**1. INTRODUCTION**

1.1 Project Overview  
1.2 Purpose

**1.1 Project Overview**

This project, **Citizen AI**, is an intelligent citizen engagement platform designed to streamline interactions between governments and citizens. Leveraging advanced AI technologies, particularly Flask, IBM Granite models, and IBM Watson, the platform provides real-time, AI-driven responses to citizen inquiries regarding government services, policies, and civic issues. Key features include natural language processing (NLP), sentiment analysis, and a dynamic analytics dashboard for policymakers. The platform aims to enhance citizen satisfaction, improve government efficiency, and build public trust in digital governance.

**1.2 Purpose**

The primary purpose of Citizen AI is to revolutionize how governments interact with citizens. Specifically, this project seeks to:

* **Automate routine inquiries:** Provide instant answers to frequently asked questions, freeing up government staff for more complex issues.
* **Improve citizen satisfaction:** Offer readily accessible and personalized information, leading to a more positive experience for the public.
* **Enable data-driven governance:** Provide valuable insights into public sentiment and emerging issues, allowing government agencies to proactively address concerns.
* **Increase government efficiency:** Automate tasks and reduce workload, allowing for more effective allocation of resources.

**2. IDEATION PHASE**

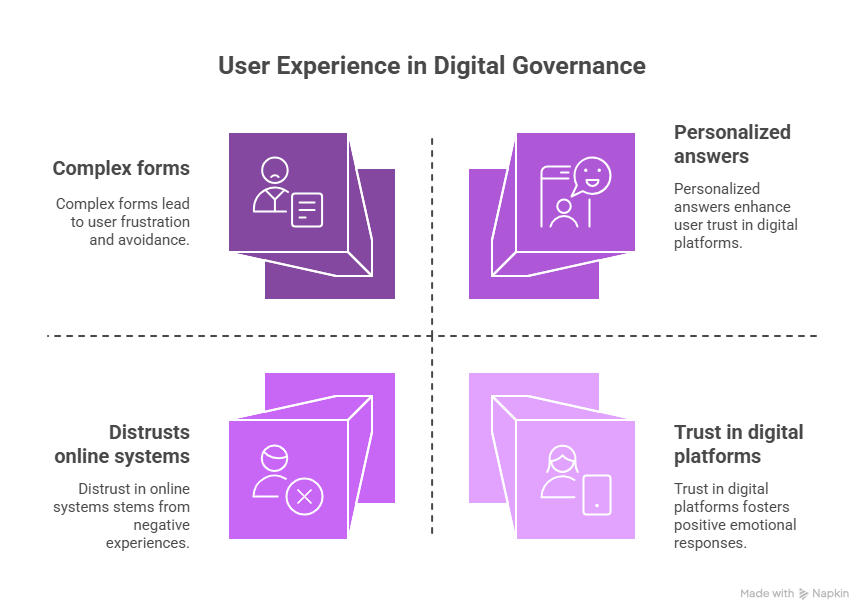
2.1 Problem Statement  
2.2 Empathy Map Canvas  
2.3 Brainstorming

**2.1 Problem Statement**

The current methods for citizen engagement in many governments are often inefficient and lead to frustration for both citizens and government officials. Problems include:

* **Long wait times:** Citizens may experience delays in receiving information or assistance.
* **Inconsistent information:** Information may not be readily accessible or may be outdated.
* **Limited interaction channels:** Citizens may be restricted to specific communication channels, hindering access.
* **Lack of understanding of public sentiment:** Governments may not have a clear picture of public concerns and satisfaction.

**2.2 Empathy Map Canvas**

****

*This section should visually represent a comprehensive empathy map including the perspectives of citizens, government officials, and other stakeholders.*

**2.3 Brainstorming**

Key brainstorming ideas and functionalities:

* Real-time conversational AI assistant
* Citizen sentiment analysis
* Dynamic dashboard for insights
* Personalized and contextual response system

