Project Title

Citizen AI: Intelligent Citizen Engagement Platform

Project Documentation

1. Introduction

- **Project title**: Citizen AI - Intelligent Citizen Engagement Platform
- **Team member**: Hashwathaman J NM ID: 9BCFED36A028F30B57F40EC4D5C51C68
- **Team member**: Paul Eugine NM ID: E9C1E579F1619A1AB5E4E3EBAD4DFB8F
- **Team member**: Kushvandhar R NM ID: 77E9C8CA92BA5179367434922E857CEC
- **Team member**: Sandhiya S NM ID: E85AED87E419527B226455BC6E3B0A1E

2. Project Overview

Purpose:

The purpose of Citizen AI is to revolutionize how governments interact with the public through an intelligent citizen engagement platform. By leveraging Flask, IBM Granite models, and IBM Watson, Citizen AI provides real-time, AI-driven responses to citizen inquiries regarding government services, policies, and civic issues. The platform integrates natural language processing (NLP) and sentiment analysis to assess public sentiment, track emerging issues, and generate actionable insights for government agencies. A dynamic analytics dashboard offers real-time visualizations of citizen feedback, helping policymakers enhance service delivery and transparency. By automating routine interactions and enabling data-driven governance, Citizen AI improves citizen satisfaction, government efficiency, and public trust in digital governance.

Features:

Real-Time Conversational AI Assistant

- Key Point: 24/7 citizen service interaction
- Functionality: Allows citizens to engage with public services naturally by typing questions or requests, with immediate AI-powered responses for government services, policies, and civic issues.

Citizen Sentiment Analysis

- • Key Point: Public opinion monitoring

  Functionality: Analyzes text input from citizen feedback to classify sentiment as Positive, Neutral, or Negative, helping identify areas of public satisfaction or concern.

Dynamic Analytics Dashboard

- Key Point: Real-time insights for officials
- Functionality: Visualizes citizen feedback, sentiment trends, interaction patterns, and service ratings through charts and metrics for data-driven decision making.

Natural Language Processing

- Key Point: Human-like communication
- Functionality: Processes citizen queries using IBM Watson NLP services to understand context and generate appropriate responses.

Issue Tracking and Reporting

- Key Point: Problem identification system
- Functionality: Automatically categorizes and tracks reported issues, enabling efficient resolution and follow-up.

Multi-Channel Integration

- Key Point: Unified communication platform
- Functionality: Supports various input methods including web interface, mobile compatibility, and potential integration with existing government systems.

Automated Response Generation

- Key Point: Instant information delivery
- Functionality: Uses IBM Granite models to generate contextually relevant responses to citizen inquiries in real-time.

Sentiment Trend Analysis

- Key Point: Long-term pattern recognition

- Functionality: Tracks sentiment changes over time to identify emerging issues and measure policy impact effectiveness.

Secure Data Handling

- Key Point: Privacy and security compliance
- Functionality: Implements encryption, authentication, and privacy law compliance for secure citizen data management.

3. Architecture

Frontend (Flask Templates): The frontend is built with Flask templating system, offering an interactive web interface with multiple pages including citizen chat interface, feedback forms, dashboard views, and administrative panels. Navigation is handled through responsive web design with Bootstrap integration. Each page is modularized for maintainability and scalability.

Backend (Flask Framework): Flask serves as the lightweight web framework that powers API endpoints for citizen interactions, sentiment analysis, dashboard data, and administrative functions. It is optimized for real-time processing and easy integration with IBM services.

LLM Integration (IBM Granite): IBM Granite LLM models are used for natural language understanding and response generation. Prompts are carefully designed to handle government service inquiries, policy explanations, and citizen support interactions.

NLP Services (IBM Watson): IBM Watson Natural Language Processing services handle sentiment analysis, text classification, and language understanding. Advanced NLP capabilities ensure accurate interpretation of citizen queries and feedback.

Database Layer (PostgreSQL/MongoDB): Citizen interactions, feedback data, and analytics are stored in a robust database system. Historical data is maintained for trend analysis and reporting purposes.

