Citizen AI: Intelligent Citizen Engagement Platform

Generative AI with IBM

CITIZEN AI

1. Introduction

• **Project Title**: Citizen AI

- **Team Leader :** Stefficlara F
- **Team member :** Rekha S
- **Team member :** Yazhini R
- **Team member :** Sujitha S

2. Project Overview

Purpose:
The AI app is designed as a **Health & Wellness Advisory Assistant** that serves two main functions:

- **Health Analysis:** Generates detailed health and wellness reports for a specified city, including common health issues, wellness programs, and recommendations.
- **Citizen Health Queries:** Provides accurate, clear, and helpful advice to citizens regarding public health, wellness programs, or health policies based on their questions.

This helps governments or health organizations provide automated, AI-driven support to citizens and stakeholders.

Conversation Interface:
The app uses a **conversational interface** where users input either:

- A city name to get a health analysis report, or
- A health-related question to get a government health response.

The AI model (GPT-2 medium) generates natural language responses based on these inputs, simulating a helpful government health assistant.

Policy Summarization:
While the current app focuses on health analysis and answering queries, it can be extended to **summarize health policies** by prompting the model to generate concise summaries of

complex health regulations or wellness programs, making them easier for citizens to understand.

---

3. Architecture

The app architecture consists of two main parts: **Frontend** and **Backend**.

Frontend: Streamlit (Suggested Alternative)

Although your current code uses **Gradio** for the UI, a similar frontend can be built with **Streamlit**, which is another popular Python framework for building interactive web apps.

- **User Interface Components:**
    - Text input boxes for city names and health questions.
    - Sliders for controlling generation parameters like temperature (creativity) and max tokens (response length).
    - Buttons to trigger the generation of responses.
    - Text areas to display the generated health reports or answers.
- **Streamlit Advantages:**
    - Simple to set up and deploy.
    - Supports real-time interaction.
    - Easy integration with Python ML models.

Backend: FastAPI

For production-grade deployment, the backend can be implemented using **FastAPI**, a modern, fast (high-performance) web framework for building APIs with Python 3.7+.

- **Responsibilities:**
    - Load and serve the AI model (GPT-2 medium or any other).
    - Receive requests from the frontend with user inputs.
    - Run the text generation pipeline.
    - Return generated responses as JSON.
- **Benefits of FastAPI:**

- Asynchronous support for high throughput.
- Automatic interactive API docs (Swagger UI).
- Easy integration with ML models and GPU acceleration.
- Scalable and suitable for containerization (Docker).

4. Setup Instructions

Prerequisites:

- Python 3.9+

- pip and virtual environment tools

- Gradio - Hugging Face Transformers

- Torch with CUDA support (optional)

Installation:

- Clone the repository
- Install dependencies: pip install -r requirements.txt
- Run the app: python app.py
- Access the Gradio link to interact with Citizen AI

5. Folder Structure

EduTutor-AI/

|— app.py # Main application script

|— requirements.txt # Dependencies

|— docs/ # Documentation files

|— models/ # Pretrained model references

|— utils/ # Helper functions (if extended later)

6. Running the Application

- Launch the Gradio interface by running app.py

- Navigate between Concept Explanation and Quiz Generator tabs

- Input the desired topic and view the AI-generated output in real time

7. API Documentation

Health Analysis: Generates a detailed health and wellness report for a specified city.

Citizen Health Queries: Provides accurate, clear, and helpful advice regarding public health, wellness programs, or health policies based on user queries.

8. Authentication

- This demo app does **not** implement authentication.
- For production deployment, consider adding:
    - API key or token-based authentication.
    - OAuth 2.0 for user authorization.
    - Rate limiting to prevent abuse.
- Gradio apps can be protected via password or hosted behind authenticated gateways.

9. User Interface

The app uses **Gradio** to provide a simple web UI with two tabs:

Tab 1: Health Analysis

- **Input:** Textbox for city name.
- **Controls:** Sliders for temperature (0.1–1.0) and max tokens (100–1024).
- **Output:** Multi-line textbox showing the generated health report.
- **Button:** "Analyze Health" triggers the generation.

Tab 2: Citizen Health Queries

- **Input:** Multi-line textbox for health-related questions.
- **Controls:** Sliders for temperature and max tokens.
- **Output:** Multi-line textbox showing the AI-generated response.
- **Button:** "Get Health Advice" triggers the generation.

