CITIZEN AI PROJECT

INTRODUCTION

TEAM MEMBERS:

- 1.ROHINI – 212304698
- 2.ROJA – 212304699
- 3.RUTHRAPRIYA -212304690
- 4.SAKTHIPRIYA – 212304691

PROJECT OVERVIEW

1. Introduction

Citizen AI is an intelligent citizen engagement platform that leverages IBM Granite large language models (LLMs) hosted on Hugging Face to provide quick, accurate, and context-aware responses about government services and civic issues. The project empowers citizens to interact with AI through a simple Gradio interface, while officials can view sentiment analysis and feedback on dashboards.

🔹 2. Objectives

1. Build an AI-based system that provides instant responses about civic and government services.
2. Integrate IBM Granite Models for natural language understanding and generation.
3. Provide a Gradio-based user interface for easy citizen interaction.
4. Deploy the solution in Google Colab for cost-effective execution with GPU acceleration.
5. Store, manage, and share code using GitHub.

🔹 3. Features

- User-Friendly Interface: Citizens can interact through a web-based Gradio UI.
- Natural Language Understanding: Powered by granite3.2-2b-instruct, a lightweight but effective LLM.

- Cloud Deployment: Runs on Google Colab (T4 GPU) for easy and affordable execution.
- Sentiment Tracking: Dashboards summarize public feedback for administrators.

🔷 4. System Architecture

The system follows a modular design with the following components:

1. User (Citizen/Student) – Interacts via the web interface.

2. Gradio UI – Provides a simple input-output interface for queries.

3. IBM Granite Model (Hugging Face) – Processes queries and generates responses.

4. Google Colab Runtime (T4 GPU) – Execution environment for hosting the Gradio app and running the AI model.

5. Feedback Dashboard – Tracks citizen queries, responses, and sentiment.

6. GitHub Repository – Stores code, project files, and version history.

5. Project Workflow

The project follows 4 main activities:

1. Activity 1 – Explore Naan Mudhalvan Smart Internz Portal

Access resources and workspace for guidance.

2. Activity 2 – Choose IBM Granite Model from Hugging Face.

Create Hugging Face account and select granite-3.2-2binstruct model

3. Activity 3 – Run in Google Colab

Setup T4 GPU runtime.

Install dependencies (transformers, torch, gradio).

Execute code and launch Gradio app.

4. Activity 4 – Upload Project to GitHub

Create repository.

Upload project files and commit changes.

6. Technologies Used

- Programming Language: Python
- Libraries: Transformers, Torch, Gradio.

- AI Model: IBM Granite (granite-3.2-2b-instruct) from Hugging Face.
- Platform: Google Colab (with T4 GPU)
- Version Control: Git & GitHub

🔹 7. Advantages

- Cost-effective (runs on free Colab GPUs).
- Easy to implement (students can follow step-by-step).
- Improves civic engagement with AI-driven insights.
- Scalable (more models and dashboards can be added later).
- Open-source and collaborative.

🔹 8. Limitations

- Dependent on Google Colab runtime (limited session time).
- Responses depend on the accuracy of the Granite model.
- Requires internet access for Hugging Face and Colab integration.

🔹 9. Future Enhancements

- Deploy on IBM Cloud or AWS for permanent hosting.
- Add multilingual support for citizens speaking regional languages.
- Enable voice-based interaction using speech-to-text models.
- Enhance dashboards with real-time analytics and sentiment graphs.

- Integrate with government portals/APIs for up-to-date information.

Architecture of Citizen AI Project

🔹 1. User Interaction Layer (Frontend)

- **Actor**: Citizens / Students / Endusers
- **Interface**: Gradio Web UI
- **Functionality**:
- Allows users to type questions about civic/government services.
- Provides a simple chat-like interface.
- Displays AI-generated answers in real time.
- Technology Used: Python + Gradio

🔹 2. Application Logic Layer

- The core intelligence of the system.
- IBM Granite Model (via Hugging Face)
- Pre-trained granite-3.2-2b-instruct model.
- Handles understanding and generating natural language answers.
- Python Code (Transformers + Torch)
- Loads the model and connects it with Gradio.
- Manages query processing and response delivery.
- Sentiment & Dashboard Module
- Tracks user queries and responses.
- Aggregates feedback for officials.