Analytics Engine: Real-time analytics processing using Python libraries like pandas and numpy for data aggregation, sentiment scoring, and trend calculation. Visualization components use Chart.js or similar libraries.

-

4. Setup Instructions

Prerequisites:

- Python 3.8 or later pip and
- virtual environment tools
- IBM Cloud account with Granite and Watson API access
- Database system (PostgreSQL recommended)
- Internet access for cloud services

Installation Process:

- Clone the repository
- Create virtual environment: `python -m venv venv`
- Activate virtual environment
- Install dependencies: `pip install -r requirements.txt`
- Create `.env` file and configure IBM API credentials
- Initialize database tables
- Run Flask development server: `python app.py`
- Access application at `http://localhost:5000`

5. Folder Structure

```
citizen-ai/
├── app.py – Main Flask application entry point
├── config.py – Configuration settings and environment variables
├── models/ – Database models and schemas
│   ├── citizen.py
│   ├── feedback.py
│   └── analytics.py
├── routes/ – Flask route handlers
│   ├── chat_routes.py
│   ├── dashboard_routes.py
│   ├── feedback_routes.py
│   └── admin_routes.py
├── services/ – Business logic and external integrations
│   ├── granite_service.py – IBM Granite model integration
│   ├── watson_service.py – IBM Watson NLP integration
│   ├── sentiment_analyzer.py – Sentiment analysis logic
│   └── analytics_service.py – Dashboard data processing
├── templates/ – HTML templates for frontend
│   ├── base.html
│   ├── chat.html
│   ├── dashboard.html
│   └── feedback.html
├── static/ – CSS, JavaScript, and static assets
│   ├── css/
│   ├── js/
│   └── images/
├── utils/ – Utility functions and helpers
└── requirements.txt – Python dependencies
```

6. Running the Application

To start the project: ➢ Set up environment variables for IBM API credentials

➢ Initialize the database with required tables

➢ Launch the Flask development server

➢ Navigate to the web interface in your browser

➢ Citizens can start chatting immediately for assistance

➢ Government officials can access the dashboard for analytics
➢ All interactions are processed in real-time using IBM AI services

- **Frontend (Flask Templates):** The frontend provides an intuitive web interface with citizen chat functionality, feedback submission forms, and administrative dashboards. The design is responsive and accessible across different devices.

Backend (Flask Framework): Flask handles HTTP requests, manages sessions, processes AI responses, and serves dynamic content. The modular structure allows for easy maintenance and feature additions.

7. API Documentation

Backend APIs available include:

- `POST /api/chat` – Accepts citizen queries and returns AI-generated responses from IBM Granite
- `POST /api/feedback` – Submits citizen feedback for sentiment analysis
- `GET /api/sentiment/{id}` – Retrieves sentiment analysis results for specific feedback
- `GET /api/dashboard/metrics` – Returns dashboard analytics data
- `GET /api/dashboard/trends` – Provides sentiment trend data over time
- `POST /api/admin/issues` – Administrative endpoint for issue management
- `GET /api/health` – Health check endpoint for system monitoring

Each endpoint includes proper error handling, input validation, and response formatting. API documentation is available through integrated Swagger UI for development and testing.

8. Authentication

This version includes basic session management for demonstration purposes. For production deployment, the following security measures are recommended:

- **JWT token-based authentication** for API access
- **OAuth2 integration** with IBM Cloud Identity services
- **Role-based access control** (citizen, government official, administrator)
- **Multi-factor authentication** for administrative dashboards
- **Session management** with secure cookie handling

- **API rate limiting** to prevent abuse

Planned enhancements include citizen account management, interaction history tracking, and personalized service recommendations.

9. User Interface

The interface is designed for accessibility and ease of use for both citizens and government officials. It includes:

- **Responsive chat interface** for natural citizen interactions
- **Real-time sentiment indicators** on dashboard
- **Interactive analytics charts** with filtering capabilities
- **Clean feedback forms** with validation
- **Administrative panels** for system management
- **Mobile-friendly design** for accessibility
- **Dark/light mode options** for user preference

The design prioritizes clarity, accessibility compliance (WCAG 2.1), and intuitive navigation with contextual help and guidance throughout the user journey.

