

# **CitizenAI: City Analysis & Citizen Services**

## **Project Documentation**

### **1.Introduction:**

- **CitizenAI: City Analysis & Citizen Services**
- Team member: Alnasreen Fathima R
- Team member: Berlin Alice M
- Team member: Bhagyashree P
- Team member: Brindha K

### **2. Project Overview:**

#### **Purpose:**

The purpose of this project is to build an AI-powered application named CitizenAI that assists in analyzing cities based on crime statistics, accident rates, and overall safety. It also provides helpful information to citizens regarding public services, government policies, and civic issues through an AI-driven interface.

#### **Features:**

- **City Analysis:**

Users can input any city name, and the system will generate a detailed analysis covering crime index, accident rates, and overall safety.

**Citizen Services:**

Users can input queries related to government policies, civic issues, and services. The system will respond with accurate, AI-generated information.

Users can input any concept, and the system will generate a clear and detailed explanation, making it easier to understand.

- **Quiz Generation:**

The system can automatically generate quiz questions based on the given topic, which helps learners in self-assessment and practice.

- **User Interface:**

The interface is simple and user-friendly, with two tabs that is one for concept explanation and the other for quiz generation.

- **Customizable:**

Users can provide any concept or topic of their choice, and the system generates explanations or quizzes accordingly.

- **Saves Time:**

It quickly provides study material and quizzes, reducing the manual effort for both students and teachers.

### **3. Architecture:**

- Frontend (Gradio):

The frontend is developed using Gradio, which provides a simple and interactive web-based interface. Users can choose between two main tabs:

City Analysis Tab – For entering a city name and generating a detailed report on crime index, accidents, and safety.

Citizen Services Tab – For entering a query related to public services or government policies, and receiving an AI-generated response.

The frontend is developed using Gradio, which provides a simple and interactive web-based interface. Users can enter concepts or topics, and the system will display either explanations or quiz questions. The interface is organized into two main tabs:

1. Concept Explanation Tab – For entering a concept and generating a detailed explanation.
2. Quiz Generator Tab – For entering a topic and generating quiz questions automatically.

- Backend (Google Colab + Python):

The backend runs on Google Colab, where the Python environment handles model loading, request processing, and interaction with the IBM Granite model. It processes city names or queries, communicates with the AI model, and returns structured results back to the Gradio interface.

The backend runs on Google Colab, where the Python environment handles model loading, request processing, and interaction with the AI model. It processes inputs, communicates with the IBM Granite model, and returns structured results back to the Gradio interface.

- LLM Integration (IBM Granite):

The project integrates the IBM Granite LLM for natural language understanding and generation. The model is responsible for providing detailed city analysis reports and responding to citizen queries accurately.

The project integrates the IBM Granite LLM for natural language understanding and generation. The model is responsible for providing detailed explanations of concepts and generating quiz questions based on topics entered by the user.

## **4. Setup Instructions:**

### **Prerequisites:**

- A Google account to access Google Colab.
- A stable internet connection to install models and libraries from the cloud.
- A Hugging Face account for loading the IBM Granite model.

### **Installation Process:**

- Open Google Colab.
- Create a new notebook.
- Change the runtime to T4-GPU.

- In the first cell, install the required libraries.
- After installation, copy the project code into the next cell and run it.

## **5. Folder Structure:**

Structure:

project/

Since the project runs entirely in Google Colab, there is only one main file used: citizenAI.ipynb

Structure:

project/  
└─ citizenAI.ipynb

## **6. Running the Application:**

- Execute the code cell with `app.launch(share=True)`.
- Colab will display a public Gradio link like:

(Running on public URL: <https://xxxxx.gradio.live>)

- Click the link to open the web app in a new tab.
- The interface contains two main functionalities:

Concept Explanation → Enter a concept → Click Explain → Get a detailed explanation.

- Quiz Generation → Enter a topic → Click Generate Quiz → AI generates quiz question.

## **7. API Documentation:**

Backend APIs available include:

POST /city-analysis – Accepts a city name as input and responds with an AI-generated safety and crime analysis.

POST /citizen-query – Accepts a public services or civic query and generates

relevant AI responses.

Each endpoint is tested and documented within the Colab environment for quick inspection and trial during development.

**POST /explain-concept** – Accepts a concept as input and responds with an AI-generated explanation.

**POST /generate-quiz** – Accepts a topic and generates quiz questions related to the input.

**POST /upload-text** – Allows users to upload raw text for processing and explanation.

Each endpoint is tested and documented within the Colab environment for quick inspection and trial during development.

## **8. User Interface:**

- **Google Colab Notebook** – The entire project runs inside a single Colab notebook with a Gradio interface.
- Two Tabs in Gradio are:

City Analysis – In this tab, users can enter a city name, and the AI generates a clear and detailed safety analysis.

Citizen Services – In this tab, users can enter a civic or government query, and the AI generates a relevant response.

Textboxes for Input/Output – Simple boxes are provided to type the city/query and to display the AI-generated responses.

Buttons to Trigger Actions – 'Analyze City' and 'Get Information' buttons execute the AI model and show the results.

Public/Local Link – Gradio provides a shareable link so that the interface can be accessed from any browser or device.

- **Concept Explanation** – In this tab, users can enter a concept, and the AI generates a clear and detailed explanation.
- **Quiz Generator** – In this tab, users enter a topic, and the AI generates relevant quiz questions for practice.

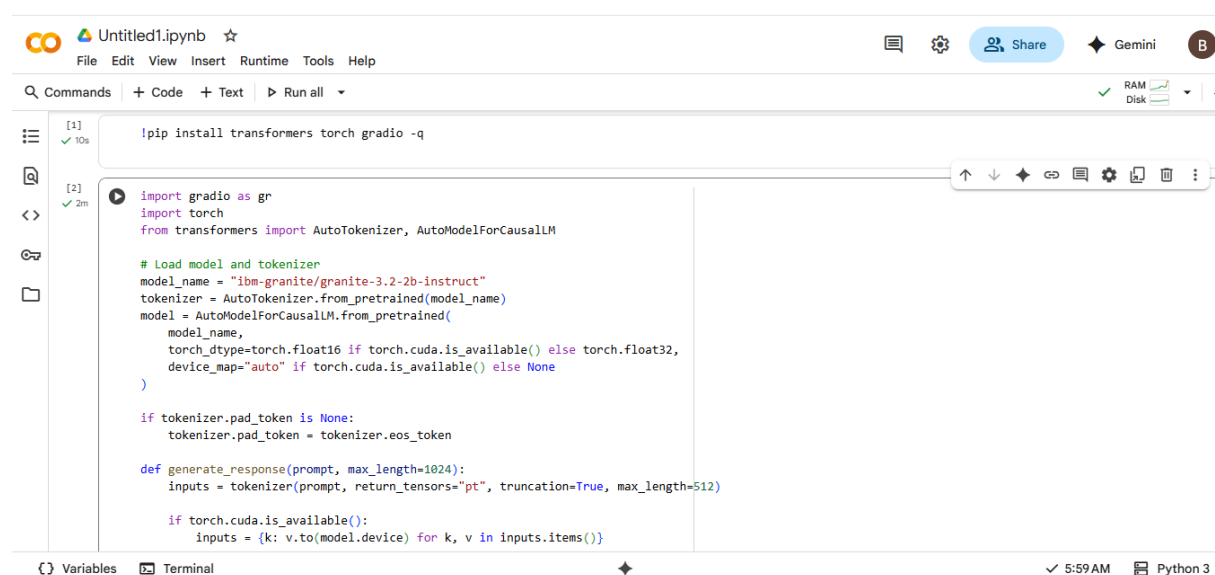
- **Textboxes for Input/Output** – Simple boxes are provided to type the concept/topic and to display the AI-generated explanations or quiz questions.
- **Buttons to Trigger Actions** – “Explain” and “Generate Quiz” buttons execute the AI model and show the results.
- **Public/Local Link** – Gradio provides a shareable link so that the interface can be accessed from any browser or device.

## 9. Testing:

- **Unit Testing:** For individual functions like `generate_response()` to ensure correct AI outputs.
- **API Testing:** By sending test prompts through Gradio’s backend and validating responses.
- **Manual Testing:** For verifying input/output quality, checking analysis accuracy, and validating responses to citizen queries.
- **Edge Case Handling:** Tested with empty inputs, long prompts, irrelevant queries, and unusual topics to ensure robust handling.

## 10.Screenshots:

### Program:



The screenshot shows a Jupyter Notebook titled 'Untitled1.ipynb' with a menu bar (File, Edit, View, Insert, Runtime, Tools, Help) and a toolbar (Commands, + Code, + Text, Run all). The code is as follows:

```
[1] ✓ 10s
!pip install transformers torch gradio -q

[2] ✓ 2m
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)

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

The bottom of the interface shows 'Variables' and 'Terminal' tabs, a status bar with '5:59 AM' and 'Python 3', and a RAM/Disk usage indicator.

```
Untitled1.ipynb
File Edit View Insert Runtime Tools Help

[2] 2m
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)

        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")
```

```
Untitled1.ipynb
File Edit View Insert Runtime Tools Help

[2] 2m
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)
```

```
Untitled1.ipynb
File Edit View Insert Runtime Tools Help

[2] 2m
vocab.json: 777k? [00:00<00:00, 9.25MB/s]
merges.txt: 442k? [00:00<00:00, 14.0MB/s]
tokenizer.json: 3.48M? [00:00<00:00, 53.3MB/s]
added_tokens.json: 100% [87.0/87.0 [00:00<00:00, 7.63kB/s]
special_tokens_map.json: 100% [701/701 [00:00<00:00, 53.2kB/s]
config.json: 100% [786/786 [00:00<00:00, 40.2kB/s]
`torch_dtype` is deprecated! Use `dtype` instead!
model.safetensors.index.json: 29.8k? [00:00<00:00, 1.11MB/s]
Fetching 2 files: 100% [2/2 [01:30<00:00, 90.51s/t]
model-00001-of-00002.safetensors: 100% [5.00G/5.00G [01:30<00:00, 106MB/s]
model-00002-of-00002.safetensors: 100% [67.1M/67.1M [00:08<00:00, 6.50MB/s]
Loading checkpoint shards: 100% [2/2 [00:27<00:00, 11.36s/t]
generation_config.json: 100% [137/137 [00:00<00:00, 8.66kB/s]
Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://400224da24657758d7.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
```

## OUTPUT:

The top screenshot displays the 'City Analysis & Citizen Services AI' application in a web browser. The 'City Analysis' tab is active, showing a form with 'california' entered and an 'Analyze City' button. The output on the right, titled 'City Analysis (Crime Index & Accidents)', includes a section on '1. Crime Index and Safety Statistics' and a detailed analysis of crime rates in California, focusing on San Francisco and Marin Counties.

The bottom screenshot shows the same application in a Colab environment. The 'Citizen Services' tab is active, showing a form with the query 'explain the waste management system in my city' and a 'Get Information' button. The output on the right, titled 'Government Response', provides a detailed overview of a city's waste management system, including sections on 'Collection and Transportation' and 'Recycling and Organics Collection'.

## 11. Known Issues:

- May run slowly if internet connection is weak or model size is large.
- Sometimes generates explanations or quiz questions that are inaccurate or incomplete.
- No login or authentication, so anyone with the Gradio link can access the application.



- Limited to features supported by the IBM Granite model.
- Requires Google Colab to run, so it cannot work offline.

## **12. Future Enhancements:**

Some of the future enhancements that can be added are:

- Add user authentication (login system) for better security.
- Improve the AI model to give more accurate and detailed results.
- Add real-time collaboration, so multiple learners or teachers can use it together.
- Allow direct deployment as a standalone website or mobile app for easier access.
- Improve UI with better formatting, layouts, and themes.
- Add support for more input formats like DOCX and TXT files.