Citizen Analysis and citizen services AI

Project Documentation

1.Introduction

- Citizen AnalysisI and citizen services AI
- S.Kavya
- S.Kavitha Shree
- P.Keerthana
- M.Karthika

2.Project overview

●purpose

The purpose of this code is to build an AI-powered City Analysis and Citizen Services platform using Hugging Face's ibm-granite-3.2-2b-instruct model and Gradio. It allows users to analyze cities by generating detailed reports on crime index, accident rates, and overall safety, as well as interact with a virtual government assistant that provides helpful information about public services, government policies, and civic issues. The application leverages GPU acceleration (T4 in Google Colab) for faster model inference, processes user inputs through the tokenizer and model, and displays AI-generated responses through an interactive web interface with two tabs—City Analysis and Citizen Services.

●Features

This code features an interactive Gradio-based web app with two main functionalities: (1) City Analysis, which generates AI-driven insights about a city's crime index, accident rates, traffic safety, and overall safety assessment, and (2) Citizen Services, which provides government-related information in response to citizen queries about public services, policies, and civic issues. It integrates the ibm-granite-3.2-2b-instruct model for natural language understanding, supports GPU acceleration (T4 in Google Colab) for faster response generation, ensures proper tokenization and padding, and delivers results through a clean tabbed interface that allows easy switching between the two services.

- **Conversational Interface**

key point: Enables natural interaction with AI

Functionality: Provides responses to city analysis and citizen queries

Policy Summarization

Key point: Simplifies complex policies

Functionality: Generates concise and easy-to-understand summaries

- **Resource Forecasting**

Key point: Predicts future needs

Functionality: Analyzes data to forecast resource requirements

Eco-Tip Generator

Key point: Promotes sustainability

Functionality: Provides daily eco-friendly suggestions for citizens

- **Citizen Feedback Loop**

Key point: Enhances public engagement

Functionality: Collects and responds to citizen feedback effectively

- **KPI Forecasting**

Key point: Predicts key performance indicators

Functionality: Uses data trends to estimate future performance metrics

- **Anomaly Detection**

Key point: Identifies unusual patterns

Functionality: Detects irregularities or unexpected behaviors in data

Multi-modal Input Support

Key point: Handles diverse input formats

Functionality: Accepts text, images, and other data types for analysis

- **Streamlit or Gradio UI**

Key point: Provides user-friendly interface

Functionality: Enables interactive dashboards and real-time responses

3.Architecture

1. **Frontend (Streamlit):**

The frontend is built using Streamlit, which provides a simple yet powerful framework to create interactive dashboards and user-friendly interfaces. It allows users to easily interact with the application, input queries, and view real-time outputs such as forecasts, anomaly detections, and policy summaries in a visually appealing format.

2. **Backend (FastAPI):**

The backend is developed with FastAPI, a fast and modern web framework that handles requests between the frontend and the AI/ML models. It ensures efficient API creation, quick data transfer, and smooth integration of multiple services, enabling high-performance communication between different system components.

3. **LLM Integration (IBM Watsonx Granite):**

The system integrates IBM Watsonx Granite, a large language model (LLM), to process natural language queries and generate intelligent responses. This integration empowers the platform with advanced reasoning, summarization, and conversational capabilities, making it more adaptive and context-aware for government and citizen services.

4. **Vector Search (Pinecone):**

Vector search is powered by Pinecone, which helps in storing and retrieving embeddings for semantic search and similarity matching. It enables the system to quickly find relevant documents, policies, or past interactions, ensuring that users receive accurate, context-rich, and meaningful information.

5. ML Modules (Forecasting and Anomaly Detection):

The ML modules include forecasting and anomaly detection, which are essential for predictive analytics and monitoring. Forecasting helps anticipate future trends such as resource needs or KPI performance, while anomaly detection identifies irregular patterns, ensuring timely alerts and proactive decision-making.

6. Setup Instructions:

To set up the project, open Google Colab, create a new notebook, and ensure the runtime is configured to use a T4 GPU for faster performance. Copy the provided code into the notebook cells, and once executed, the Gradio interface will launch with two tabs: City Analysis and Citizen Services, allowing you to interact with the AI directly.

7. Prerequisites:

Before running the code, you need to have a working Google account to access Colab and an active internet connection for downloading the required model and libraries. Familiarity with Python basics, natural language processing concepts, and Gradio usage will help in understanding and customizing the project.

8. Installation Process:

The installation is handled automatically within the notebook. The required Python packages—transformers, torch, and gradio—are installed using pip. After installing dependencies, the model (ibm-granite/granite-3.2-2b-instruct) and tokenizer are loaded from Hugging Face, ensuring the system is ready to generate responses and serve through the web-based Gradio app.