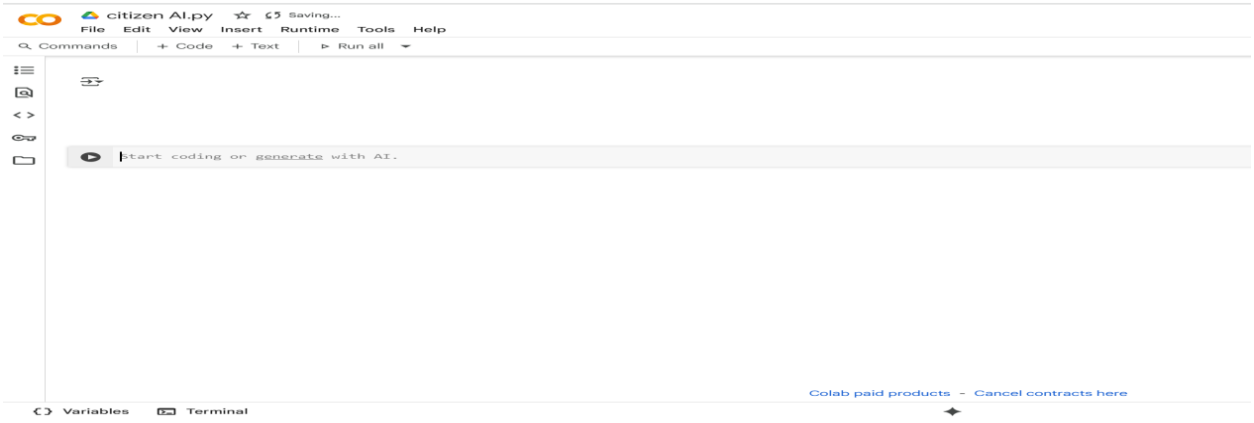
The UI is responsive and uses columns and rows for layout clarity.

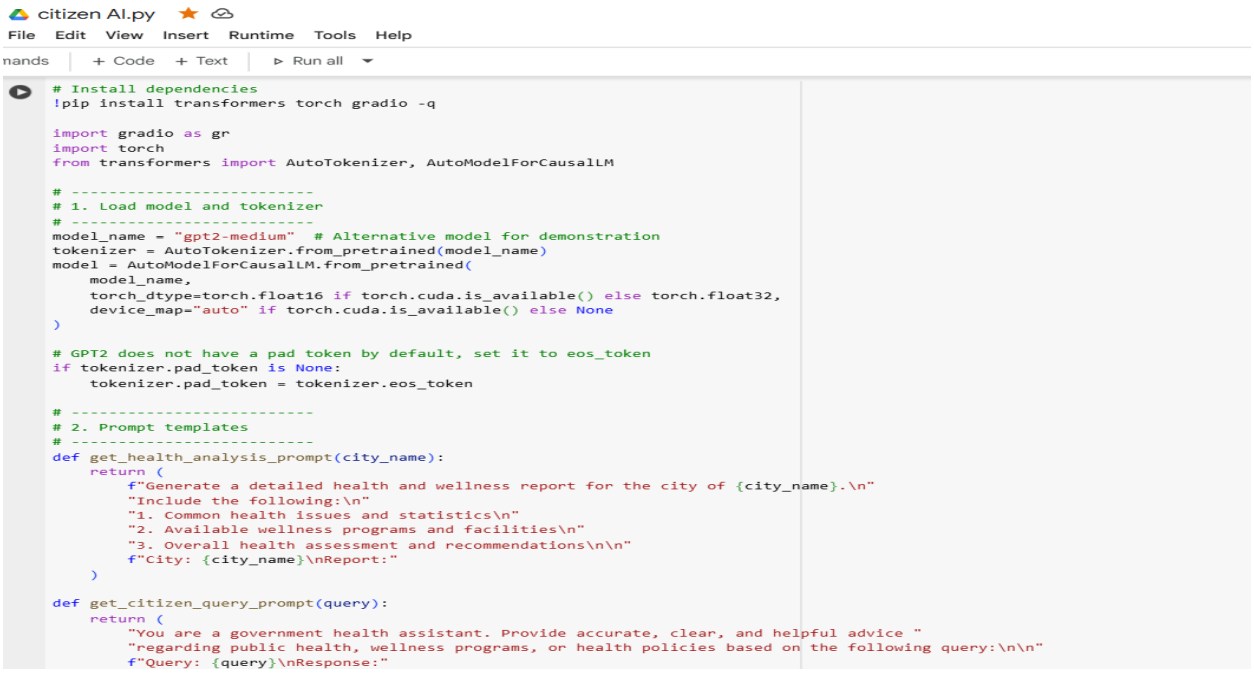
---

10. Testing

- Run the app locally or on Colab.
- Test with valid and invalid inputs:
    - Empty city name or query should return validation messages.
    - Vary temperature and max tokens to observe output diversity.
- Verify outputs are relevant and coherent.

