**Citizen AI Intelligent Citizen Engagement Platform**

1. **Introduction**

* • **Project title** : Citizen AI Platform
* • **Team member** : K.ARUN

* • **Team member** : S.ARIKARAN
* • **Team membe**r : M.ANBARASAN
* • **Team membe**r : G.AKASH
* • **Team membe**r :S.AJAY

1. **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.

1. **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").

1. **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.

1. **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.

1. **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

1. **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.

1. **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.

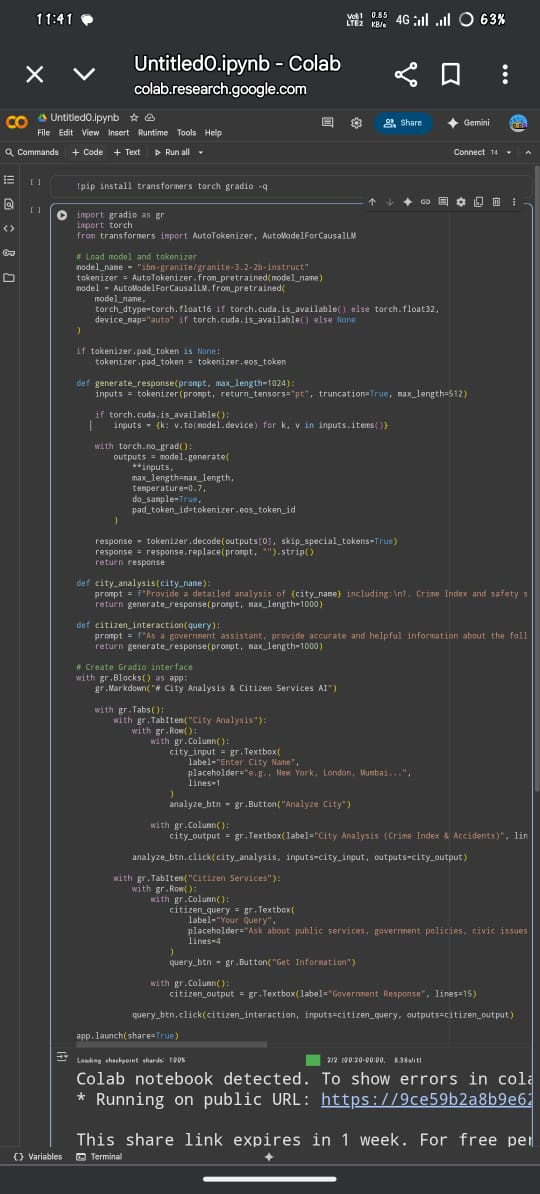

* Enter an invalid/empty city → Graceful error message.


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

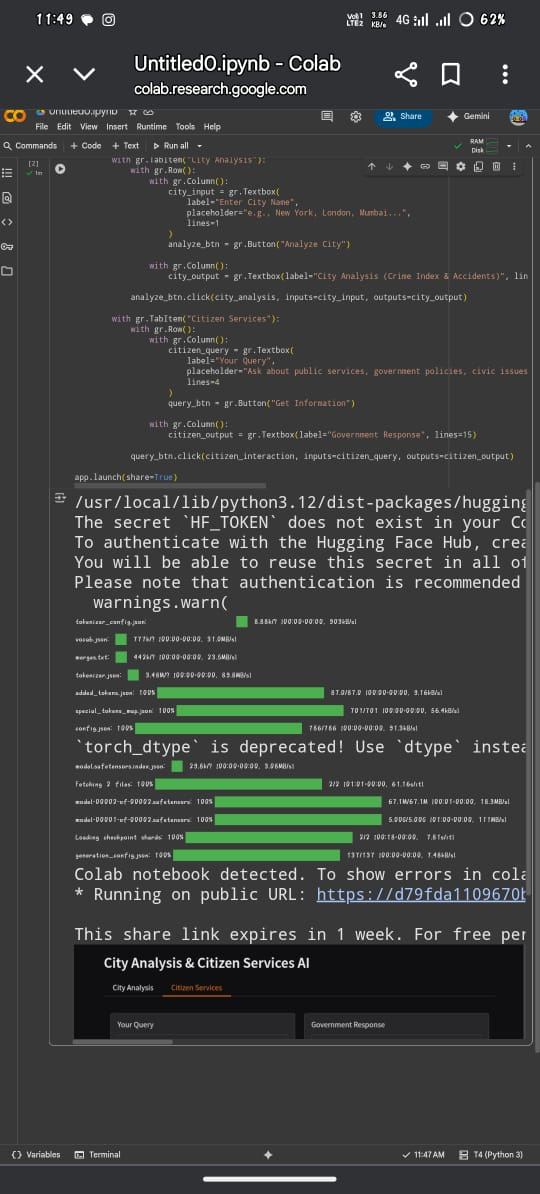

* Ask irrelevant query (e.g., favorite color) → AI gives fallback response.

1. **screen shots**





* **Input**



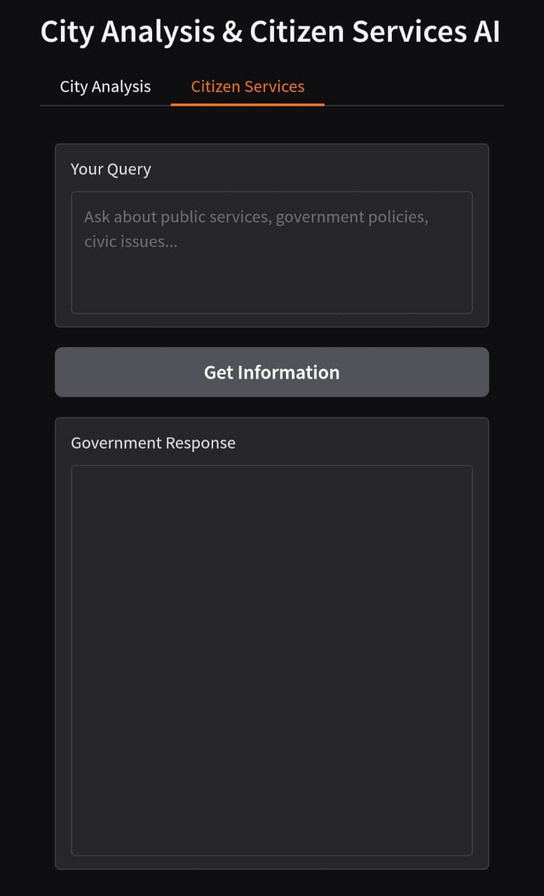
* **Output**













1. **Known Issues**

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

1. **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