation

city_analysis(city_name: str) ->str

Description:
Generates a detailed analysis of a given city.

Parameters:
city_name (str) → Name of the city (e.g.,"Mumbai","London").

Response Structure:
Crime index&safety statistics.

Accident rates&traffic safety information.
Overall safety assessment.

citizen_interaction(query: str) ->str

Description:
Provides AI-powered responses to citizen queries related to government services, policies, or civic issues.

Parameters:
query (str) → Citizen’s question (e.g.,"How to apply for a driving license?").

Response Structure:
Clear, government-style response with actionable details.

8. Authentication

1. Simple Password Protection (Gradio Built-in)
Gradio provides username/password login out of the box:

2. Environment Variable Authentication

Store credentials in .env (never hardcode passwords):

3. Token-Based Authentication (for API use)

If you want API endpoints use a Bearer Token:

```
from fastapi import FastAPI, Header, HTTPException
```

4. OAuth2 / Google Login (Advanced)

If you want government/corporate style login (like Google or GitHub OAuth), you'll need to integrate Gradio with FastAPI/Flask + OAuth.

```
Gradio → UI
FastAPI/Flask → Authentication middleware
Example libraries: authlib, flask_oauthlib
```

9. User Interface

The application uses Gradio Blocks to build an interactive web-based UI.

It is divided into two main tabs:

1. City Analysis – for analyzing crime, accident, and safety data of a city.

2. Citizen Services – for answering queries about government services, policies, and civic issues.

Simple design with text inputs, buttons, and output boxes for readability.

UI Components

gr.Markdown("# City Analysis&Citizen Services AI")
Displays the app title at the top of the interface.

Input Box:
gr.Textbox (label: Enter City Name)
Example:"Mumbai","New York".

Analyze Button:
gr.Button("Analyze City")
Triggers the city_analysis() function.

Output Box:
gr.Textbox (label: City Analysis (Crime Index&Accidents))
Displays AI-generated city safety analysis in multi-line format.

Query Box:
gr.Textbox (label: Your Query)
Example:"How to apply for a driving license?".

Get Info Button:
gr.Button("Get Information")

Triggers the citizen_interaction() function.

Response Box:
gr.Textbox (label: Government Response)
Displays AI-powered answers to civic queries.

User Flow

1. Select a Tab (City Analysis / Citizen Services).
2. Enter Input (City name or Query).
3. Click Button (Analyze City / Get Information).
4. View Output in the response textbox.

10. Testing

Unit Testing → Test individual functions (generate_response, city_analysis, citizen_interaction).

Integration Testing → Ensure Gradio UI connects correctly with backend functions.

Error Handling Testing → Test invalid inputs (empty city names, nonsensical queries).

Performance Testing → Ensure responses are generated within acceptable time.

Testing Tools

pytest → For unit and integration testing.

unittest → Built-in Python testing framework (alternative).

manual testing → Run app locally and test UI flows.

Enter a valid city (e.g., Mumbai) → Response generated.