5.Folder structure

app.py → Main Python script with Gradio app

requirements.txt → Dependencies (transformers, torch, gradio)

notebooks/

city_analysis_colab.ipynb → Google Colab notebook version

config/

settings.py → Configurations (model name, parameters)

modules/

init.py

model_loader.py → Loads tokenizer and model

response_generator.py → Handles AI response generation

city_analysis.py → City analysis functionality

citizen_services.py → Citizen query handling

static/

style.css → Optional custom CSS for Gradio UI

logs/

app.log → Log files (if needed)

README.md → Project description and setup guide

6.Running the application

To run the application, first open Google Colab and create a new notebook.

Next, go to Runtime → Change runtime type → Hardware accelerator → T4 GPU to enable GPU support.

Install the required libraries by running:

!pip install transformers torch gradio -q

Then, copy and paste the provided project code into a new cell and execute it.

Once the code runs, Gradio will launch and provide a shareable link to access the app.

In the interface, use the City Analysis tab to get crime index, accident rates, and safety details, or the Citizen Services tab to get policy and civic information.

When done, stop the notebook execution to free up GPU resources.

7.API Documentation

The application provides three main APIs for interaction. The City Analysis API (city_analysis(city_name)) takes the name of a city as input and returns a detailed report including crime index, accident rates, traffic safety details, and overall safety assessment; for example, calling city_analysis("Mumbai") will generate an AI-based analysis of Mumbai’s safety profile. The Citizen Services API (citizen_interaction(query)) allows users to ask queries about public services, government policies, or civic issues, and it responds with accurate and easy-to-understand information; for instance, citizen_interaction("What are the traffic rules in Mumbai?") will provide a clear

explanation of the rules applicable in the city. Finally, the Response Generator API (generate_response(prompt, max_length=1024)) is the core function that processes a given prompt and generates an AI-powered response within a specified maximum length; for example, generate_response("Summarize the public transport system in Mumbai", max_length=500) will return a concise summary about Mumbai's buses, trains, and metro system.

8.Authentication

This application currently runs as an open Gradio interface without authentication. To secure access, a simple authentication mechanism can be added. Gradio provides built-in username and password login support by using the auth parameter when launching the app. For example:

app.launch(share=True, auth=("admin", "password123"))

With this setup, only users who enter the correct username and password can access the app. For more advanced use cases, authentication can be extended using:

Token-based Authentication: Generate API tokens for authorized users and validate requests before processing.

OAuth Integration: Connect with Google, GitHub, or enterprise logins for secure access.

Environment Variables: Store and manage credentials securely without hardcoding them into the code.

This ensures that only authorized users can access sensitive functionalities like City Analysis and Citizen Services.

9.User interface

The application uses a Gradio-based web interface that is simple and interactive. When launched, the UI displays a title header: “City Analysis & Citizen Services AI”. Below this, there are two main tabs:

1. City Analysis Tab – This tab allows the user to input a city name into a text box labeled “Enter City Name”. A button labeled “Analyze City” triggers the AI model to generate a detailed analysis of the selected city, including crime index, accident rates, traffic safety, and overall safety assessment. The results are shown in a large text area labeled “City Analysis (Crime Index & Accidents)”.

2. Citizen Services Tab – In this tab, users see a text box labeled “Your Query” where they can type questions about government policies, public services, or civic issues. A button labeled “Get Information” sends the query to the AI model, and the generated response is displayed in a large output box labeled “Government Response”.

Both tabs are designed with clean rows and columns, making the interface easy to navigate. The text boxes allow free-form input, while the output boxes provide AI-generated results in a readable format.

10. Testing

1. Environment and Setup Testing

Before running the project, ensure that all dependencies are correctly installed and the hardware configuration is appropriate:

Dependencies: Check that transformers, torch, and gradio are installed. Missing packages will prevent the model from loading or the interface from running.

GPU: Verify that the Colab runtime is set to T4 GPU. This ensures faster model inference and reduces latency.

PyTorch and CUDA compatibility: Confirm that PyTorch recognizes the GPU. If CUDA is not available, the model will run on CPU, which is slower.

Purpose: Ensures the system environment supports the project's requirements.

2. Model Loading Testing

Check that the model and tokenizer load without errors:

Tokenizer: Should initialize correctly, and if pad_token is missing, it should default to eos_token.

Model: Should load in the correct data type (float16 for GPU, float32 for CPU) and map to the appropriate device.

Purpose: Confirms that the AI model is ready to generate responses.

3. Core Function Testing (generate_response)

Test the function responsible for generating text responses:

Provide a sample prompt and verify that the output is coherent and relevant.

Ensure that the generated text does not include the original prompt.

