Citizen AI

Intelligent Citizen Engagement Platform

1. INTRODUCTION

- **Project Title : Citizen AI - Intelligent Citizen Engagement Platform**
- **Team Member : Maghalaxmi V**
- **Team Member : Sivagami S**
- **Team Member : Thrisha P**
- **Team Member : Anusuya R**

2. PROJECT OVERVIEW

- **Purpose :**

The purpose of Citizen AI is to enhance the way citizens interact with government services by providing a seamless, intelligent, and accessible digital interface. By leveraging AI-driven conversational systems and real-time data, the platform enables residents to easily access public information, lodge requests, and receive personalized guidance for civic services. For government officials, it acts as a powerful decision support tool offering actionable insights, analytics, and automated summaries of citizen feedback to improve transparency and service delivery. Ultimately, Citizen AI bridges the gap between governance and community by fostering trust, inclusivity, and efficiency in citizen engagement, creating a more responsive and participatory society.

- **Features :**

Conversational Interface :

Key Point : Natural language interaction

Functionality : Enables citizens to ask queries, access services, and receive updates in plain, everyday language.

Policy Summarization :

Key Point : Simplified governance understanding

Functionality : Transforms lengthy government policies and documents into clear, concise, and citizen-friendly summaries.

Service Request Handling :

Key Point : Streamlined civic issue reporting

Functionality : Allows citizens to lodge complaints, request services, and track progress in real time.

Decision Support for Officials :

Key Point : Data-driven governance

Functionality : Provides analytics, citizen sentiment insights, and automated reports to assist officials in making informed decisions.

Citizen Feedback Loop :

Key Point : Community participation

Functionality : Collects, analyses, and visualizes public input to improve policies, projects, and service delivery.

Information Accessibility :

Key Point : Inclusive access

Functionality : Offers multilingual support and accessibility features to ensure all citizens can use the platform effectively.

Anomaly Detection :

Key Point : Early issue identification

Functionality : Detects irregularities in citizen complaints, requests, or service usage patterns to flag emerging problems.

Multimodal Input Support :

Key Point : Flexible data handling

Functionality : Accepts text, voice, PDFs, and structured data for analysis, ensuring ease of interaction for diverse users.

User Friendly Interface (Streamlit/Gradio) :

Key Point : Intuitive experience

Functionality : Provides a clean, interactive dashboard for both citizens and officials to engage with AI-driven services.

3. ARCHITECTURE

Frontend :

In this project, the frontend is implemented using Gradio Blocks and Tabs, which provide a lightweight, interactive web interface. The application consists of two main tabs: City Analysis and Citizen Services. Each tab uses input forms such as textboxes for entering a city name or typing a citizen query, along with buttons to trigger analysis or responses. The outputs are displayed in multiline textboxes, making the interface simple and user-friendly for citizens to interact with AI-powered services.

Backend :

The backend logic is embedded in Python, where Flask is not used here, but Gradio handles the web server functionality. The backend defines helper functions like generate_response(), city_analysis(), and citizen_interaction(). These functions manage the interaction between user inputs and the AI model, handle text generation, and format outputs. Gradio connects these backend functions to the frontend interface components (buttons and textboxes) to provide end-to-end functionality.

LLM Integration :

At the core of the system lies the IBM Granite model, integrated through Hugging Face's transformers library. The code loads the model and tokenizer (AutoTokenizer and AutoModelForCausalLM) with GPU support for efficient inference. The function generate_response() uses the Granite model to generate contextually accurate answers by applying decoding strategies such as temperature, sampling, and maximum length. This ensures that responses are human-like and relevant to queries.

ML Modules :

The existing code demonstrates simple AI functionalities like city analysis covering crime index, accident rates, and safety and citizen interaction answering questions about policies or services. These represent domain-specific ML tasks driven by prompt engineering. Additional ML modules like sentiment analysis, categorization of citizen concerns, and dashboard analytics could be layered on top to enrich insights and enhance decision-making for policymakers.

4. SETUP INSTRUCTIONS

Prerequisites :

- Python 3.9 or later
- pip and virtual environment tools
- Hugging Face Transformers, PyTorch, and Gradio libraries
- GPU with CUDA support
- Internet access to download IBM Granite model from Hugging Face

Installation Process :

- Clone the repository
- Install dependencies from requirements.txt
- Create a virtual environment and activate it
- Run the application using python Citizen AI.py
- Access the Gradio interface from the local URL
- Enter queries or city names and interact with the modules

5. FOLDER STRUCTURE

- app/ – Contains the main Gradio application logic, including response functions and interface setup.
- app/modules/ – Subdirectory for modular Python scripts like city analysis, citizen interaction, and helper functions.
- ui/ – Holds frontend assets such as CSS stylesheets, images, or future UI customizations.
- citizen_ai.py – Entry script for launching the Gradio interface with City Analysis and Citizen Services tabs.
- granite_llm.py – Handles all communication with the IBM Granite model, including loading, tokenization, and text generation.
- city_analysis.py – Provides detailed safety and accident analysis for a given city.
- citizen_services.py – Manages AI-powered responses for public service and policy-related citizen queries.
- requirements.txt – Lists all dependencies (torch, transformers, gradio) for easy installation.
- README.md – Documentation file explaining setup, usage, and project overview.

