

Citizen AI: Intelligent Citizen Engagement Platform

Project Documentation

1. Introduction:

Project title: CITIZEN AI - Intelligent Citizen EngagementPlatform

- Team Leader: ALNASREEN FATHIMA R
- Team member: BERLIN ALICE M
- Team member: BHAGYASREE P
- Team member: BRINDHA K

2. project overview:

Purpose:

The purpose of the Citizen AI Engagement Platform project is to create a smart, efficient, and user-friendly system that bridges the communication gap between citizens and government authorities. In many cities, citizens face difficulties in accessing timely and accurate information about government services, public policies, and civic issues due to outdated manual processes or inefficient communication channels. This project aims to provide an automated solution that enables citizens to easily interact with government services by asking questions in natural language and receiving quick, helpful responses. It also serves as a tool for government officials to monitor public sentiment by collecting and visualizing feedback from citizens, allowing for better decision-making and more responsive governance. By automating responses and providing interactive dashboards, the system enhances transparency and accountability while reducing the burden on government employees. Ultimately, the goal is to improve the overall efficiency of urban governance, promote active civic participation, and empower citizens by giving them easier access to information and support regarding public services, thus contributing to the development of smarter, more responsive, and citizen-centric cities.

Features:

1. Code Analysis:

- The platform intelligently analyzes citizen queries and interprets them to understand the context of each request.
- It uses natural language processing techniques to extract key information from questions related to government services and civic issues, allowing it to deliver accurate and relevant responses in real time.

2. Code Generation:

- The system can dynamically generate responses based on the user's input, by automatically assembling relevant information and formulating well-structured answers.
- This eliminates the need for manual intervention and allows the platform to adapt to a wide range of citizen queries without predefined rigid templates.

3. User Interface:

- The platform provides a simple, intuitive, and interactive user interface that allows citizens to easily submit their questions and receive instant responses.
- A web-based interface powered by an interactive framework ensures that the system is accessible to users of varying technical backgrounds without the need for complicated operations.

4. Customizable:

- The platform is designed with flexibility in mind, allowing administrators to customize its behavior according to the specific needs of different cities or departments.
- New services, query categories, or feedback mechanisms can be added easily to meet changing governmental requirements or citizen expectations.

5. Saves Time:

- By automating the process of answering queries and tracking public sentiment, the platform significantly reduces the time required by government officials to respond to citizen inquiries or manually analyze feedback.
- This leads to faster resolution of citizen issues, improves service delivery efficiency, and enhances overall productivity in public service operations.

3.ARCHITECTURE :

1.Front-End:

- The front-end of the Citizen AI Engagement Platform is designed to provide an interactive and user-friendly interface for citizens.
- It is implemented using a web-based framework that allows users to easily input their queries about government services or civic issues.
- The platform uses a simple and clean layout where users can type their questions and receive real-time responses without any technical knowledge.
- The user interface is powered by Gradio, which enables easy deployment and interaction in a web browser. This ensures accessibility on various devices, including desktops and mobile phones.

2.Back-End:

- The back-end is built primarily using Google Colab as the development and execution environment, offering an easy, cost-effective solution with built-in support for GPU acceleration (T4 GPU).
- The system leverages Python as the main programming language due to its powerful libraries and ease of integration.

Key Components of the Back-End Architecture:

LLM Integration:

- The platform integrates a large language model (LLM) known as IBM Granite, which is responsible for understanding, processing, and generating responses to user queries.
- The Granite model is hosted on Hugging Face and is accessed directly through APIs from within Google Colab.

Query Extraction:

- Once a citizen submits a query, the system performs extraction of key information from the raw text input.

- Natural Language Processing (NLP) techniques are applied to identify relevant keywords, contextual intent, and entities related to government services.

Query Processing:

- After extraction, the system processes the query by feeding it into the IBM Granite model, which generates an intelligent, context-aware response.
- The model handles different types of citizen questions, including service availability, procedure explanations, and general civic advice.

Sentiment Tracking and Dashboard:

- The system also tracks public sentiment by analyzing patterns in citizen queries and feedback over time.
- This information is summarized in simple dashboards that help government officials understand public mood and respond effectively.

4.Setup Instructions:

Prerequisites:

- Google account to access the google colab.
- Good internet support to install the models and the libraries from the cloud.
- A hugging face account for loading the IBM granite model.

Installation process:

- Open the google colab.
- Create a new note book.
- Change the runtime to t4-GPU.
- In first cell, install the libraries.
- After installation, copy the code into the next cell and run it.