Check that the function respects the maximum token length and temperature settings.

Purpose: Ensures the text generation logic works correctly before integrating with the interface.

4. City Analysis Function Testing (city_analysis)

Test with sample city names to verify output:

Check that the response includes key information:

1. Crime index and safety statistics

2. Accident rates and traffic information

3. Overall safety assessment

Confirm that the output is clear, formatted properly, and informative.

Purpose: Validates that the city analysis feature provides meaningful and accurate insights.

5. Citizen Interaction Function Testing (citizen_interaction)

Test with sample citizen queries:

Ensure responses are relevant to the question asked.

Responses should be accurate, helpful, and easy to understand.

Check for formatting issues or truncation.

Purpose: Confirms that the AI can assist users with civic, public service, or government-related queries effectively.

6. Gradio Interface Testing

Check the functionality of the user interface:

Input fields: Ensure textboxes accept inputs correctly.

Buttons: Clicking "Analyze City" or "Get Information" should trigger the correct function.

Output fields: Should display AI-generated responses clearly.

Tabs: Verify that switching between "City Analysis" and "Citizen Services" works as expected.

Sharing: Using share=True should generate a public link for testing externally.

Purpose: Confirms that users can interact with the AI system smoothly through the web interface.

7. Edge Case Testing

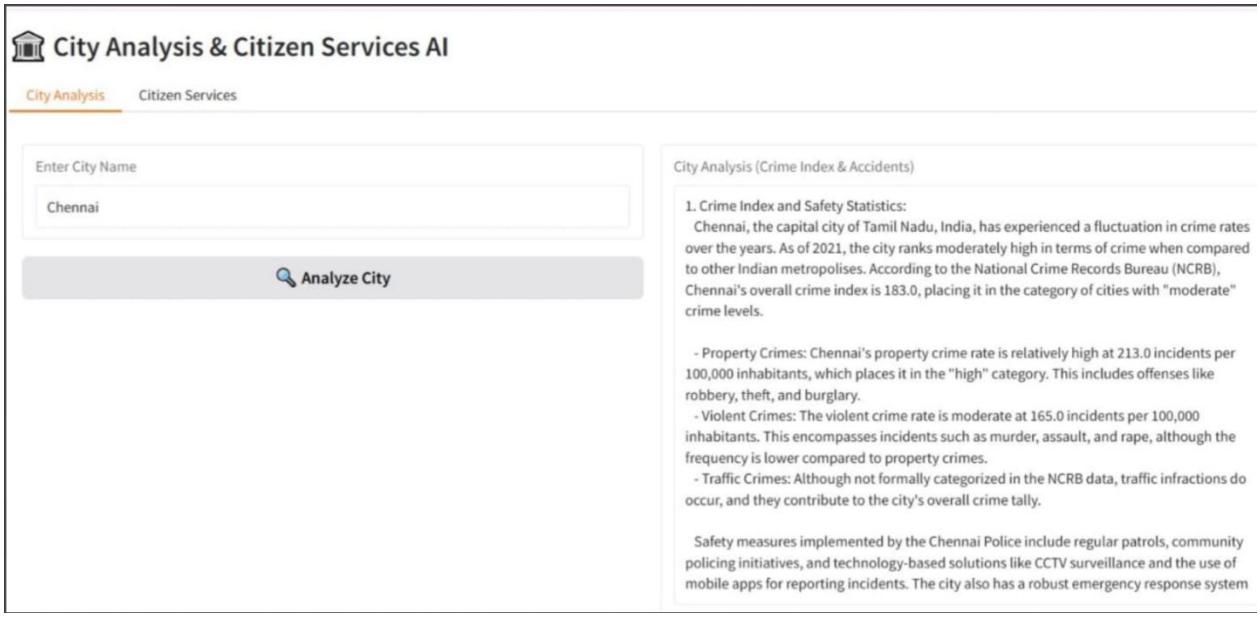
Invalid inputs: Empty strings, special characters, or unusual city names.