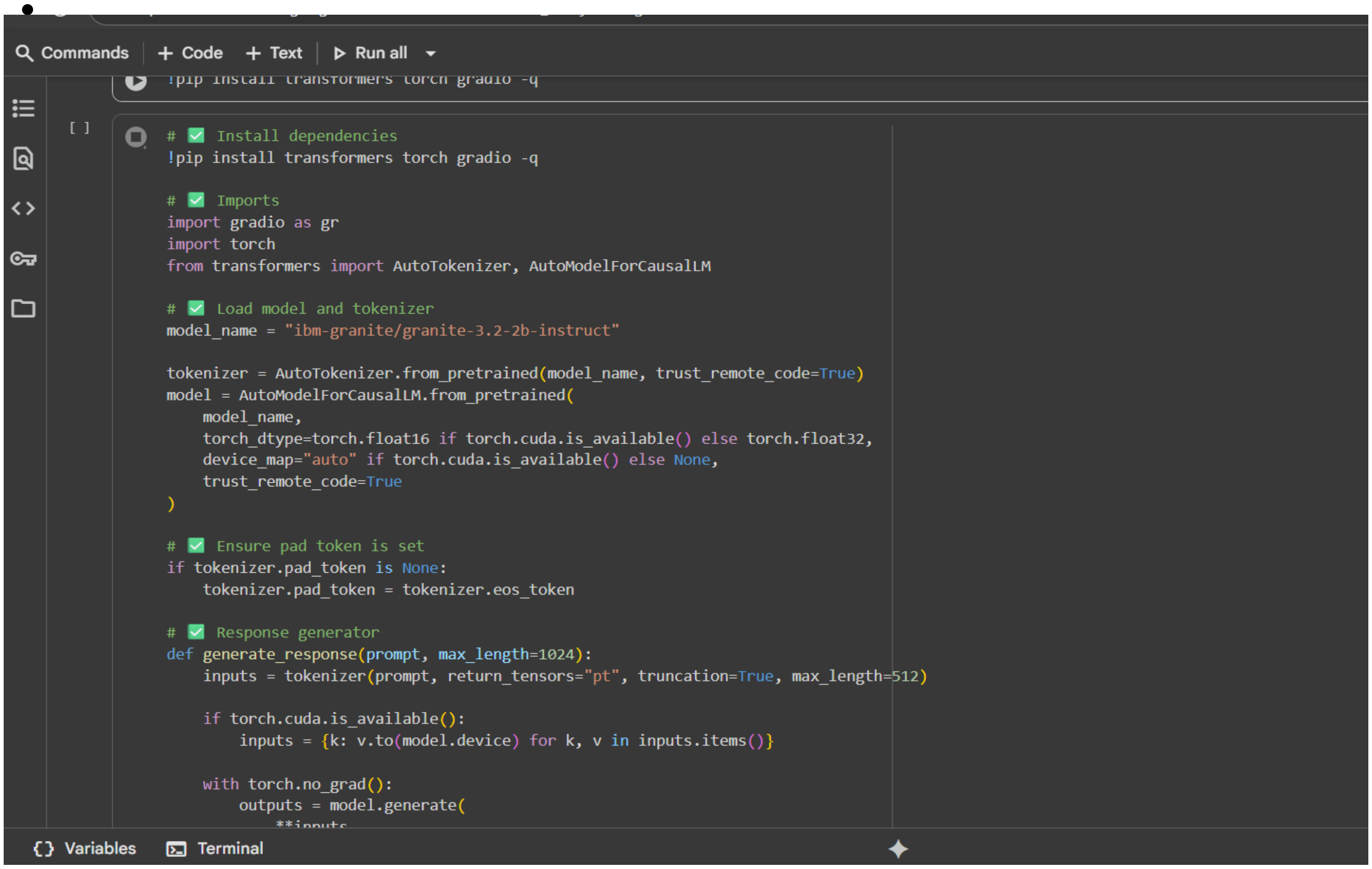
10. Testing

Testing was conducted in multiple phases:

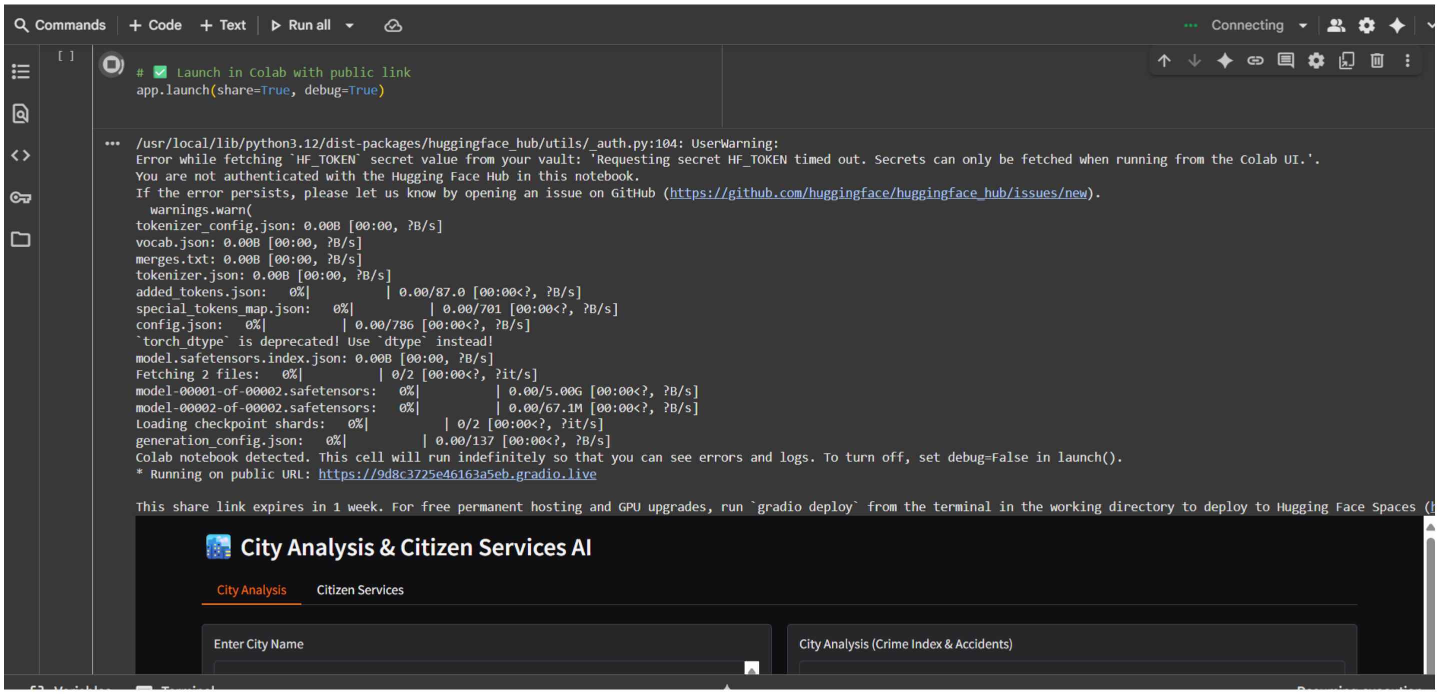
Unit Testing: Individual functions for sentiment analysis, AI response generation, and data processing
Integration Testing: IBM Granite and Watson API connections, database operations
API Testing: All endpoints tested using Postman and automated test scripts
User Acceptance Testing: Citizen interaction flows and government dashboard functionality
Security Testing: Input validation, authentication, and data protection measures
Performance Testing: Load testing for concurrent users and response times

Each component was validated to ensure reliability, accuracy, and performance under various usage scenarios.