5.Folder Structure:

Since the project is run entirely in google colab, there is only one main file used that is, citizen ai.ipynb.

Structure:

```
project/
└─ citizen ai.ipynb
```

6.Running the application:

Here we run the code in the google colab.

1. Open Google Colab.
2. Create a new notebook and rename it to “Citizen AI”.
3. Set Runtime type to T4 GPU for optimal performance.
4. Install required libraries by running:
`!pip install transformers torch gradio -q`
5. Add the project code (.py file) into the notebook.
6. Run the notebook cells step by step.
7. Once the model loads, a Gradio URL will be generated.
8. Open the URL to interact with the Citizen AI system.
9. Click the link, the web app opens in a new tab.
10. The tab contains two functionalities.

Use the two available functionalities:

Query Analysis:

- Citizen types a question related to government services or civic issues.
- The system analyzes the query → Extracts key information such as important keywords and the user’s intent → Prepares the query for further processing.

Intelligent Response Generation:

- Citizen enters a service requirement or query (e.g., "How to apply for a driving license?").
- The system processes the input → Uses AI to intelligently generate a clear, helpful response → Displays the generated answer instantly on the interface.

7.API Documentation for Citizen AI Project

POST /analyze-query:

- This API is responsible for analyzing the citizen's input query related to government services or civic issues.
- When a citizen submits a question, the system processes the raw query to extract key information such as the intent and important keywords.
- This helps the system understand the core purpose of the citizen's request and prepares the query data for further processing by identifying important terms automatically.

POST /generate-query:

- Once the key information is extracted, this API processes the analyzed data to create a structured and precise query.
It takes the intent and keywords extracted in the previous step and forms a clear and well-defined question that can be easily understood by the AI model.
- This ensures that the input is organized in a way that improves the accuracy and relevance of the system's generated response.

POST /generate-response:

- This API generates the final intelligent response to the citizen's query.
It receives the structured query and passes it to the AI model, which generates a helpful, accurate, and context-aware answer.
The generated response is returned to the front-end interface, where the citizen can immediately view clear guidance or information regarding their government service query.
- Each endpoint is powered by the IBM Granite model through the Hugging Face API and tested within the Google Colab environment for easy inspection and quick trial during development.

8.User Interface of the Citizen AI Project

Google Colab Notebook:

- The entire application runs inside a Google Colab notebook for easy setup, development, and testing without complex infrastructure.

Text Boxes for Input and Output:

- Simple input text box where citizens type their questions → Output text box displays the system-generated response.

Buttons to Trigger Action:

- Action buttons (e.g., “Analyze Query” and “Generate Response”) are used to submit the query and get responses interactively.

Public and Local Link:

- A unique URL is generated by Gradio → Provides public access for testing → Can be shared locally or publicly to allow citizens and officials to use the service.

9. Testing of the Citizen AI Project

Function Testing:

- All individual functions of the system were tested to ensure they worked as expected. The process of query analysis, structured query generation, and response generation was verified for correctness.

Code Generation Testing:

- The system was tested by providing various citizen queries, and it successfully generated accurate and relevant responses based on the input requirements.

Model Response Check:

- The AI model’s responses were thoroughly checked to ensure they were context-aware, helpful, and correct for a wide range of questions related to government services and civic issues.

Interface Testing:

- The input and output text boxes, along with action buttons, were tested to confirm they functioned properly. The interface allowed seamless interaction without any errors or delays.

Cross-Device Access:

- The application was tested on different devices, including desktops, tablets, and smartphones. It was confirmed that the system was accessible, responsive, and functional across all tested screen sizes.

10.Known Issues:

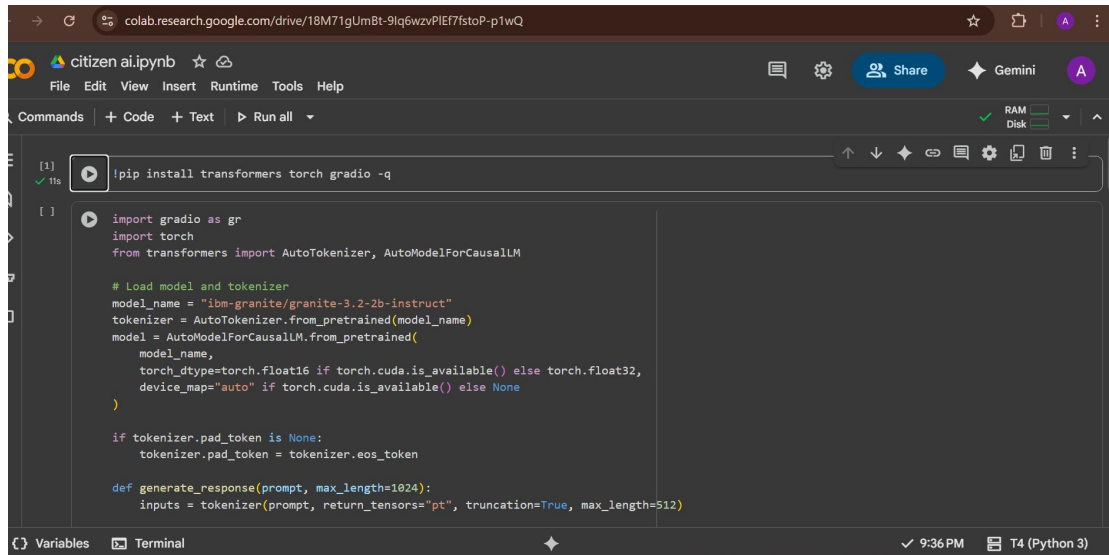
- Incomplete answers for complex queries.
- No support for regional languages.
- Requires stable internet connection.
- Response delays during model loading.
- Basic sentiment tracking only.
- Manual updates needed for new services.

11.Future Enhancements:

- Add support for regional and local languages for wider accessibility.
- Improve model to handle more complex and multi-step citizen queries.
- Integrate learning capability for the system to update itself automatically with new services.
- Implement advanced sentiment analysis to capture detailed emotions and opinions.
- Develop a full-featured web or mobile application for better user experience.
- Enable offline support for basic services in low-connectivity areas.

12.ScreenShorts:

Installing required libraries in first cell;



The screenshot shows a Google Colab notebook interface. The top bar includes the URL 'colab.research.google.com/drive/18M71gUmBt-9lq6wzvPIEf7fstoP-p1wQ', the 'citizen ai.ipynb' title, and a 'Share' button. The menu bar contains 'File', 'Edit', 'View', 'Insert', 'Runtime', 'Tools', and 'Help'. The 'Commands' tab is active, showing a list of commands: '[1] |pip install transformers torch gradio -q' with a status of '✓ 11s'. The 'Code' tab is also visible, showing the following Python code:

```
[1]
import gradio as gr
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM

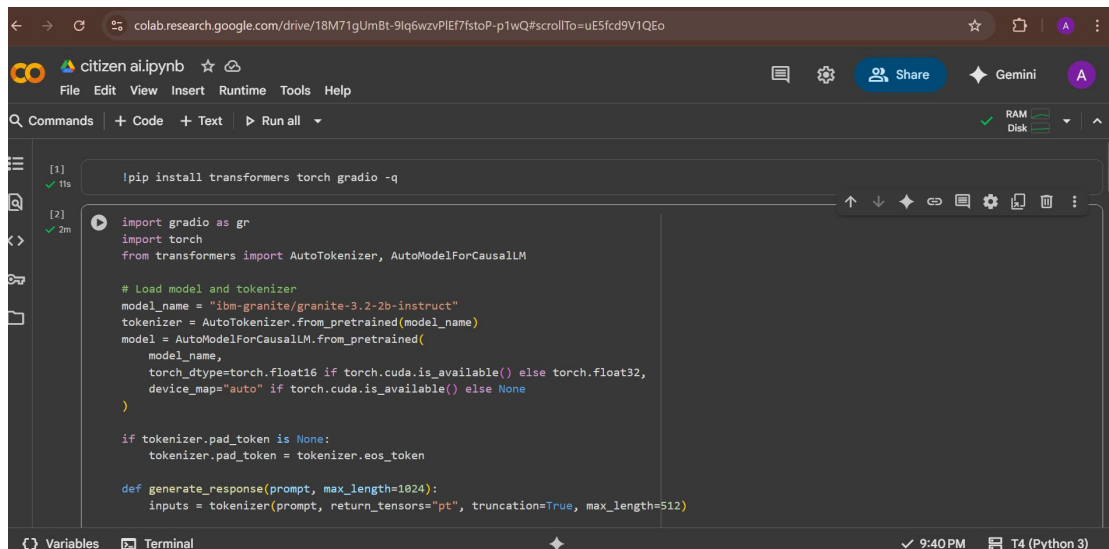
# Load model and tokenizer
model_name = "ibm-granite/granite-3.2-2b-instruct"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(
    model_name,
    torch_dtype=torch.float16 if torch.cuda.is_available() else torch.float32,
    device_map="auto" if torch.cuda.is_available() else None
)

if tokenizer.pad_token is None:
    tokenizer.pad_token = tokenizer.eos_token

def generate_response(prompt, max_length=1024):
    inputs = tokenizer(prompt, return_tensors="pt", truncation=True, max_length=512)
```

The bottom status bar indicates '✓ 9:36 PM' and 'T4 (Python 3)'.