11. Screenshots





```python
# ----------------------------
def health_citizen_query(query, temperature, max_tokens):
    query = query.strip()
    if not query:
        return "Please enter a valid health-related question."
    prompt = get_citizen_query_prompt(query)
    return generate_response(prompt, max_length=max_tokens, temperature=temperature)

# ----------------------------
# 6. Build Gradio app
# ----------------------------
with gr.Blocks() as app:
    gr.Markdown("# Health & Wellness Advisory AI (Alternative)")

    with gr.Tabs():
        # Tab 1: Health Analysis
        with gr.TabItem("Health Analysis"):
            with gr.Row():
                with gr.Column(scale=1):
                    city_input = gr.Textbox(
                        label="Enter City Name",
                        placeholder="e.g., New York, London, Mumbai...",
                        lines=1
                    )
                    temperature_slider = gr.Slider(
                        minimum=0.1, maximum=1.0, value=0.7, step=0.05,
                        label="Temperature (Creativity)"
                    )
                    max_tokens_slider = gr.Slider(
                        minimum=100, maximum=1024, value=512, step=50,
                        label="Max Tokens"
                    )
                    analyze_btn = gr.Button("Analyze Health")

                with gr.Column(scale=2):
                    city_output = gr.Textbox(label="Health Analysis Report", lines=20)

            analyze_btn.click(
                health_analysis,
                inputs=[city_input, temperature_slider, max_tokens_slider],
```


```python
# ----------------------------
def health_citizen_query(query, temperature, max_tokens):
    query = query.strip()
    if not query:
        return "Please enter a valid health-related question."
    prompt = get_citizen_query_prompt(query)
    return generate_response(prompt, max_length=max_tokens, temperature=temperature)

# ----------------------------
# 6. Build Gradio app
# ----------------------------
with gr.Blocks() as app:
    gr.Markdown("# Health & Wellness Advisory AI (Alternative)")

    with gr.Tabs():
        # Tab 1: Health Analysis
        with gr.TabItem("Health Analysis"):
            with gr.Row():
                with gr.Column(scale=1):
                    city_input = gr.Textbox(
                        label="Enter City Name",
                        placeholder="e.g., New York, London, Mumbai...",
                        lines=1
                    )
                    temperature_slider = gr.Slider(
                        minimum=0.1, maximum=1.0, value=0.7, step=0.05,
                        label="Temperature (Creativity)"
                    )
                    max_tokens_slider = gr.Slider(
                        minimum=100, maximum=1024, value=512, step=50,
                        label="Max Tokens"
                    )
                    analyze_btn = gr.Button("Analyze Health")

                with gr.Column(scale=2):
                    city_output = gr.Textbox(label="Health Analysis Report", lines=20)

            analyze_btn.click(
                health_analysis,
                inputs=[city_input, temperature_slider, max_tokens_slider],
```

```
                    city_output = gr.Textbox(label="Health Analysis Report", lines=20)

            analyze_btn.click(
                health_analysis,
                inputs=[city_input, temperature_slider, max_tokens_slider],
                outputs=city_output
            )

        # Tab 2: Citizen Health Queries
        with gr.TabItem("Citizen Health Queries"):
            with gr.Row():
                with gr.Column(scale=1):
                    citizen_query = gr.Textbox(
                        label="Your Health Question",
                        placeholder="Ask about public health, wellness programs, or policies...",
                        lines=4
                    )
                    temperature_slider_q = gr.Slider(
                        minimum=0.1, maximum=1.0, value=0.7, step=0.05,
                        label="Temperature (Creativity)"
                    )
                    max_tokens_slider_q = gr.Slider(
                        minimum=100, maximum=1024, value=512, step=50,
                        label="Max Tokens"
                    )
                    query_btn = gr.Button("Get Health Advice")

                with gr.Column(scale=2):
                    citizen_output = gr.Textbox(label="Government Health Response", lines=20)

            query_btn.click(
                health_citizen_query,
                inputs=[citizen_query, temperature_slider_q, max_tokens_slider_q],
                outputs=citizen_output
            )

# ----------------------------
# 7. Launch app
# ----------------------------
app.launch(share=True)
```

OUTPUT: