Citizen AI: Intelligent Citizen Engagement Platform

Citizen AI
Intelligent Citizen
Engagement Platform

1. Introduction

Citizen AI is an intelligent citizen engagement platform designed to provide quick, helpful answers about government services and civic issues. By leveraging IBM Granite models from Hugging Face, the project uses generative AI to assist citizens while also tracking public sentiment. The system presents results via a simple dashboard for government officials, ensuring transparency and better decision-making.

2. Project Overview

Objective: Improve citizen–government interactions using AI.

Core Features:

AI-powered chatbot for answering civic/government service queries.

Sentiment analysis for understanding public feedback.

Dashboards for officials to visualize trends.

Deployment: Built and executed on Google Colab with lightweight Granite models for low-cost, GPU-backed performance.

Tech Stack:

Python (core programming)

Transformers & Torch (model execution)

Gradio (UI for chatbot)

IBM Granite Models (Hugging Face)

GitHub (version control & hosting code)

3. Architecture

High-Level Flow:

1. Frontend (Gradio UI) – User interacts via chatbot interface.

2. Model (Granite-3.2-2b-instruct) – Processes queries and generates responses.

3. Sentiment Analysis Layer – Tracks public mood and feedback.

4. Backend (Colab Runtime) – Executes Python code, loads AI models, and manages API calls.

5. Dashboard (Optional) – Displays analytics for government officials.

6. Version Control – Source code maintained on GitHub.

4. Setup Instructions

Prerequisites

Basic knowledge of Python, Gradio, Git.

IBM Granite Models access (Hugging Face account).

Google Colab (with T4 GPU).

Steps

1. Google Colab Setup

Open Google Colab

Create a new notebook and rename it to Citizen AI.

Go to Runtime → Change runtime type → Select T4 GPU.

Install dependencies:

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

2. Model Selection

Create a Hugging Face account.

Choose granite-3.2-2b-instruct model.

3. Run Application

Execute provided Python script.

Launch Gradio app → interact via generated URL.

4. GitHub Deployment

Create repo → upload .py code → commit changes.

5. Authentication

Hugging Face: Requires user account for accessing IBM Granite models.

Google Colab: Login with Google account to run GPU-backed notebooks.

GitHub: Authentication for project hosting and collaboration.

(Optional: If using IBM Watsonx API, API key authentication may be required.)

6. User Interface

Gradio-based Chatbot:

Input box for user queries.

Output area for AI responses.

Dashboard (Future Scope):

Visualizes sentiment analysis results.

Provides actionable insights for officials.

7. Testing

Functional Testing

Verify chatbot response quality.

Test sentiment analysis accuracy.

Performance Testing

Evaluate model response speed on Colab GPU.

Integration Testing

Ensure Gradio UI works seamlessly with Granite model.

User Testing

Collect feedback from sample users.

8. Known Issues

Limited GPU session time on Google Colab (12-hour cap).

Requires stable internet connection.

Dependency on Hugging Face model availability.

Currently lacks advanced error handling.

Dashboard not fully integrated in this version.

9. Future Enhancements

Add multi-language support for inclusivity.

Develop mobile-friendly interface.

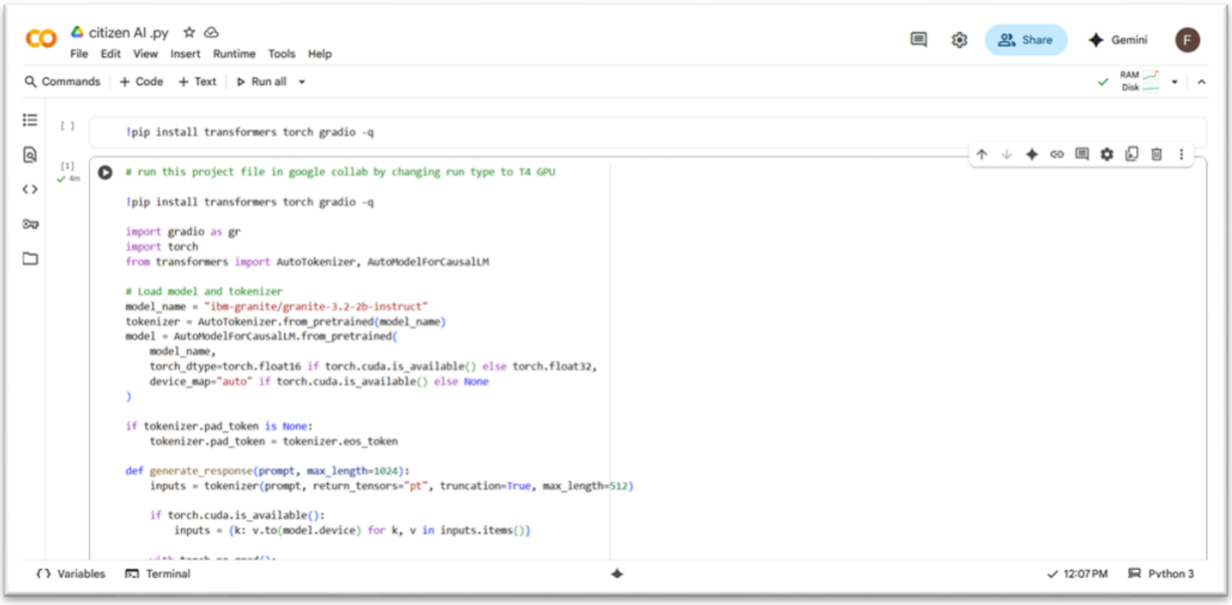
Enable offline deployment (via Docker or local hosting).

