# CITIZEN AI:INTELLIGENT CITIZEN ENGAGEMENT PLATFORM

## **Project Documentation**

#### 1. Introduction:

Project title: Citizen Ai:Intelligent Citizen Engagement Platform.

Team leader: SHADIK BASHA.H

Team member :KRISHNA KUMAR.R

• Team member : DINESH.G

• Team member :JEROME.J

## 2. Project Overview:

• Purpose: The goal of the Citizen AI Project is to empower city residents by leveraging AI and real-time data to create a more eco-conscious and connected urban environment. It helps optimize resources like energy, water, and waste, and provides personalized eco-tips to encourage sustainable behaviours among citizens. For city officials, the project serves as a decision-making tool by providing insights, forecasting capabilities, and summaries of complex policies. The project aims to connect technology, governance, and community to build more efficient, resilient, and greener cities.

#### • Features:

- o **Conversational Interface:** This allows for natural language interaction, enabling citizens and officials to ask questions and receive guidance.
- Policy Summarization: Converts long government documents into clear, actionable summaries for easier understanding.
- **Resource Forecasting:** Uses historical and real-time data to predict future usage of energy, water, and waste.
- **Eco-Tip Generator:** Recommends daily actions to help users reduce their environmental impact based on their behaviour.
- Citizen Feedback Loop: Gathers and analyzes public input to assist with city planning and service enhancements.

- o **KPI Forecasting:** Projects key performance indicators to help officials monitor progress and plan strategically.
- o **Anomaly Detection:** Acts as an early warning system by identifying unusual patterns in sensor or usage data to flag potential issues.
- o **Multimodal Input Support:** Can handle different data types, including text, PDFs, and CSVs, for analysis and forecasting.
- o **User-friendly Interface:** An intuitive dashboard built with Streamlit or Gradio UI that allows both citizens and city officials to easily interact with the assistant.

#### 3. Architecture:

- Frontend (Streamlit): The frontend is an interactive web UI with multiple pages for dashboards, file uploads, a chat interface, feedback forms, and report viewers. It uses the Streamlit-option-menu library for sidebar navigation, and each page is modularized for scalability.
- **Backend (FastAPI):** This serves as the REST framework for API endpoints that handle document processing, chat, eco-tip generation, and more. It is optimized for asynchronous performance and easy Swagger integration.
- LLM Integration (IBM Watsonx Granite): The project uses Granite LLM models from IBM Watsonx for natural language understanding and generation. Prompts are specifically designed to produce summaries, reports, and sustainability tips.
- Vector Search (Pinecone): Uploaded policy documents are converted into embeddings using Sentence Transformers and stored in Pinecone. Semantic search is enabled via cosine similarity, letting users search documents using natural language queries.
- ML Modules (Forecasting and Anomaly Detection): Lightweight ML models from Scikit-learn are used for forecasting and anomaly detection. Time-series data is parsed, modeled, and visualized using pandas and matplotlib.

## 4. Setup Instructions:

#### · Prerequisites:

- Python 3.9 or later ∘ pip and virtual environment tools ∘ API keys for IBM Watsonx and Pinecone ∘ Internet access for cloud services
   □ Installation Process:
- Clone the repository.
- o Install dependencies from requirements.txt.

- Create and configure a .env file with credentials.
- Run the backend server using FastAPI.
- Launch the frontend via Streamlit.
   Upload data and interact with the modules.

#### 5. Folder Structure:

- app/ Contains all FastAPI backend logic, including routers, models, and integration modules.
- app/api/ Subdirectory for modular API routes like chat, feedback, and document vectorization.
- ui/ Contains frontend components for Streamlit pages and form UIs.
- smart dashboard.py The entry script for the main Streamlit dashboard.
- granite\_llm.py Handles all communication with the IBM Watsonx Granite model.
- document\_embedder.py Converts documents to embeddings and stores them in Pinecone.
- kpi\_file\_forecaster.py Forecasts future trends for energy/water using regression.
- anomaly\_file\_checker.py Flags unusual values in uploaded KPI data.
- report\_generator.py Constructs AI-generated sustainability reports.

## 6. Running the Application:

- To start the project, launch the FastAPI server and then run the Streamlit dashboard.
- Navigate through the pages using the sidebar.
- Users can upload documents or CSVs, interact with the chat assistant, and view outputs like reports, summaries, and predictions.
- All interactions are real-time, with the frontend dynamically updating via backend APIs.

## 7. API Documentation:

- The backend APIs include:
  - POST /chat/ask Accepts a user query and returns an AI-generated message.
  - POST /upload-doc Uploads and embeds documents in Pinecone.