🔹 3. Execution & Hosting Layer (Backend Runtime) Platform: Google Colab (T4 GPU runtime)

Responsibilities:

- Runs all Python code and executes the model.
- Hosts the Gradio interface so users can interact with the system.

Advantages:

- Free and cloud-based (no local setup required).
- GPU acceleration for smooth model performance.

🔹 4. Data Management & Storage Layer

- GitHub Repository
- Stores code, notebooks, and documentation.
- Provides version control and team collaboration.
- Feedback Dashboard
- Logs queries and responses.
- Provides a visual summary of citizen sentiment.

🔹 5. Architecture Flow (Step-by-Step)

1. User enters a query in the Gradio UI.

2. The query is sent to the Application Logic Layer in Google Colab.

3. The Granite Model (Hugging Face) processes the query and generates an answer.

4. The response is sent back to the Gradio UI and shown to the user.

5. Queries & responses are forwarded to the Feedback Dashboard for analysis.

6. Project code and updates are stored in the GitHub Repository.

Citizen AI Project – Detailed Setup Instruction

🔹 1. Pre-requisites

Before starting, make sure you have:

- Basic Knowledge of Python programming.
- Familiarity with Google Colab.
- A GitHub account.
- A Hugging Face account to access IBM Granite models.

🔹 2. Step 1: Access Naan Mudhalvan Smart Internz Portal

1. Open your web browser and go to: https://naanmudhalvan.smartinternz.com
2. Login with your student credentials.
3. Navigate to Projects Section → Select Citizen AI Project.
4. Click Access Resources → Guided Project.
5. Go to Workspace → Here you'll find project progress tracking and instructions.

🔹 3. Step 2: Setup Hugging Face & Select IBM Granite Model

1. Go to https://huggingface.co.
2. Sign up / Login using your email.
3. In the search bar, type: IBM Granite.
4. Select the model: granite-3.2-2b-instruct (recommended for this project).

This model is lightweight, efficient, and fast for studentlevel projects.

5. Save the model page link for future use.

🔹 4. Step 3: Run the Project in Google Colab

1. Open Google Colab.
2. Click New Notebook.
3. Rename it as Citizen AI.
4. Change runtime settings:

Go to Runtime → Change runtime type.

Set Hardware accelerator = T4 GPU.

Click Save.

Install Required Libraries

In the first cell of Colab, run:

citizenAI.ipynb

File Edit View Insert Runtime Tools Help

[1]

```python
!pip install transformers torch gradio -q
```

[2]