Expand dashboard with advanced data visualizations.

Integrate Watsonx API for enterprise-grade scalability.

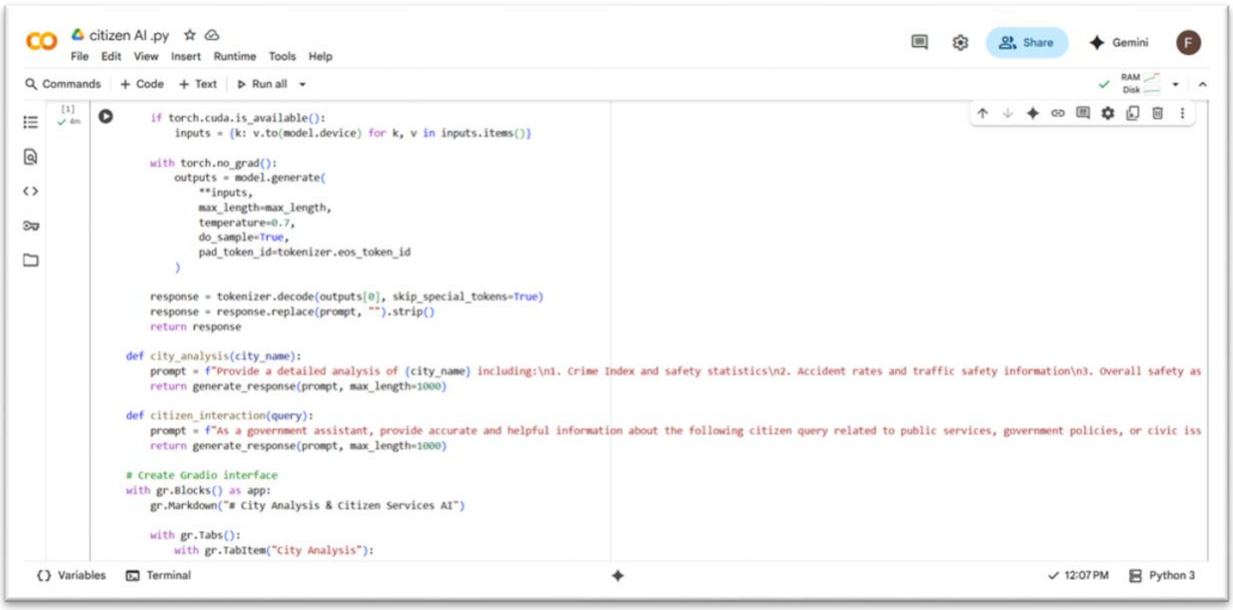
Improve authentication and access control for secure usage.

10.Project Screenshort

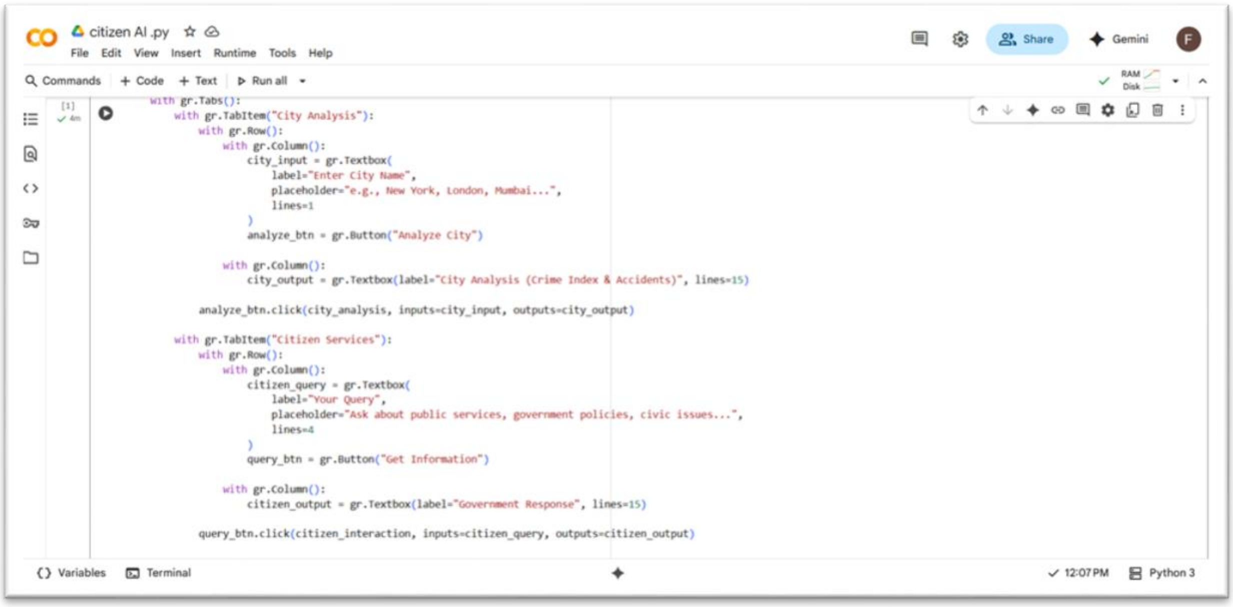


```
!pip install transformers torch gradio -q
```