11. Screenshots

- 

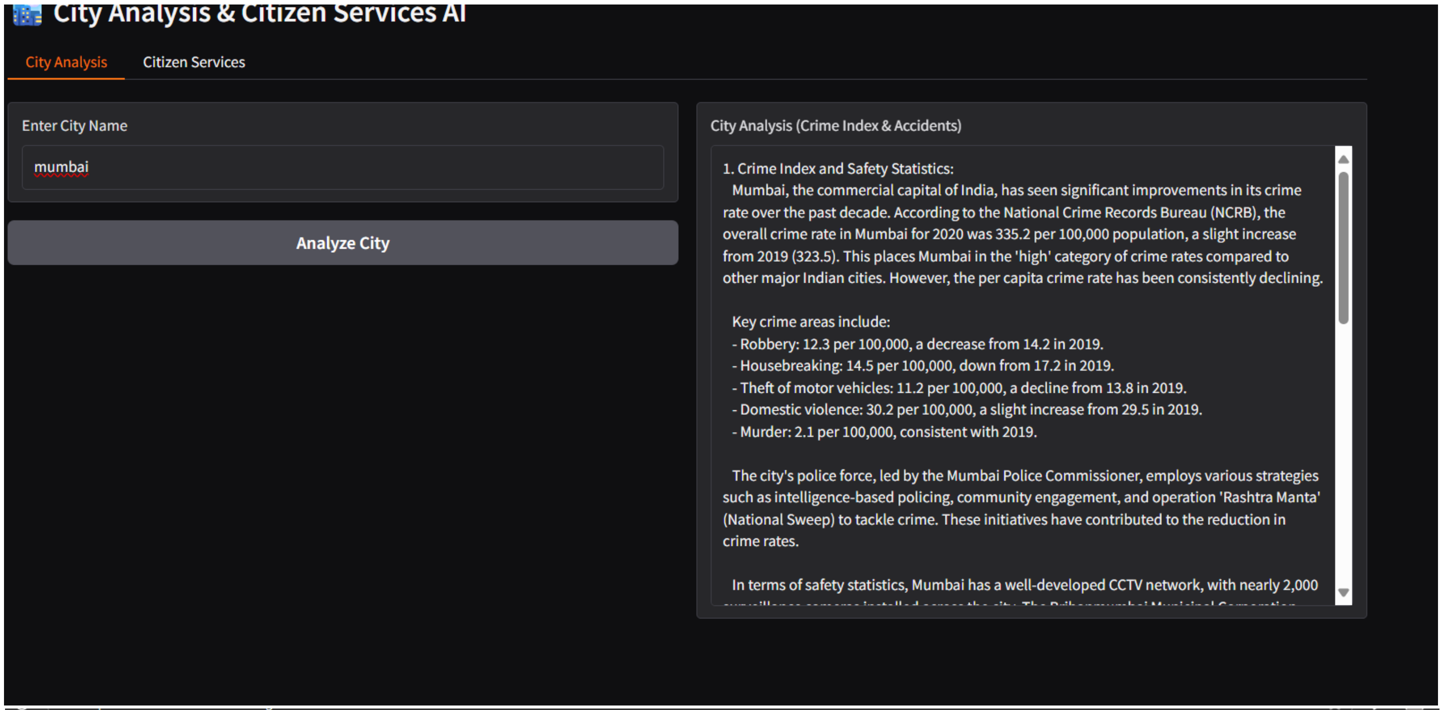
```
!pip install transformers torch gradio -q

# ✅ Install dependencies
!pip install transformers torch gradio -q

# ✅ Imports
import gradio as gr
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM

# ✅ Load model and tokenizer
model_name = "ibm-granite/granite-3.2-2b-instruct"

tokenizer = AutoTokenizer.from_pretrained(model_name, trust_remote_code=True)
model = AutoModelForCausalLM.from_pretrained(
    model_name,
    torch_dtype=torch.float16 if torch.cuda.is_available() else torch.float32,
    device_map="auto" if torch.cuda.is_available() else None,
    trust_remote_code=True
)

# ✅ Ensure pad token is set
if tokenizer.pad_token is None:
    tokenizer.pad_token = tokenizer.eos_token

# ✅ Response generator
def generate_response(prompt, max_length=1024):
    inputs = tokenizer(prompt, return_tensors="pt", truncation=True, max_length=512)

    if torch.cuda.is_available():
        inputs = {k: v.to(model.device) for k, v in inputs.items()}

    with torch.no_grad():
        outputs = model.generate(
            **inputs
```



```
# ✅ Launch in Colab with public link
app.launch(share=True, debug=True)
```

```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:104: UserWarning:
Error while fetching `HF_TOKEN` secret value from your vault: 'Requesting secret HF_TOKEN timed out. Secrets can only be fetched when running from the Colab UI.'.
You are not authenticated with the Hugging Face Hub in this notebook.
If the error persists, please let us know by opening an issue on GitHub (https://github.com/huggingface/huggingface_hub/issues/new).
  warnings.warn(
tokenizer_config.json: 0.00B [00:00, ?B/s]
vocab.json: 0.00B [00:00, ?B/s]
merges.txt: 0.00B [00:00, ?B/s]
tokenizer.json: 0.00B [00:00, ?B/s]
added_tokens.json:   0%|          | 0.00/87.0 [00:00<?, ?B/s]
special_tokens_map.json:   0%|          | 0.00/701 [00:00<?, ?B/s]
config.json:   0%|          | 0.00/786 [00:00<?, ?B/s]
`torch_dtype` is deprecated! Use `dtype` instead!
model.safetensors.index.json: 0.00B [00:00, ?B/s]
Fetching 2 files:   0%|          | 0/2 [00:00<?, ?it/s]
model-00001-of-00002.safetensors:   0%|          | 0.00/5.00G [00:00<?, ?B/s]
model-00002-of-00002.safetensors:   0%|          | 0.00/67.1M [00:00<?, ?B/s]
Loading checkpoint shards:   0%|          | 0/2 [00:00<?, ?it/s]
generation_config.json:   0%|          | 0.00/137 [00:00<?, ?B/s]
Colab notebook detected. This cell will run indefinitely so that you can see errors and logs. To turn off, set debug=False in launch().
* Running on public URL: https://9d8c3725e46163a5eb.gradio.live

This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the working directory to deploy to Hugging Face Spaces (
```

City Analysis & Citizen Services AI

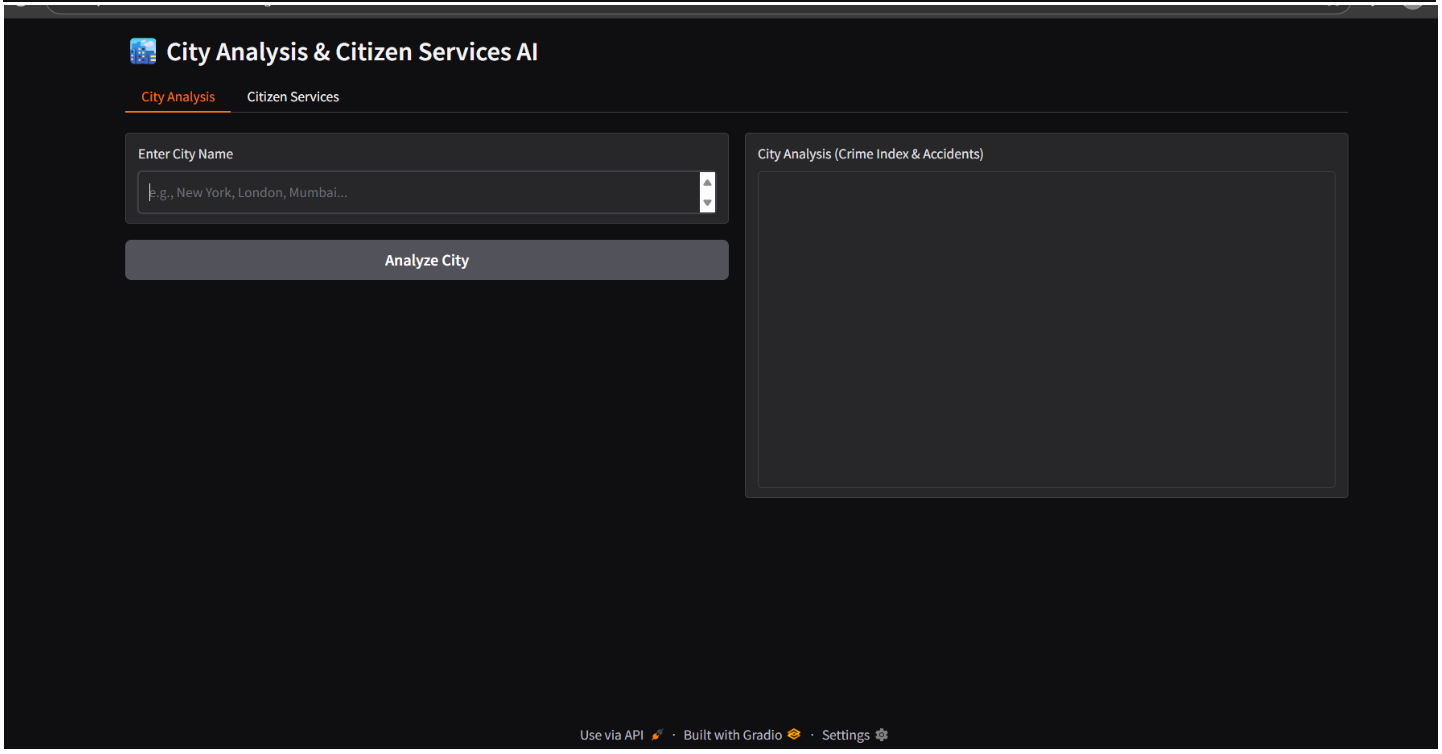

City Analysis & Citizen Services AI
City Analysis
Citizen Services
Enter City Name
mumbai
Analyze City
City Analysis (Crime Index & Accidents)

1. Crime Index and Safety Statistics:
Mumbai, the commercial capital of India, has seen significant improvements in its crime rate over the past decade. According to the National Crime Records Bureau (NCRB), the overall crime rate in Mumbai for 2020 was 335.2 per 100,000 population, a slight increase from 2019 (323.5). This places Mumbai in the 'high' category of crime rates compared to other major Indian cities. However, the per capita crime rate has been consistently declining.

Key crime areas include:
- Robbery: 12.3 per 100,000, a decrease from 14.2 in 2019.
- Housebreaking: 14.5 per 100,000, down from 17.2 in 2019.
- Theft of motor vehicles: 11.2 per 100,000, a decline from 13.8 in 2019.
- Domestic violence: 30.2 per 100,000, a slight increase from 29.5 in 2019.
- Murder: 2.1 per 100,000, consistent with 2019.

The city's police force, led by the Mumbai Police Commissioner, employs various strategies such as intelligence-based policing, community engagement, and operation 'Rashtra Manta' (National Sweep) to tackle crime. These initiatives have contributed to the reduction in crime rates.

In terms of safety statistics, Mumbai has a well-developed CCTV network, with nearly 2,000


City Analysis & Citizen Services AI
City Analysis
Citizen Services
Enter City Name
e.g., New York, London, Mumbai...
Analyze City
City Analysis (Crime Index & Accidents)
Use via API · Built with Gradio · Settings

12. Known Issues

- **Rate Limiting:** IBM API rate limits may affect response times during peak usage
- **Language Support:** Currently supports English only, multilingual support planned
- **Data Storage:** Large conversation histories may require database optimization
- **Offline Mode:** No offline functionality when IBM services are unavailable
- **Mobile Chat:** Some mobile browsers may have keyboard overlay issues

13. Future Enhancements

Short-term (3-6 months):

- Multi-language support for diverse populations
- Voice interface integration for accessibility
- Mobile application development
- Advanced analytics with predictive modeling
- Integration with existing government databases

Long-term (6-12 months):

- Machine learning for personalized responses
- Blockchain integration for secure document verification
- IoT integration for smart city data
- Advanced reporting and policy recommendation engine
- Cross-department data integration and analysis

Advanced Features:

- **Chatbot personality customization** based on department/service type

Predictive citizen service needs based on historical data

- **Automated policy impact assessment** through sentiment tracking
- **Integration with social media** for broader public sentiment analysis