**CITIZEN AI**

PROJECT DOCUMENTATION

**Introduction**

* Project Title:Citizen AI:Intelligent Citizen Engagement Platform
* Team Leader:Ayswariya R
* Team Member: Janani B
* Team Member: Dhana lakshmi R
* Team Member: Pavithra P

**Project Overview**

**Purpose:**

* Analyzing **city safety metrics** such as crime index and traffic accident rates.
* Answering **citizen queries** about public services, policies, and civic issues.
* Providing responses in a **chatbot-like interface** using Gradio.

This project leverages **IBM Granite AI models** (from Hugging Face) and is deployed on **Google Colab** with a simple and interactive **Gradio web app**.

**Objectives:**

* Provide citizens with **quick and accurate** information about civic services.
* Help governments track and analyze **public safety concerns**.
* Build a **low-cost, easy-to-use AI platform** with open-source tools.

### Features:

1. **City Analysis Tab**
   * Input: City name
   * Output: Crime statistics, accident rates, safety report
2. **Citizen Services Tab**
   * Input: Citizen query (e.g., government policies, schemes, services)
   * Output: AI-generated, helpful response User asks a question related to public services, government policies, or civic issues.

 **How it Works (Under the Hood):**

* The citizen\_interaction() function crafts a prompt that sets the AI's persona: "As a government assistant, provide accurate and helpful information about the following citizen query..."
* This "role-playing" instruction helps the AI generate relevant and authoritative responses

### Architecture:

The application follows a simple but effective client-server architecture, enabled by the Gradio library. It consists of three main components:

**Frontend (User Interface):** This is the part the user sees and interacts with. Gradio dynamically generates a web interface directly from the Python code. It handles user inputs (text from textboxes), triggers events (button clicks), and displays the final output. The UI is automatically made responsive.

**Backend (Core Logic):** This is your Python script. It acts as the server, hosting the Gradio application. Its primary functions are:

**Setup Instructions:**

To run this application, you need to set up a Python environment with the required libraries.

**Prerequisites:** Ensure you have Python installed on your machine.

**Install Libraries:** Open your terminal or command prompt and run the following command. The -q flag keeps the installation output quiet.

**Save the Code:** Save the provided Python code into a single file, for example, citizen\_ai.py.

* 1. Executing the corresponding Python function (city\_analysis or citizen\_interaction).
  2. Preparing a structured prompt for the AI model.
  3. Passing the prompt to the AI model and receiving its generated response.
  4. Sending the AI's response back to the frontend for display.
* **Model (AI Brain):** This is the large language model (ibm-granite/granite-3.2-2b-instruct). It's the "brain" of the application, responsible for generating the intelligent, human-like responses to the user's queries. The transformers library handles the complex process of downloading, loading, and running this model.

**Folder Structure:**

* + - generate\_response(): Generates AI responses
  + - city\_analysis(): Provides crime index & safety data for a city
  + - citizen\_interaction(): Answers public service & policy queries
  + Python
  + - Hugging Face Transformers
  + - IBM Granite Models
  + - Torch (PyTorch)
  + - Gradio for User Interface
  + - Google Colab & GitHub

**Running the Application**

* Once the dependencies are installed and the code is saved, running the application is a single command.
* Bash
* python citizen\_ai.py
* This command will start a local web server, and a URL will be printed to your terminal. You can open this URL in your web browser to use the application. The share=True parameter in app.launch() also generates a public, shareable URL, allowing others to access your application temporarily.

### Authentication: The current application does not have any built-in authentication. Anyone who can access the generated URL will be able to use the application without logging in or providing any credentials. For a production environment, you would need to implement an authentication layer using a framework like Flask or FastAPI in conjunction with Gradio to secure access.

### User Interface Details:

The user interface is entirely defined within the gr.Blocks() context using Gradio components.