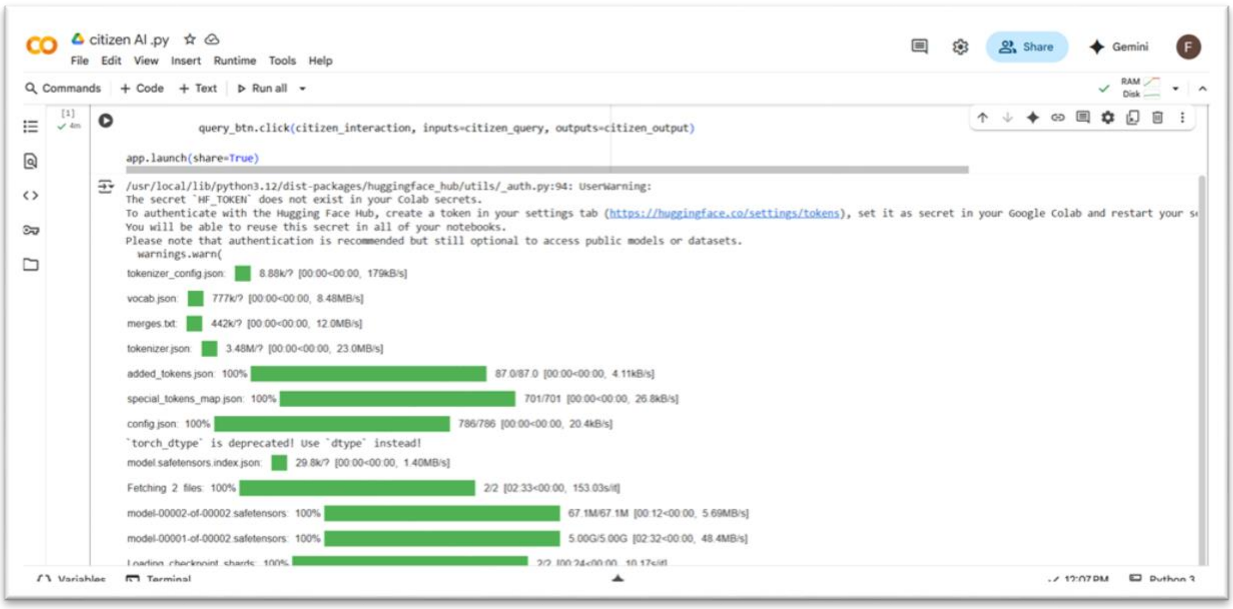
```
# run this project file in google collab by changing run type to T4 GPU

!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()}
```



```
        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 statistics\n2. Accident rates and traffic safety information\n3. Overall safety as
        return generate_response(prompt, max_length=1000)

    def citizen_interaction(query):
        prompt = f"As a government assistant, provide accurate and helpful information about the following citizen query related to public services, government policies, or civic iss
        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"):
```