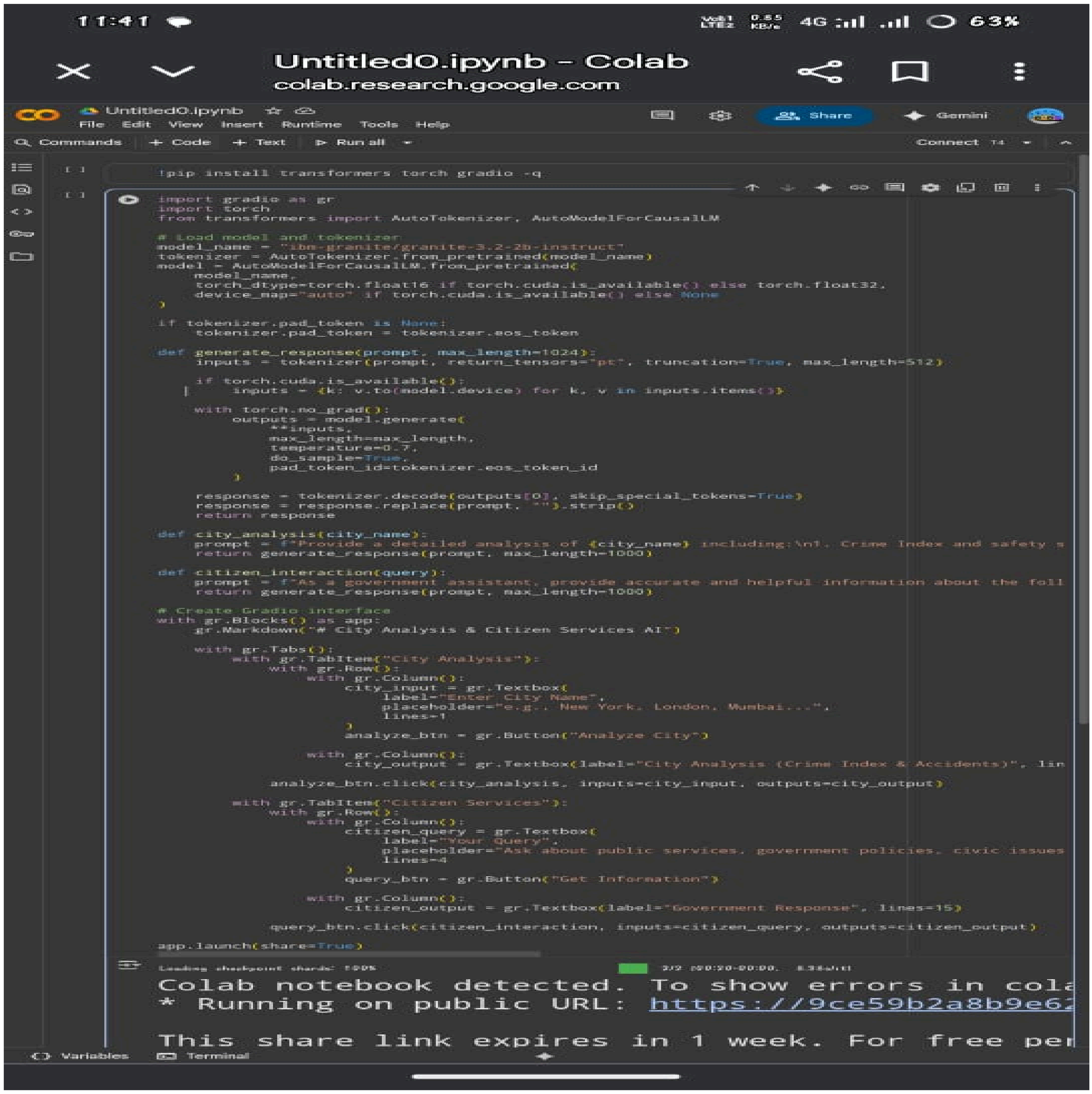
Enter an invalid/empty city → Graceful error message.