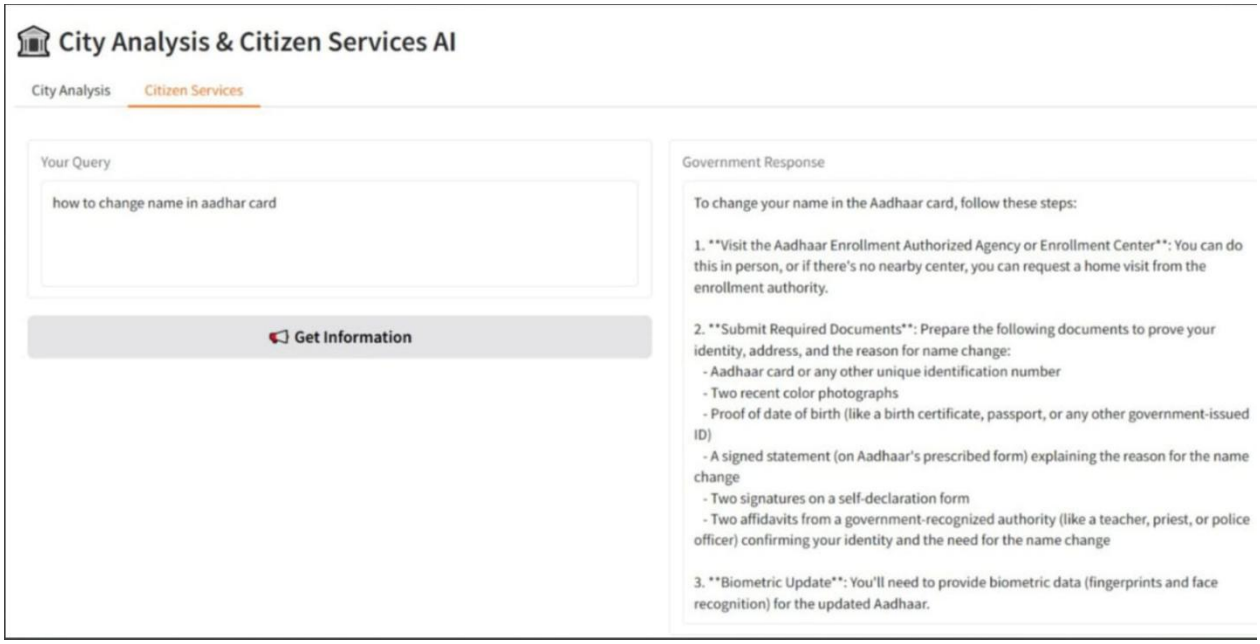
Long inputs: Very long prompts should not crash the system.

Concurrency: Multiple users or rapid queries in the interface should be handled without errors.

Purpose: Ensures the system is robust and handles unexpected situations gracefully.

11. Screen shots





12.Known issues

1. GPU and Memory Limitations

The model ibm-granite-3.2-2b-instruct is large and requires significant GPU memory.

On Google Colab, even with a T4 GPU, long prompts or multiple concurrent requests may cause out-of-memory errors.

Running on CPU will be very slow, and large prompts may fail entirely.

Impact: Users may experience delays or crashes with complex queries or long inputs.

2. Tokenizer and Padding Issues

If the tokenizer's pad_token is not set correctly, some text generation functions (like generate) may produce truncated or misaligned output.

The code currently sets pad_token to eos_token if missing, which is a workaround but may affect generation quality for certain prompts.

Impact: Outputs may occasionally be incomplete or have formatting issues.

3. Response Truncation

The max_length parameter in generate_response is fixed (default 1024), while input tokenization is limited to 512 tokens.

Long prompts or complex queries may cause truncated outputs, leading to incomplete city analyses or citizen service answers.

Impact: Users may not receive full responses for detailed queries.

4. No Error Handling

The current code does not handle exceptions for:

Model loading failures

Invalid or empty user inputs

CUDA or device-related errors

Any runtime error can crash the app without user-friendly feedback.

Impact: Users may see the interface freeze or stop responding without explanation.

5. AI Response Accuracy

The AI model may generate inaccurate or outdated information, especially for city statistics, crime indices, or government policies.

There is no verification against real-world data sources.

Impact: Users might receive misleading or incorrect advice if they rely on AI output alone.

6. Security and Privacy

The Gradio interface uses share=True, which exposes the app publicly.

User queries are processed by the model but are not encrypted or stored securely, which could be a privacy concern.

Impact: Sensitive or personal queries may be exposed over the internet.

7. User Interface Limitations

Textboxes and outputs are plain; long responses can overflow or become hard to read.

There is no input validation, spell correction, or guidance for users on formatting queries.

Impact: Poor user experience with very long or improperly formatted inputs.

13.Future enhancements

A potential future enhancement for this project would be to improve both the accuracy and usability of the AI system. The model could be integrated with real-time city data sources, such as government crime statistics, traffic reports, and civic service databases, to provide verified and up-to-date information rather than relying solely on the AI's knowledge. Additionally, implementing better error handling and input validation would make the system more robust, preventing crashes or incomplete responses. The user interface could also be enhanced with dynamic formatting, longer text handling, and interactive visualizations like charts or maps for city analysis. Incorporating multi-language support would expand accessibility, while logging and analytics could help track user queries to further improve response quality. Finally, adding security measures and privacy protections would ensure that citizen data and queries are handled safely, making the platform suitable for wider public deployment.