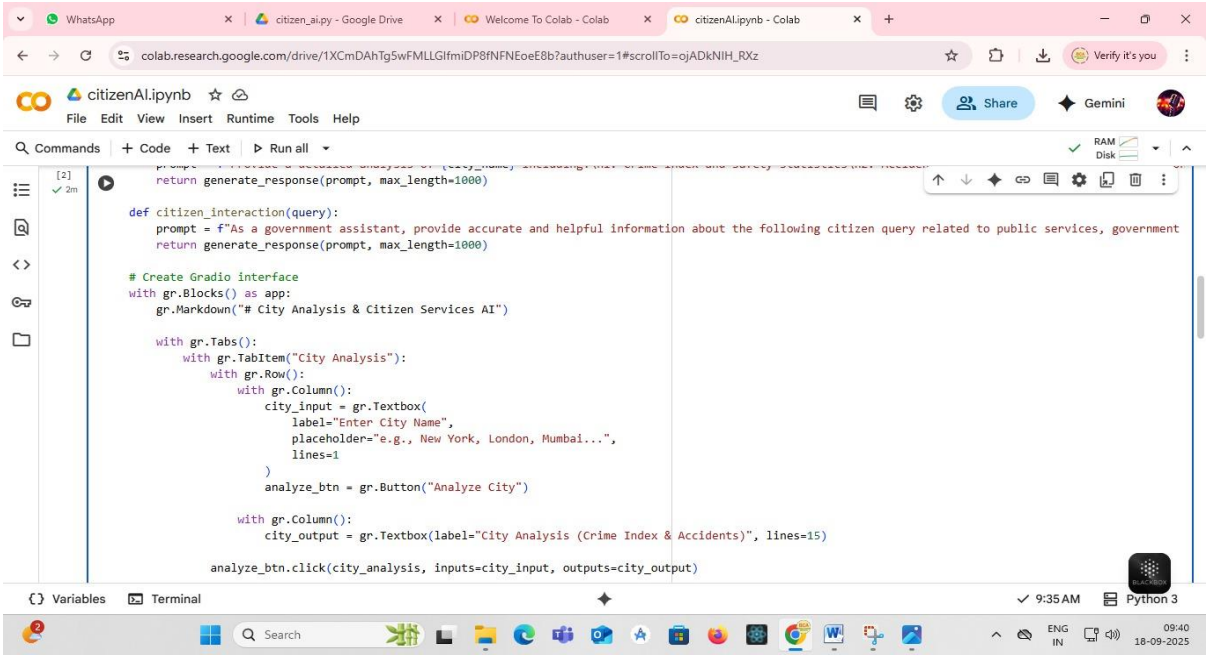
```python
import gradio as gr
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM

# 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():
```

citizenAI.ipynb

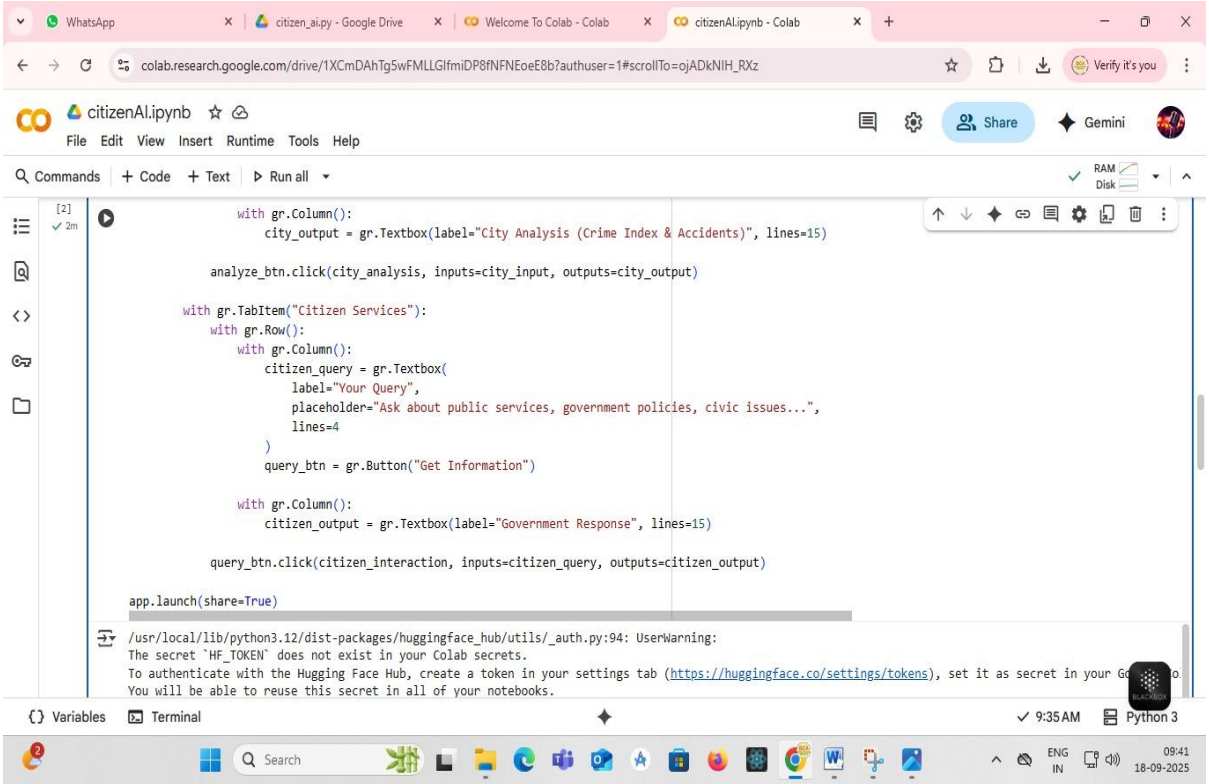
File Edit View Insert Runtime Tools Help

[2]

```python
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 informatior
    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, go
```



```
        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, government
    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)

            analyze_btn.click(city_analysis, inputs=city_input, outputs=city_output)
```



```
                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)
```

```
/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 Go
You will be able to reuse this secret in all of your notebooks.
```

!pip install transformers torch gradio -q

Import and Setup Python Code

In the next cell, paste this sample code (modify if needed):

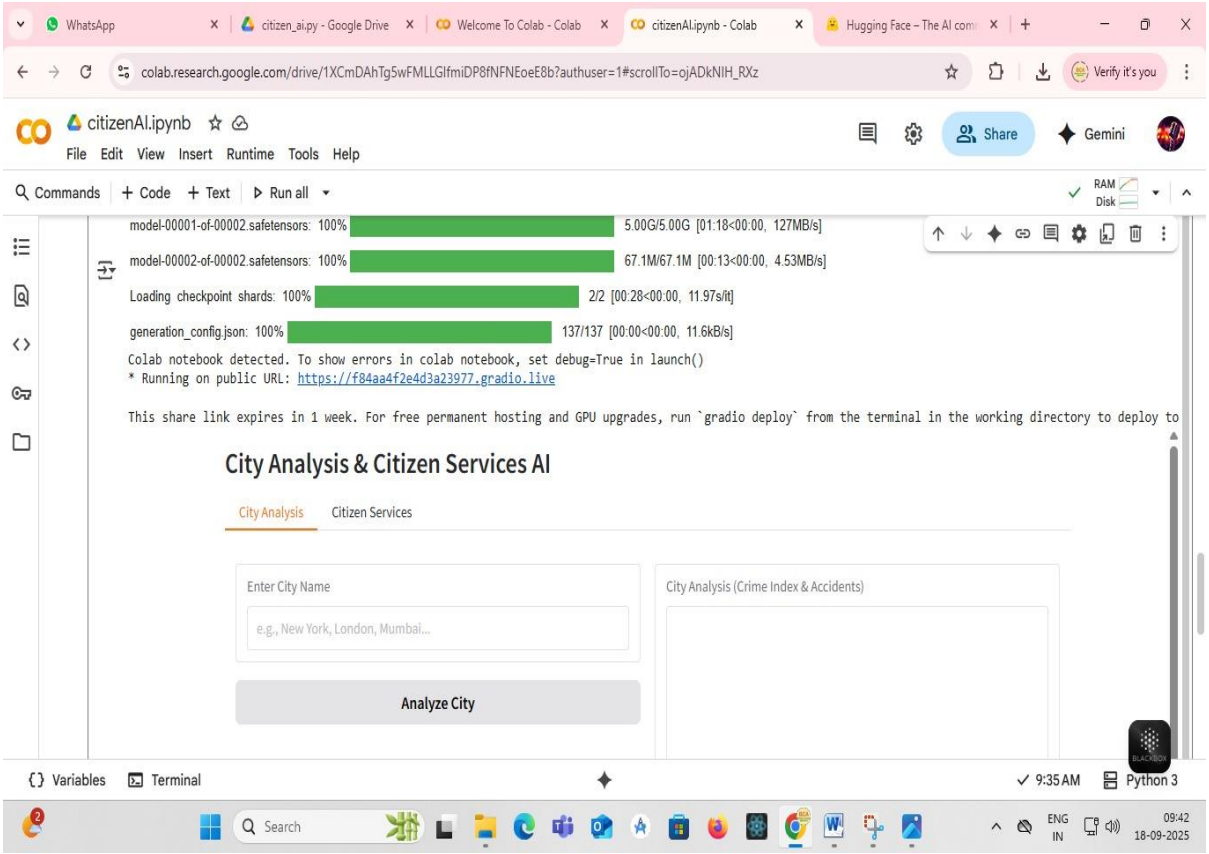
```
import gradio as gr from
transformers import pipeline

# Load IBM Granite model model_name = "ibm-
granite/granite-3.2-2b-instruct" generator =
pipeline("text-generation", model=model_name)

# Define chatbot function
def citizen_ai(query):
   response = generator(query, max_length=200,
do_sample=True)     return
response[0]['generated_text']

# Create Gradio interface interface =
gr.Interface(fn=citizen_ai, inputs="text",
outputs="text", title="Citizen AI") interface.launch()
```