6. RUNNING THE APPLICATION

- To start the project : Run the Gradio app script using : python Citizen AI.py
- The Gradio server will launch automatically and display a local URL in the terminal.
- Open the URL in your browser to access the web interface.
- Navigate between the two tabs – City Analysis and Citizen Services.
- In the City Analysis tab, enter a city name to generate a detailed safety and accident analysis.
- In the Citizen Services tab, type a public service or policy-related query to receive AI-generated responses.
- All responses are generated in real-time using the IBM Granite model integrated through Hugging Face.

Frontend (Gradio) :

The frontend is built using Gradio Blocks and Tabs, which provide an interactive UI for users. It includes multiple pages (tabs) such as City Analysis and Citizen Services. Users can enter text inputs, submit queries via buttons, and view responses in real-time. The layout uses rows and columns for clear organization, ensuring a simple and user-friendly interface.

Backend (Python + Hugging Face Transformers) :

The backend is implemented in Python, where Gradio also handles the web server logic. Functions like city_analysis() and citizen_interaction() process user inputs and forward them to the IBM Granite LLM. The backend manages model loading, tokenization, text generation, and returning formatted outputs. It ensures smooth integration between the UI and AI engine, providing instant feedback to the user.

7. AUTHENTICATION

This version of the project runs in an open environment for demonstration. However, secure deployments can integrate :

- Basic username/password authentication using Gradio's built-in auth parameter
- Token-based authentication (JWT or API keys) for API-style access
- OAuth2 integration with IBM Cloud or third-party identity providers
- Role-based access (admin, citizen, analyst) for controlled usage
- Planned enhancements include user sessions and chat history tracking

8. USER INTERFACE

The interface is minimalist and functional, focusing on accessibility for non-technical users. It includes :

- Tabbed layout with “City Analysis” and “Citizen Services” for easy navigation
- Textboxes and buttons for submitting queries and city names
- Real-time response display in output textboxes
- Interactive form handling with instant AI-generated feedback
- Clear guidance and placeholders in input fields to help users enter queries correctly

The design prioritizes clarity, speed, and user guidance, ensuring a smooth experience for all users interacting with the AI-powered system.

9. TESTING

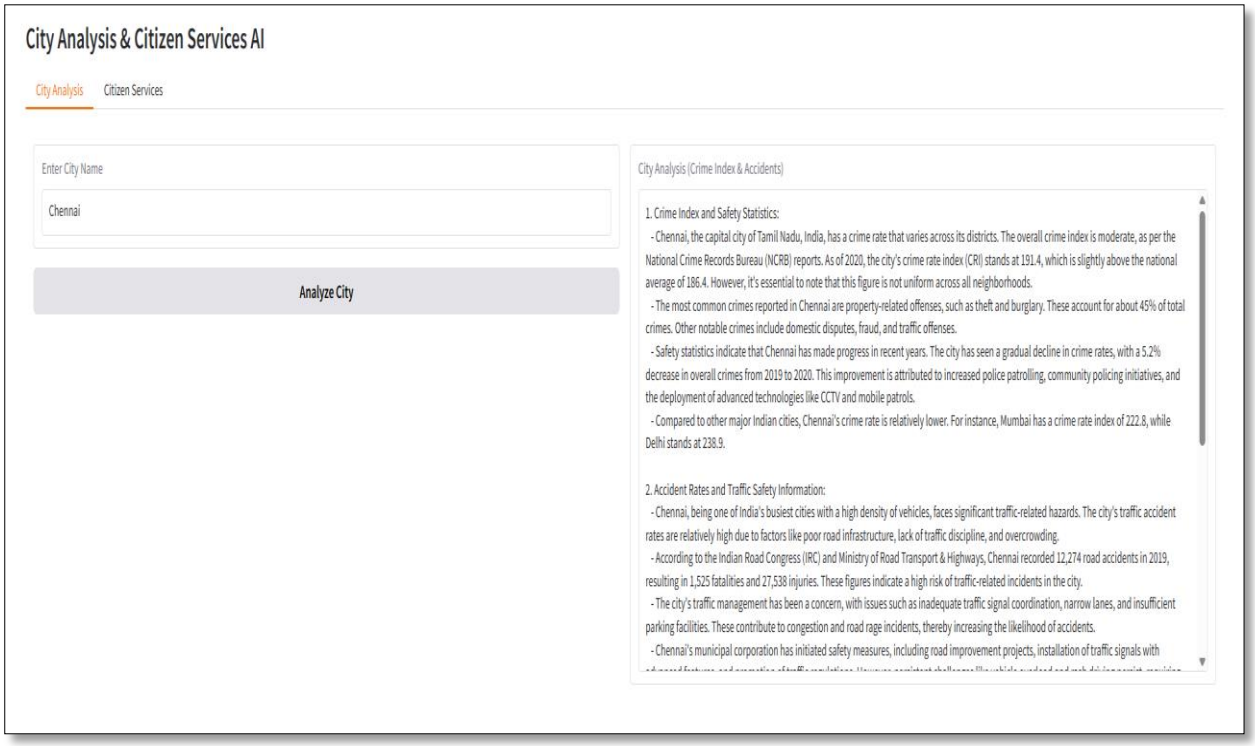
Testing was done in multiple phases :

- Unit Testing : For functions like generate_response(), city_analysis(), and citizen_interaction() to ensure correct output generation
- Interface Testing : Checking Gradio input/output components, button actions, and tab navigation
- Manual Testing : For entering city names, submitting citizen queries, and verifying AI-generated responses
- Edge Case Handling : Malformed queries, empty inputs, very long text, and unexpected characters

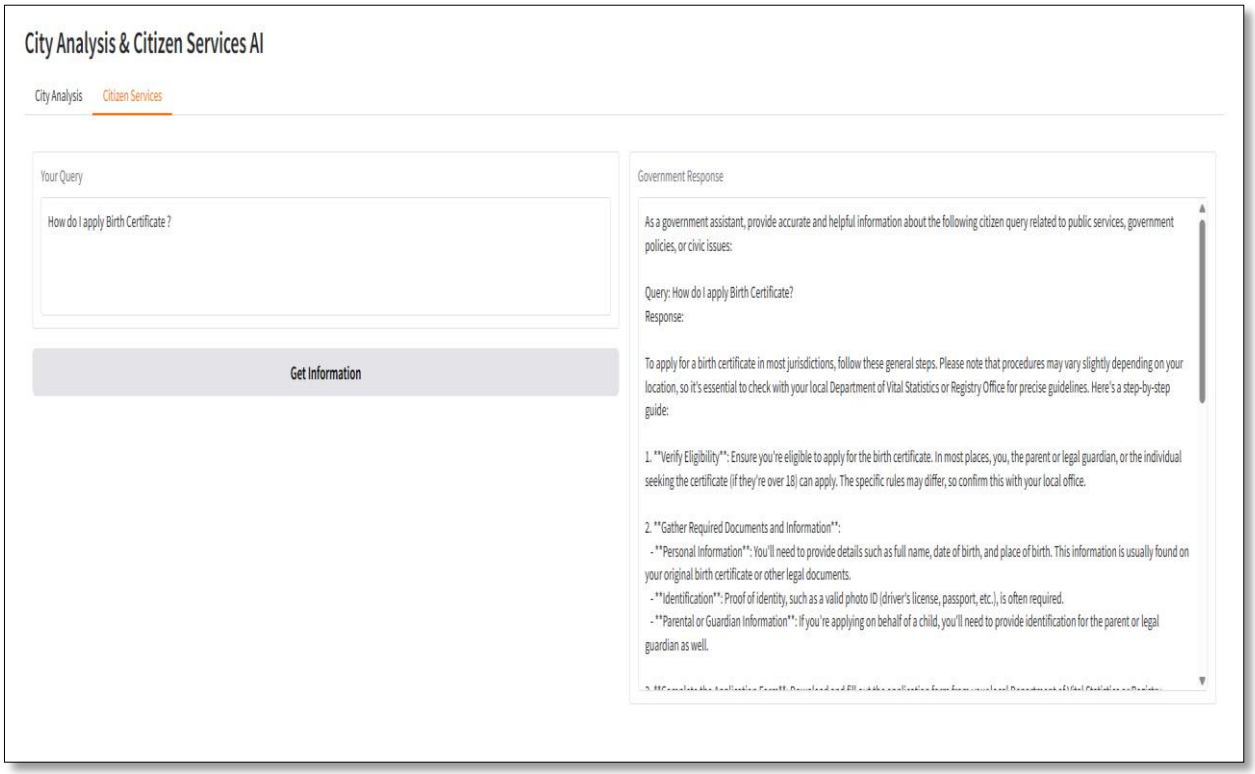
Each function and interaction was validated to ensure reliability, responsiveness, and consistency in the real-time Gradio interface.

10. SCREEN SHOTS

- **City Analysis :**



- **Citizen Services :**



11. KNOWN ISSUES

- Large inputs may cause slower response times due to model inference.
- The app currently has no user authentication; anyone with the link can access it.
- No persistent chat history or session tracking is implemented.
- Extremely long or malformed queries can sometimes produce incomplete responses.
- Vector search or knowledge grounding is not implemented, so all responses rely solely on the LLM.

12. FUTURE ENHANCEMENT

- Add user authentication and role-based access for secure usage.
- Implement persistent chat history and session tracking.
- Integrate vector search with FAISS or Pinecone for fact-grounded responses.
- Improve UI/UX, including dashboards, summary cards, and visual analytics.
- Add file upload support for documents or datasets to enhance city analysis.
- Optimize model inference speed for large inputs.
- Include analytics modules like sentiment analysis and citizen query categorization.