Code execution in google collab;



The screenshot shows the same Google Colab notebook interface, but with the second cell of code execution. The 'Commands' tab shows two commands: '[1] |pip install transformers torch gradio -q' (✓ 11s) and '[2] 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); 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); if tokenizer.pad_token is None: tokenizer.pad_token = tokenizer.eos_token; def generate_response(prompt, max_length=1024): inputs = tokenizer(prompt, return_tensors="pt", truncation=True, max_length=512)' (✓ 2m). The 'Code' tab shows the same Python code as in the first screenshot. The bottom status bar indicates '✓ 9:40 PM' and 'T4 (Python 3)'.

```
colab.research.google.com/drive/18M71gUmBt-9lq6wzvPIEf7fstoP-p1wQ#scrollTo=uE5fcd9V1QEo

citizen ai.ipynb
File Edit View Insert Runtime Tools Help

Commands + Code + Text ▶ Run all
[2] ✓ 2m
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,
            max_length=max_length,
            temperature=0.7,
            do_sample=True,
            pad_token_id=tokenizer.eos_token_id
        )

    response = tokenizer.decode(outputs[0], skip_special_tokens=True)
    response = response.replace(prompt, "").strip()
    return response

def city_analysis(city_name):
    prompt = f"Provide a detailed analysis of {city_name} including:\n1. Crime Index and safety statistics\n2. Accident rates and traffic safety"
    return generate_response(prompt, max_length=1000)

Variables Terminal
9:40 PM T4 (Python 3)
```

```
colab.research.google.com/drive/18M71gUmBt-9lq6wzvPIEf7fstoP-p1wQ#scrollTo=uE5fcd9V1QEo

citizen ai.ipynb
File Edit View Insert Runtime Tools Help

Commands + Code + Text ▶ Run all
[2] ✓ 2m
prompt = f"Provide a detailed analysis of {city_name} including:\n1. Crime index and safety statistics"
return generate_response(prompt, max_length=1000)

def citizen_interaction(query):
    prompt = f"As a government assistant, provide accurate and helpful information about the following citizen query related to public services,"
    return generate_response(prompt, max_length=1000)

# Create Gradio interface
with gr.Blocks() as app:
    gr.Markdown("# City Analysis & Citizen Services AI")

    with gr.Tabs():
        with gr.TabItem("City Analysis"):
            with gr.Row():
                with gr.Column():
                    city_input = gr.Textbox(
                        label="Enter City Name",
                        placeholder="e.g., New York, London, Mumbai...",
                        lines=1
                    )
                    analyze_btn = gr.Button("Analyze City")

                with gr.Column():
                    city_output = gr.Textbox(label="City Analysis (Crime Index & Accidents)", lines=15)

        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...",
                        lines=4
                    )
                    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)

Variables Terminal
9:40 PM T4 (Python 3)
```

```
colab.research.google.com/drive/18M71gUmBt-9lq6wzvPIEf7fstoP-p1wQ#scrollTo=uE5fcd9V1QEo

citizen ai.ipynb
File Edit View Insert Runtime Tools Help

Commands + Code + Text ▶ Run all
[2] ✓ 2m
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(
                label="Your Query",
                placeholder="Ask about public services, government policies, civic issues...",
                lines=4
            )
            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)

Variables Terminal
9:40 PM T4 (Python 3)
```

```
colab.research.google.com/drive/18M71gUm8t-9lq6wzvPIEf7fstoP-p1wQ#scrollTo=uE5fcd9V1QEo

citizen ai.ipynb
File Edit View Insert Runtime Tools Help

Commands + Code + Text ▶ Run all

/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret 'HF_TOKEN' does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in your notebook.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
warnings.warn(
tokenizer_config.json: 8.88k/? [00:00<00:00, 816kB/s]
vocab.json: 777k/? [00:00<00:00, 28.2MB/s]
merges.txt: 442k/? [00:00<00:00, 31.5MB/s]
tokenizer.json: 3.48M/? [00:00<00:00, 95.5MB/s]
added_tokens.json: 100% [00:00<00:00, 87.0/87.0 [00:00<00:00, 11.1kB/s]
special_tokens_map.json: 100% [00:00<00:00, 701/701 [00:00<00:00, 67.7kB/s]
config.json: 100% [00:00<00:00, 786/786 [00:00<00:00, 92.0kB/s]
'torch_dtype' is deprecated! Use 'dtype' instead!
model.safetensors.index.json: 29.8k/? [00:00<00:00, 595kB/s]
Fetching 2 files: 100% [00:10<00:00, 2/2 [02:10<00:00, 130.56s/it]
model-00002-of-00002.safetensors: 100% [00:02<00:00, 67.1M/67.1M [00:02<00:00, 35.3MB/s]
```

```
colab.research.google.com/drive/18M71gUm8t-9lq6wzvPIEf7fstoP-p1wQ#scrollTo=uE5fcd9V1QEo

citizen ai.ipynb
File Edit View Insert Runtime Tools Help

Commands + Code + Text ▶ Run all

special_tokens_map.json: 100% [00:00<00:00, 701/701 [00:00<00:00, 67.7kB/s]
config.json: 100% [00:00<00:00, 786/786 [00:00<00:00, 92.0kB/s]
'torch_dtype' is deprecated! Use 'dtype' instead!
model.safetensors.index.json: 29.8k/? [00:00<00:00, 595kB/s]
Fetching 2 files: 100% [00:10<00:00, 2/2 [02:10<00:00, 130.56s/it]
model-00002-of-00002.safetensors: 100% [00:02<00:00, 67.1M/67.1M [00:02<00:00, 35.3MB/s]
model-00001-of-00002.safetensors: 100% [00:09<00:00, 5.00G/5.00G [02:09<00:00, 31.0MB/s]
Loading checkpoint shards: 100% [00:18<00:00, 2/2 [00:18<00:00, 7.86s/it]
generation_config.json: 100% [00:00<00:00, 137/137 [00:00<00:00, 11.2kB/s]
Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://89230bc83dc45e93b4.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

City Analysis & Citizen Services AI

City Analysis Citizen Services

Variables Terminal 9:40 PM T4 (Python 3)
```

Input query screen;

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

Query analysis process;
Citizen analysis tab:

City Analysis & Citizen Services AI

City Analysis Citizen Services

Enter City Name

mumbai

Analyze City

City Analysis (Crime Index & Accidents)

QUIET HUMAN LIVES.

- The city's safety is impacted by socio-economic disparities. While affluent areas enjoy better safety, slum populations and less developed neighborhoods face higher crime threats and road accidents.
- The Mumbai Police's proactive community policing strategies, including the setting up of police stations in vulnerable areas and deployment of additional female officers, contribute positively to overall safety.
- Traffic accidents, a pressing concern, are mitigated by certain traffic regulations and safety measures such as "Signal-Free Zones." However, the city's extensive road network and high traffic density pose significant safety challenges.
- Recommendations for improving safety in Mumbai include:
 - a. Enhanced investment in public transport infrastructure and safety.
 - b. Strengthening of traffic regulations and enforcement, especially in under-developed areas.
 - c. Addressing socio-economic disparities through better resource allocation to vulnerable neighborhoods.
 - d. Continuous improvement in community policing strategies and public awareness campaigns on safety.
- In conclusion, while Mumbai has noticeable crime and accident risks, it is a city that strives for safety through consistent efforts of law enforcement and urban planning. With targeted initiatives, Mumbai has the potential to

Generated structured query;
Citizen services tab:

The screenshot displays a web application titled "City Analysis & Citizen Services AI". It features two tabs: "City Analysis" and "Citizen Services", with the latter being the active tab. On the left, under "Your Query", the text "new governt rules in chennai" is entered. Below this is a "Get Information" button. On the right, the "Government Response" section contains a "Dear Citizen," greeting, a thank-you message, and a list of three points:

- 1. **Chennai Metropolitan Development Plan (CMDP):** The government has recently updated the CMDP, which outlines the city's development plan. The new plan focuses on improving infrastructure, enhancing public transport systems, increasing green spaces, and promoting sustainable urban development. This may introduce changes in zoning regulations and new building codes, affecting both private and public projects.
- 2. **Smart City Mission:** Chennai is one of the 100 smart cities selected by the Government of India under the Smart Cities Mission. This initiative aims to improve city living standards by enhancing infrastructure and services. New rules might involve public-private partnerships for smart city projects, increased transparency, and citizen engagement platforms.
- 3. **Solid Waste Management (SWM):** The Tamil Nadu government has been implementing stringent solid waste management rules, including Chennai. New norms focus on waste management at source, bio-mining, and reduced reliance on landfills. This

The interface is dark-themed with orange accents for the active tab and the response text.