- GET /search-docs Returns semantically similar policies to a user query.
- GET /get-eco-tips Provides sustainability tips on selected topics.
   POST /submit-feedback Stores citizen feedback.
- Each endpoint is documented and tested in Swagger UI.

#### 8. Authentication:

- For demonstration purposes, this version of the project runs in an open environment.
- Secure deployments can include:
  - Token-based authentication (JWT or API keys). OAuth2 with IBM Cloud credentials.
  - o Role-based access for different user types (admin, citizen, researcher).
- Future enhancements will include user sessions and history tracking.

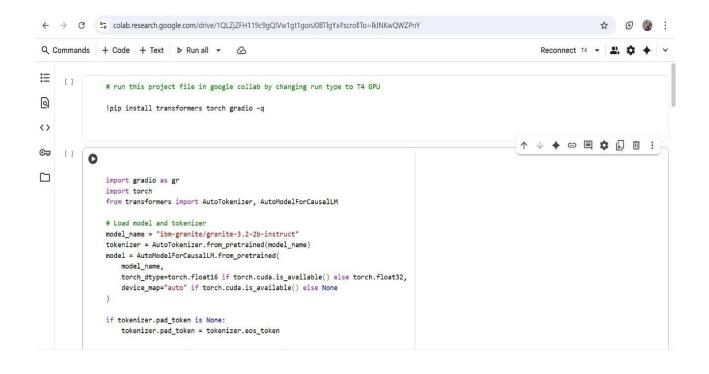
#### 9. User Interface:

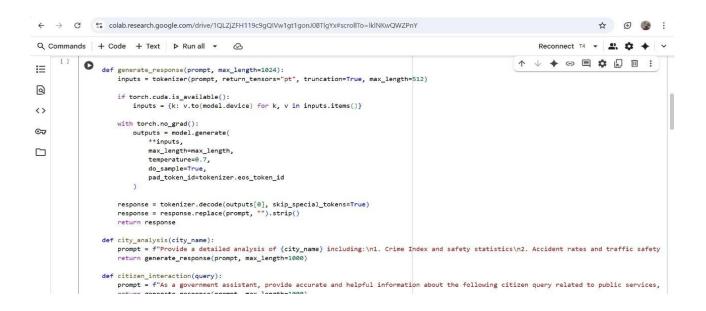
- The interface is minimalist and designed for accessibility for nontechnical users.
- Key elements include:
  - A sidebar for navigation.
     KPI visualizations with summary cards.
     Tabbed layouts for chat, eco tips, and forecasting.
     Real-time
    - form handling. PDF report download capability.

## 10. Testing:

- Testing was conducted in several phases:
  - o Unit Testing: For prompt engineering functions and utility scripts.
    - o **API Testing:** Done via Swagger UI, Postman, and test scripts. o **Manual Testing:** To validate file uploads, chat responses, and output consistency.
  - Edge Case Handling: To address malformed inputs, large files, and invalid API keys.
- Each function was validated to ensure reliability in both offline and APIconnected modes.

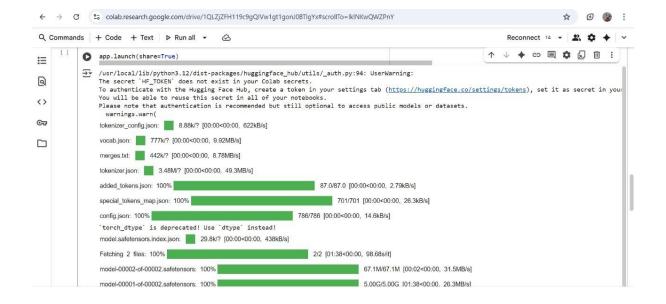
#### 11. Source Code Screenshots:

