Detailed Documentation for citizen_ai.py

The script `citizen_ai.py` is a Python-based project that integrates a Large Language Model (LLM) with Gradio to provide two main services: (1) City Safety Analysis and (2) Citizen-Government Interaction. It demonstrates the use of Hugging Face Transformers for natural language processing and Gradio for building interactive user interfaces. The program is intended to run in Google Colab with optional GPU support.

1. Dependencies

The script installs and imports the following dependencies: - **transformers**: Provides the `AutoTokenizer` and `AutoModelForCausalLM` classes used to load and run the language model. - **torch**: Handles tensor computations and GPU acceleration. - **gradio**: Creates a web-based user interface for interacting with the model. Installation is done using pip in Colab: `!pip install transformers torch gradio -q`

2. Model Loading

The script loads the IBM Granite model (`ibm-granite/granite-3.2-2b-instruct`) via Hugging Face Hub. - **AutoTokenizer**: Converts text into token IDs and back. - **AutoModelForCausalLM**: Loads a causal language model suitable for text generation. - **GPU Detection**: Uses `torch.cuda.is_available()` to decide whether to use GPU with float16 precision or CPU with float32. - **Pad Token Handling**: If the tokenizer has no pad token, it assigns the EOS (end-of-sequence) token as the pad token.

3. Function: generate_response

This core function is responsible for generating AI responses: Steps: 1. Tokenizes the input prompt (with truncation at 512 tokens). 2. Moves the tensors to GPU if available. 3. Calls `model.generate()` with parameters: - `max_length`: Maximum response length. - `temperature=0.7`: Controls creativity vs. determinism. - `do_sample=True`: Ensures varied, non-greedy outputs. - `pad_token_id`: Prevents errors during generation. 4. Decodes the generated output and removes the original prompt. This function is reused by both city and citizen query analysis.

4. Function: city_analysis

This function builds a structured prompt asking the model to analyze a given city. The analysis covers: - Crime index and safety statistics. - Accident and traffic safety information. - General safety assessment. It then calls `generate_response` to produce the AI-generated text.

5. Function: citizen_interaction

This function handles citizen queries about governance, policies, and public services. It frames the query in a government-assistant style prompt to ensure responses remain formal and helpful. It is particularly useful for answering civic or administrative questions.

6. Gradio Interface

The Gradio interface is created using `gr.Blocks()` with two tabs: 1. **City Analysis Tab**: - Input: Textbox for entering city name. - Output: Multi-line textbox displaying city analysis. - Action: Clicking 'Analyze City' runs the `city_analysis` function. 2. **Citizen Services Tab**: - Input: Textbox for entering a query. - Output: Multi-line textbox displaying government-style response. - Action: Clicking 'Get Information' runs the `citizen_interaction` function.

7. Application Launch

The application is launched with: `app.launch(share=True)` This starts a local Gradio server and provides a shareable public link so others can use the interface.

8. Potential Improvements

The script can be extended in the following ways: - **Data Integration**: Connect real-time APIs (crime reports, traffic stats) instead of relying solely on model-generated text. - **Error Handling**: Add try-except blocks for network errors or invalid inputs. - **UI Enhancements**: Provide charts or maps for better visualization. - **Customization**: Allow users to set parameters like temperature, max tokens, etc.

Appendix: Full Source Code

```
# run this project file in google collab by changing run type to T4 GPU

!pip install transformers torch gradio -q

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()}

    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 stati
    return generate_response(prompt, max_length=1000)

def citizen_interaction(query):
    prompt = f"As a government assistant, provide accurate and helpful information about the followi
    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=

            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)
```