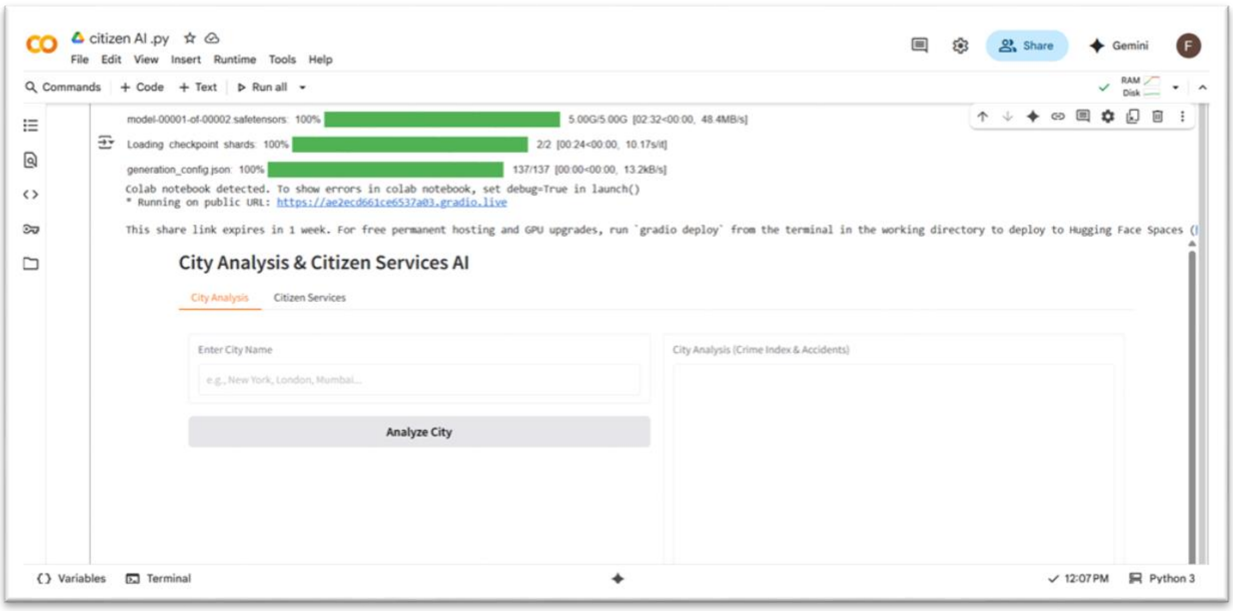


```
        with gr.Tabs():
            with gr.TabItem("City Analysis"):
                with gr.Row():
                    with gr.Column():
                        city_input = gr.Textbox(
                            label="Enter City Name",
                            placeholder="e.g., New York, London, Mumbai...",
                            lines=1
                        )
                        analyze_btn = gr.Button("Analyze City")

                    with gr.Column():
                        city_output = gr.Textbox(label="City Analysis (Crime Index & Accidents)", lines=15)

                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)
```


citizen AI .py
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share
Gemini
RAM Disk
Variables
Terminal
12:07 PM
Python 3

```
    query_btn.click(citizen_interaction, inputs=citizen_query, outputs=citizen_output)

app.launch(share=True)
```

```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret `HF_TOKEN` does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in your Google Colab and restart your s
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
  warnings.warn(
tokenizer_config.json:  8.88k/? [00:00<00:00, 179kB/s]
vocab.json:  777k/? [00:00<00:00, 8.48MB/s]
merges.txt:  442k/? [00:00<00:00, 12.0MB/s]
tokenizer.json:  3.48M/? [00:00<00:00, 23.0MB/s]
added_tokens.json: 100%  87.0/87.0 [00:00<00:00, 4.11kB/s]
special_tokens_map.json: 100%  701/701 [00:00<00:00, 26.8kB/s]
config.json: 100%  786/786 [00:00<00:00, 20.4kB/s]
`torch_dtype` is deprecated! Use `dtype` instead!
model.safetensors.index.json:  29.8k/? [00:00<00:00, 1.40MB/s]
Fetching 2 files: 100%  2/2 [02:33<00:00, 153.03s/it]
model-00002-of-00002.safetensors: 100%  67.1M/67.1M [00:12<00:00, 5.69MB/s]
model-00001-of-00002.safetensors: 100%  5.00G/5.00G [02:32<00:00, 48.4MB/s]
```


citizen AI .py
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
Share
Gemini
RAM Disk
City Analysis
Citizen Services
Enter City Name
e.g., New York, London, Mumbai...
Analyze City
City Analysis (Crime Index & Accidents)
Variables
Terminal
12:07 PM
Python 3

```
model-00001-of-00002.safetensors: 100%  5.00G/5.00G [02:32<00:00, 48.4MB/s]
Loading checkpoint shards: 100%  2/2 [00:24<00:00, 10.17s/it]
generation_config.json: 100%  137/137 [00:00<00:00, 13.2kB/s]
Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://ae2ecd661ce6537a03.gradio.live

This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the working directory to deploy to Hugging Face Spaces (
```

City Analysis & Citizen Services AI

11.Conclusion

The Citizen AI: Intelligent Citizen Engagement Platform project was successfully done by H.Femeena ,R.Harini , Harini.R , R.Kavipriya.

12.Demo video link

https://drive.google.com/file/d/1ljjrF7LLVlMT8_TWl0HYGwLYCFA2sTeB/view?usp=sharing