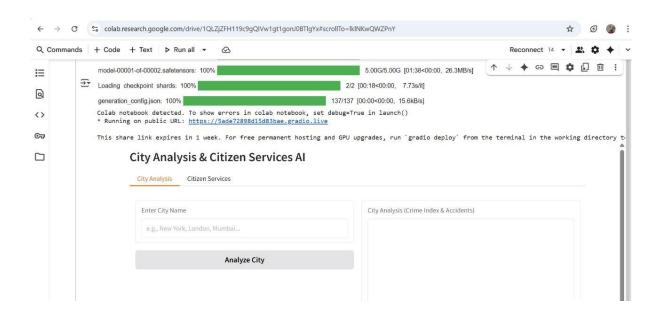




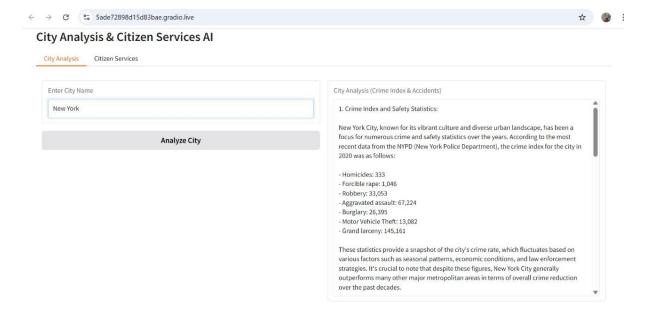
```
☆ ⑤ ③ :
← → C º= colab.research.google.com/drive/1QLZjZFH119c9gQIVw1gt1gonJ0BTlgYx#scrollTo=lklNKwQWZPnY
                                                                                                                                        Reconnect T4 ▼ 😩 🌣 ♦ ∨
Q Commands + Code + Text ▶ Run all ▼ 🙆
                                                                                                                                  ↑ ↓ ♦ 🗇 🗏 🛱 📔 🔡
                      return generate_response(prompt, max_length=1000)
∷
                  # Create Gradio interface
0
                  with gr.Blocks() as app:
gr.Markdown("# City Analysis & Citizen Services AI")
<>
                      with gr.Tabs():
                          with gr.TabItem("City Analysis"):
<del>От</del>
                              with gr.Row():
with gr.Column():
city_input = gr.Textbox(
    label="Enter City Name",
    placeholder="e.g., New York, London, Mumbai...",
                                       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(
```

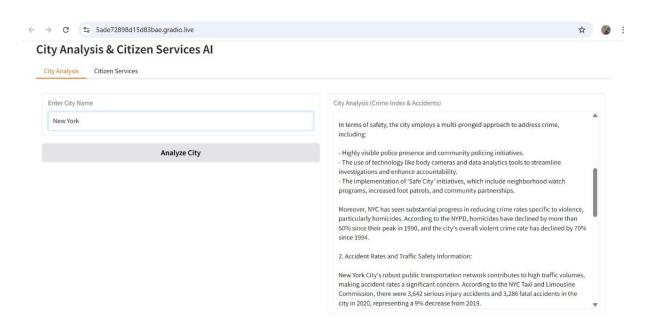
```
← → C 25 colab.research.google.com/drive/1QLZjZFH119c9gQIVw1gt1gonJ0BTlgYx#scrollTo=lklNKwQWZPnY
                                                                                                                                            ☆ ⑤ 🚷 :
Reconnect T4 ▼ 😩 🌣 💠 ∨
                                                                                                                        ↑ ↓ ♦ 🖘 🗏 🔟 :
            0
                                        placeholder="e.g., New York, London, Mumbai...",
∷
                                       lines=1
0
                                    analyze_btn = gr.Button("Analyze City")
<>
                                    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...",
                                    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)
```

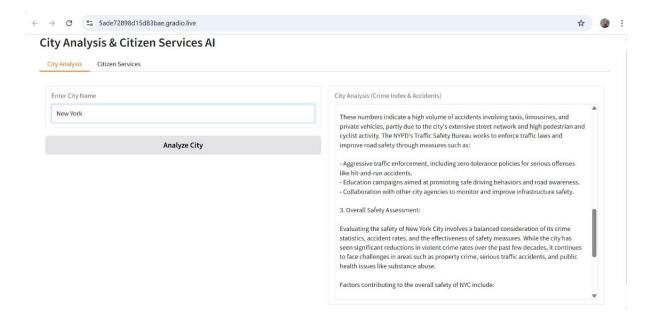


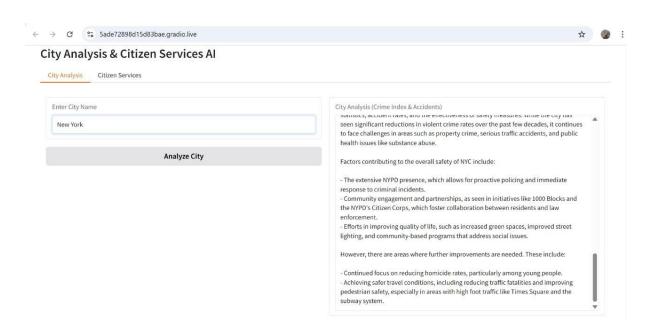


## 12. Source Output:









## 13. Future Enhancements:

- User Sessions and History Tracking: The project plans to add the ability to track user sessions and interaction history. This will allow for a more personalized experience.
- **Security:** For secure deployments, the project can integrate token-based authentication (JWT or API keys), OAuth2 with IBM Cloud credentials, and role-based access for different users (e.g., admin, citizen, researcher).