Project Title:

Citizen AI

Team Members
Dinesh.R
Joy immanuvel.E
Naveen.G
Praveen Kumar.P

Project Overview
Purpose:
This AI assistant empowers citizens and city officials by providing detailed safety and crime analysis of cities and by answering citizen queries related to public services, government policies, and civic issues. Leveraging the IBM Granite LLM (Large Language Model), it offers natural language insights through an interactive conversational interface built with Gradio.

Features:

City Safety and Crime Analysis
Provides automated, detailed reports on crime index, safety statistics, accident rates, and overall city safety assessments using AI-generated responses.

Citizen Query Handling
Responds to public queries with accurate and helpful information about government services, policies, and civic issues via natural language interaction.

Gradio Web UI
An intuitive, tabbed interface allowing users to interact with analysis and query modules easily.

IBM Granite LLM Integration
Uses the IBM Granite 3.2 2B-instruct model for generating knowledgeable and context-aware responses.

Architecture
Frontend (Gradio):

Implements a user-friendly, block-based interface with tabs for city analysis and citizen services.

Input components for city name and citizen queries with output textboxes for AI responses.

Button triggers to invoke backend model functions and display results live.

Backend (Transformers model):

Loads the Granite LLM model and corresponding tokenizer from Hugging Face transformers.

Generates responses based on prompts tailored for either city safety analysis or government

query answering.

Supports both CPU and GPU execution transparently.

Model Handling:

Uses AutoModelForCausalLM for causal language modeling to generate natural language completions.

Tokenizer configured with padding and truncation to handle input prompt size limits.

Dynamically moves tensors to GPU if available for efficient processing.

Core Functional Modules
generate_response(prompt, max_length=1024)
Takes a textual prompt, tokenizes it, and generates a language model response with sampling temperature and length control.

city_analysis(city_name)
Constructs a tailored prompt requesting detailed crime, accident, and safety information for the specified city and generates the text response.

citizen_interaction(query)
Prepares a prompt to provide government-related answers to citizen questions about services, policies, and civic matters.

User Interface Details
City Analysis Tab:

Text input for city name (e.g., New York, London, Mumbai).

Button to analyze and fetch AI-generated city safety and crime report.

Output textbox to display analysis.

Citizen Services Tab:

Multi-line textbox for users to enter queries related to government and civic issues.

Button to fetch AI-generated responses.

Output textbox for display of provided information.

Setup Instructions
Prerequisites:

PyTorch with CUDA support (optional for GPU acceleration)

Transformers and Gradio libraries installed (pip install torch transformers gradio)

Installation and Running:

Clone or copy the script.

Install dependencies via pip.

Run the script to launch the Gradio app locally in a browser.

Use the UI to interact with AI models for city analysis and citizen query answering.

Future Enhancements
Add user authentication for personalized sessions.

Expand model capability with further fine-tuning on local government datasets.

Integrate with live city data sources (crime databases, traffic systems) for real-time updates.

Include more modules (resource forecasting, anomaly detection) similar to IBM’s Sustainable Smart City Assistant for holistic smart city management.

[1]
11s

```
!pip install transformers torch gradio
```

[ ]

```
import gradio as gr
import torch
from transformers import AutoTokenizer

# Load model and tokenizer
model_name = "ibm-granite/granite-3.2-
tokenizer = AutoTokenizer.from_pretrai
model = AutoModelForCausalLM.from_pret
    model_name,
    torch_dtype=torch.float16 if torch
    device_map="auto" if torch.cuda.is
)

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

def generate_response(prompt, max_leng
    inputs = tokenizer(prompt, return_

    if torch.cuda.is_available():
        inputs = {k: v.to(model.device

    with torch.no_grad():
        outputs = model.generate(
            **inputs,
            max_length=max_length,
            temperature=0.7,
            do_sample=True,
            pad_token_id=tokenizer.eos
        )

    response = tokenizer.decode(output
    response = response.replace(prompt
    return response

def city_analysis(city_name):
    prompt = f"Provide a detailed anal
    return generate_response(prompt, m
```

6:15

colab.research.google.com/c

citizenAI0.ipynb

RAM
Disk

[ ]

```
                with gr.Column():
                    citizen_output = g

            query_btn.click(citizen_in

app.launch(share=True)
```

```
tokenizer_config.json:  8.88k/? [00:00<00:00,
vocab.json:  777k/? [00:00<00:00, 5.67MB/s]
merges.txt:  442k/? [00:00<00:00, 5.86MB/s]
tokenizer.json:  3.48M/? [00:00<00:00, 36.8M
added_tokens.json: 100%  87.0/87.0 [00:00<00
special_tokens_map.json: 100%  701/701 [00:0
config.json: 100%  786/786 [00:00<00:00, 15.5
`torch_dtype` is deprecated! Use `
model.safetensors.index.json:  29.8k/? [00:00
Fetching 2 files: 100%  2/2 [03:13<00:00, 193
model-00002-of-00002.safetensors: 100%  67.1M/67.1M [00:13
model-00001-of-00002.safetens 100%  5.00G/5.00G [03:13
Loading checkpoint shards: 100%  2/2 [00:34
```

6:15

colab.research.google.com/c

citizenAI0.ipynb

RAM
Disk

[ ]

```
                    with gr.Column():
                        citizen_output = g

                query_btn.click(citizen_in

app.launch(share=True)
```

```
):00<00:00, 15.5

ated! Use `dtype` instead!

 29.8k/? [00:00

3:13<00:00, 193

M/67.1M [00:13

G/5.00G [03:13

)%  2/2 [00:34

 37/137 [00:00<

. To show errors in colab notebook,
: https://0f5a41489aa0fca386.gradio

 in 1 week. For free permanent host
```