Run the Notebook

Execute the code cells one by one.

Once it runs successfully, Colab will generate a Gradio app link.

Click the link to open the Citizen AI chatbot in your browser.



🔷 5. Step 4: Upload Project to GitHub

1. Go to https://github.com.
2. Sign up / Login to your account.
3. Click New Repository → Name it (e.g., Citizen-AIProject).
4. Check Add a README file.

5. In Google Colab, click File → Download .py to save your project script.

6. In GitHub → Open your repo → Add File → Upload Files.

7. Select your project file (.py) and upload.

8. Click Commit changes.

Your project is now stored in GitHub.

🔹 6. Step 5: Feedback Dashboard (Optional)

You can extend your project by adding a dashboard:

Collect user queries and responses in a CSV file.

Use Python libraries like Matplotlib / Plotly to generate sentiment charts.

Display insights about public opinion for administrators.

🔹 7. Verification of Setup

If everything is correct, you should be able to:

Open the Gradio link in your browser.

Ask questions about government services.

Get AI-generated responses.

Upload your final project code to GitHub for submission.

Running the Citizen AI Application

🔹 Step 1: Open Google Colab

1. Go to Google Colab.
2. Click on File → New Notebook.
3. Rename the notebook to CitizenAI.

⬙ Step 2: Setup Runtime

1. In the menu, go to Runtime → Change runtime type.
2. Set:
    - Hardware accelerator = GPU (T4) 3.

Click Save.

⬙ Step 3: Install Dependencies

Run the following command in the first cell:

```
!pip install transformers torch gradio -q
```

This installs:

- transformers → for Hugging Face model handling.
- torch → for deep learning backend.
- gradio → for creating the web interface.

⬙ Step 4: Load the IBM Granite Model

In a new cell, paste: from

transformers import pipeline

import gradio as gr

Load IBM Granite model from Hugging Face model_name =

"ibm-granite/granite-3.2-2b-instruct"

generator = pipeline("text-generation", model=model_name)

Define chatbot function

def citizen_ai(query):

response = generator(query, max_length=200, do_sample=True) return response[0]['generated_text']

⬙ Step 5: Create Gradio Application

Now, add this code in the next cell: # Create Gradio

interface interface = gr.Interface(fn=citizen_ai,

inputs="text", outputs="text", title="Citizen AI -

Intelligent Citizen Engagement"

)

Launch app

interface.launch(share=True) # share=True gives you a public link

---

⬙ Step 6: Run the Notebook

1. Run all the cells in order.
2. Wait for the model to load (this may take some time).
3. At the end, Colab will display a Gradio link (something like https://xxxxx.gradio.live).
4. Click that link → It will open the Citizen AI chatbot in a new browser tab.

⬙ Step 7: Interact with Citizen AI

- Type in a question (e.g., *“What is the procedure to apply for a driving license?”*).
- The model will generate an answer.
- Try multiple queries to test.