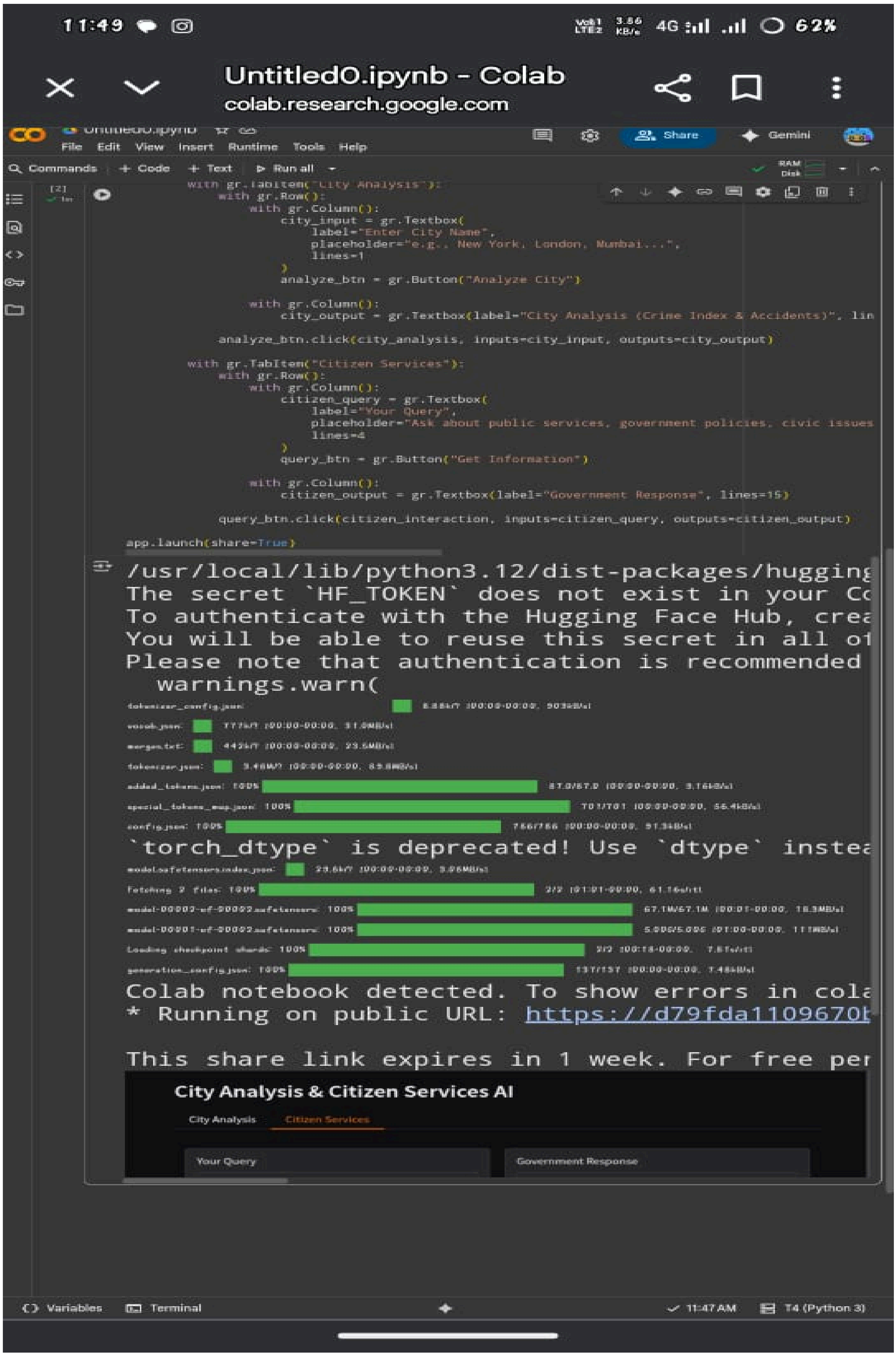
Ask valid query (e.g., Driving License) → Correct response.

Ask irrelevant query (e.g., favorite color) → AI gives fallback response.

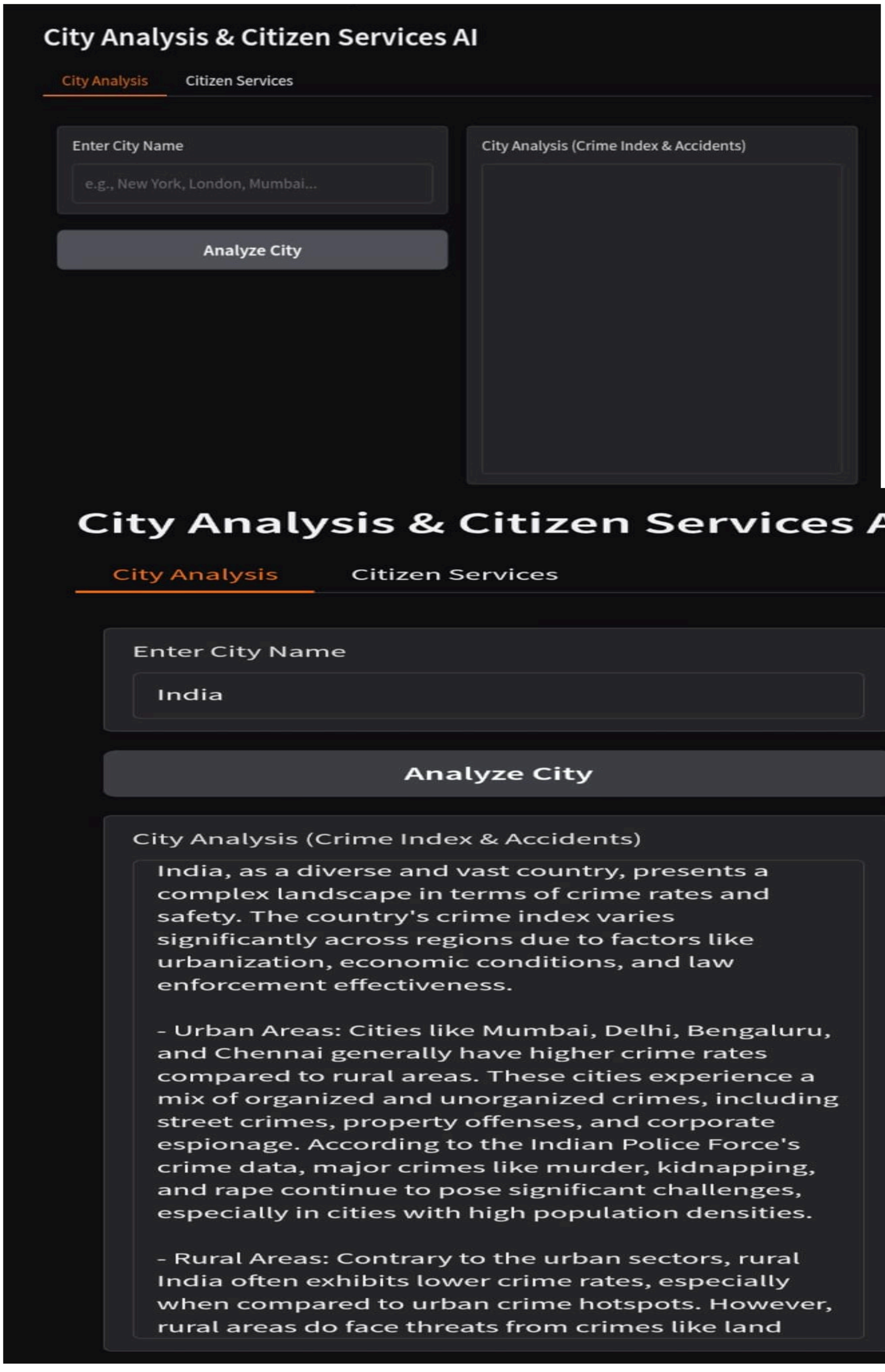
11. screen shots

Input


11:41
63%
UntitledO.ipynb – Colab
colab.research.google.com
UntitledO.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands
+ Code
+ Text
Run all
Connect T4
!pip install transformers torch gradio -q
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(
**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 city_analysis(city_name):
prompt = f"Provide a detailed analysis of {city_name} including:\n1. Crime Index and safety s
return generate_response(prompt, max_length=1000)
def citizen_interaction(query):
prompt = f"As a government assistant, provide accurate and helpful information about the foll
return generate_response(prompt, max_length=1000)
Create Gradio interface
with gr.Blocks() as app:
gr.Markdown("# City Analysis & Citizen Services AI")
with gr.Tabs():
with gr.TabItem("City Analysis"):
with gr.Row():
with gr.Column():
city_input = gr.Textbox(
label="Enter City Name",
placeholder="e.g., New York, London, Mumbai...",
lines=1
)
analyze_btn = gr.Button("Analyze City")
with gr.Column():
city_output = gr.Textbox(label="City Analysis (Crime Index & Accidents)", lin
analyze_btn.click(city_analysis, inputs=city_input, outputs=city_output)
with gr.TabItem("Citizen Services"):
with gr.Row():
with gr.Column():
citizen_query = gr.Textbox(
label="Your Query",
placeholder="Ask about public services, government policies, civic issues
lines=4
)
query_btn = gr.Button("Get Information")
with gr.Column():
citizen_output = gr.Textbox(label="Government Response", lines=15)
query_btn.click(citizen_interaction, inputs=citizen_query, outputs=citizen_output)
app.launch(share=True)
Colab notebook detected. To show errors in cola
* Running on public URL: https://9ce59b2a8b9e6
This share link expires in 1 week. For free per
Variables
Terminal