* **gr.Blocks():** This acts as the main container for the entire application, providing a flexible layout.
* **gr.Markdown():** Used to display the main title, "City Analysis & Citizen Services AI," in a clear, formatted way.
* **gr.Tabs():** This component organizes the application into distinct, clickable tabs.
* **gr.TabItem("City Analysis") and gr.TabItem("Citizen Services"):** These define the content for each tab, containing the specific components for each use case.
* **gr.Row() and gr.Column():** These are layout components that help arrange the textboxes and buttons horizontally and vertically, creating a clean and organized layout.
* **gr.Textbox():** This component is used for both input and output.
  + For input, it has a label and a placeholder.
  + For output, it has a label and a fixed number of lines to display the response.
* gr.Button(): This is the interactive element that a user clicks to trigger a function. Its click() method is crucial, as it links the button to the Python function (city\_analysis or citizen\_interaction) and specifies which components are used for input and output.

### Testing:

A basic testing plan for this application would involve verifying the functionality of both tabs.

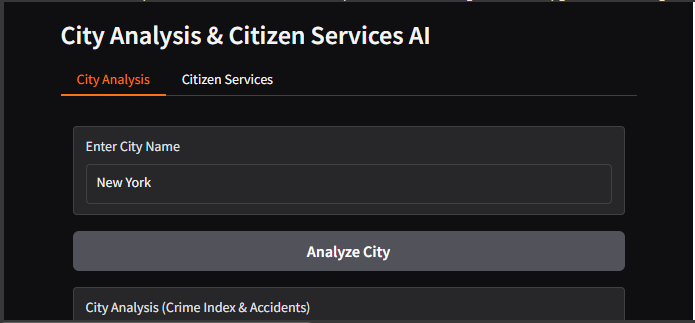
#### Test Case 1: City Analysis

* **Input:** Enter a city name into the "Enter City Name" textbox (e.g., "Paris" or "Tokyo").
* **Steps:** Click the "Analyze City" button.
* **Expected Behavior:** The application should display a detailed analysis of the city's crime, accident rates, and overall safety assessment in the output textbox. The response should be well-structured and relevant to the city provided.

#### Test Case 2: Citizen Services

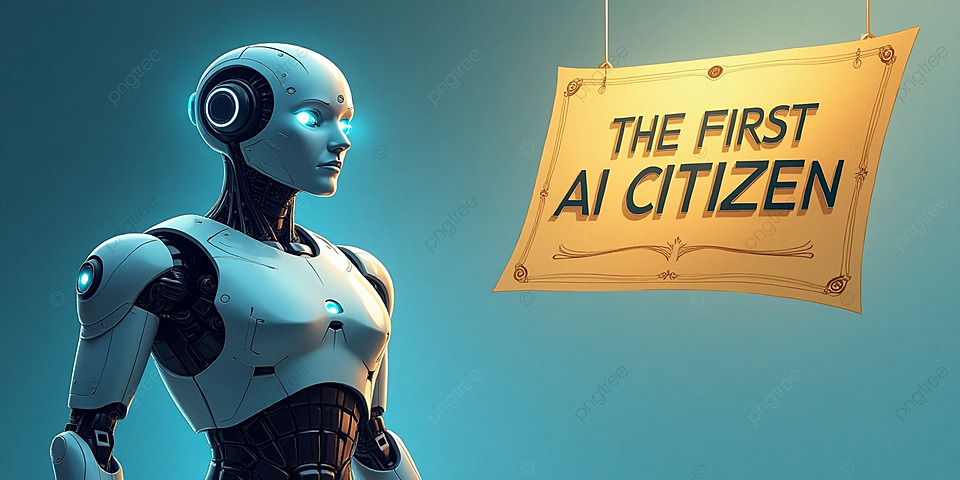
* **Input:** Enter a public service query into the "Your Query" textbox (e.g., "How do I register to vote in New York?").
* **Steps:** Click the "Get Information" button.
* **Expected Behavior:** The application should provide a helpful and accurate response to the query, written from the perspective of a government assistant. The response should be polite and informative.

**Output:**



**Conclusion:**

* Citizen AI leverages IBM Granite models & Gradio to create a smart assistant
* That empowers citizens with city safety analysis and government information. This project demonstrates AI for better governance and civic engagement.



**THANK YOU**