11:49
Untitled0.ipynb - Colab
colab.research.google.com
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands + Code + Text Run all
RAM
Disk
with gr.TabItem("City Analysis"):
with gr.Row():
with gr.Column():
city_input = gr.Textbox(
label="Enter City Name",
placeholder="e.g., New York, London, Mumbai...",
lines=1
)
analyze_btn = gr.Button("Analyze City")
with gr.Column():
city_output = gr.Textbox(label="City Analysis (Crime Index & Accidents)", lin
analyze_btn.click(city_analysis, inputs=city_input, outputs=city_output)
with gr.TabItem("Citizen Services"):
with gr.Row():
with gr.Column():
citizen_query = gr.Textbox(
label="Your Query",
placeholder="Ask about public services, government policies, civic issues
lines=4
)
query_btn = gr.Button("Get Information")
with gr.Column():
citizen_output = gr.Textbox(label="Government Response", lines=15)
query_btn.click(citizen_interaction, inputs=citizen_query, outputs=citizen_output)
app.launch(share=True)
/usr/local/lib/python3.12/dist-packages/hugging
The secret `HF_TOKEN` does not exist in your Co
To authenticate with the Hugging Face Hub, crea
You will be able to reuse this secret in all of
Please note that authentication is recommended
warnings.warn(
`torch_dtype` is deprecated! Use `dtype` instea
Colab notebook detected. To show errors in cola
* Running on public URL: https://d79fda1109670b
This share link expires in 1 week. For free per
City Analysis & Citizen Services AI
City Analysis
Citizen Services
Your Query
Government Response
Variables
Terminal
11:47 AM
T4 (Python 3)

Output


City Analysis & Citizen Services AI
City Analysis
Citizen Services
Enter City Name
e.g., New York, London, Mumbai...
Analyze City
City Analysis (Crime Index & Accidents)
City Analysis & Citizen Services AI
City Analysis
Citizen Services
Enter City Name
India
Analyze City
City Analysis (Crime Index & Accidents)
India, as a diverse and vast country, presents a complex landscape in terms of crime rates and safety. The country's crime index varies significantly across regions due to factors like urbanization, economic conditions, and law enforcement effectiveness.
- Urban Areas: Cities like Mumbai, Delhi, Bengaluru, and Chennai generally have higher crime rates compared to rural areas. These cities experience a mix of organized and unorganized crimes, including street crimes, property offenses, and corporate espionage. According to the Indian Police Force's crime data, major crimes like murder, kidnapping, and rape continue to pose significant challenges, especially in cities with high population densities.
- Rural Areas: Contrary to the urban sectors, rural India often exhibits lower crime rates, especially when compared to urban crime hotspots. However, rural areas do face threats from crimes like land

City Analysis & Citizen Services AI

City Analysis | Citizen Services

Your Query

Ask about public services, government policies, civic issues...

Get Information

Government Response

12. Known Issues

AI Response Accuracy
Performance Limitations
Authentication Simplicity
Input Sensitivity
Limited Multilingual Support
Session Dependency

13. Future enhancement

Integration with Real-Time Data Sources
Advanced Authentication&Security
Multilingual Support
Mobile-Friendly Interface
Offline&Low-Resource Mode
Improved Error Handling
Personalized Citizen Assistance
Dashboard&Visualization
Scalability&Deployment
Testing&Mocking Improvements