**3. REQUIREMENT ANALYSIS**

3.1 Customer Journey map  
3.2 Solution Requirement  
3.3 Data Flow Diagram  
3.4 Technology Stack

**3.1 Customer Journey map**

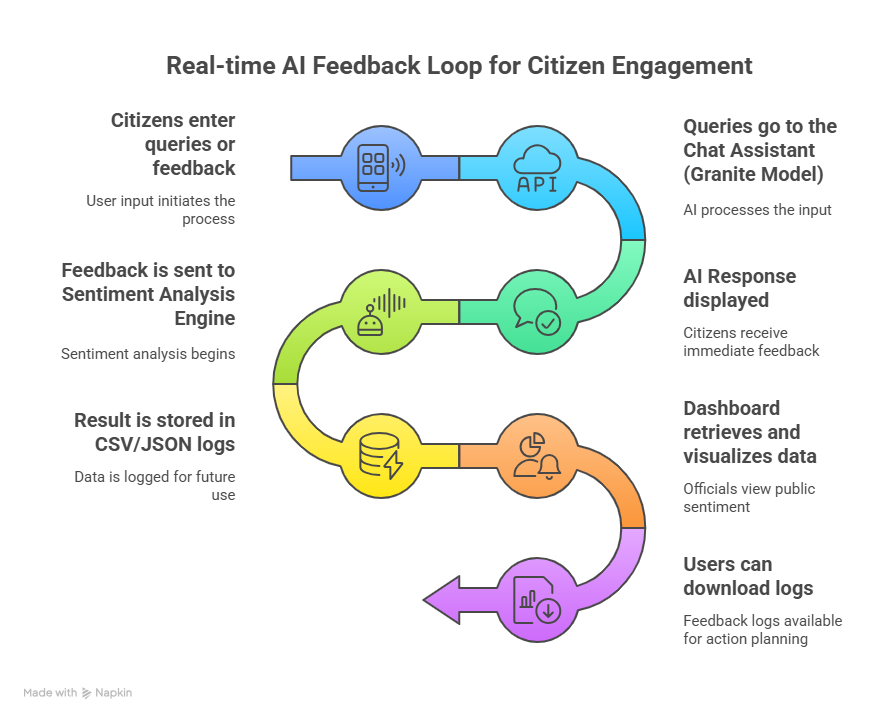
|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Step | Citizen Action | Touchpoints / Devices | Experience (Positive/Negative) | Goal / Motivation | Opportunities |
| Awareness | Hears about the Citizen AI platform via a government social post or email. | Mobile phone, social media, WhatsApp | Curious but skeptical about another gov-tech tool. | Help me get accurate civic info without waiting in queues. | Use testimonials, simple onboarding UI. |
| Interaction | Opens the Gradio app and enters a query about documentation or a service. | Web browser / Gradio UI | Feels impressed with speed and clarity of AI response. | Help me understand rules and get things done fast. | Enable voice input, show related FAQs. |
| Feedback | Submits service feedback through textbox after interaction. | Textbox, Feedback form | Satisfied when sentiment is recognized correctly. | Help me make my voice heard. | Allow emoji rating, tag suggestions. |
| Monitoring | Views public sentiment charts on the dashboard. | Dashboard tab, graphs | Feels reassured seeing transparency. | Help me know I’m not alone in facing this. | Add filter options for departments or topics. |
| Extension | Downloads CSV report for local review or sharing. | Download buttons | Happy to access offline or forward to officials. | Help me keep proof or share my opinion broadly. | Include timestamp and summary notes in download. |
| Future Use | Returns to ask another question next week. | Chatbot tab | Feels loyal to a tool that 'just works'. | Help me avoid misinformation next time. | Enable history tracking or bookmark feature. |

*This section should visually represent the customer journey of a citizen interacting with Citizen AI, highlighting key touchpoints and pain points.*

**3.2 Solution Requirement**

* **Real-time response:** The system must provide immediate and relevant responses to citizen queries.
* **Natural language understanding:** The platform should effectively interpret and understand the nuances of citizen questions.
* **Sentiment analysis:** Analyze citizen feedback to identify trends and prioritize issues.
* **Data visualization:** Present data in a clear and user-friendly dashboard.
* **Scalability:** The system should be able to handle a growing number of users and interactions.

**3.3 Data Flow Diagram**

****

*Visual representation of the data flow within Citizen AI, illustrating the flow of information from citizens to the platform and back to relevant government officials.*

**3.4 Technology Stack**

* **Backend:** Flask, Python
* **AI Models:** IBM Granite, IBM Watson
* **NLP:** Transformers
* **Frontend:** Gradio
* **Data Storage:** CSV, JSON

**4. PROJECT DESIGN**

4.1 Problem Solution Fit  
4.2 Proposed Solution  
4.3 Solution Architecture

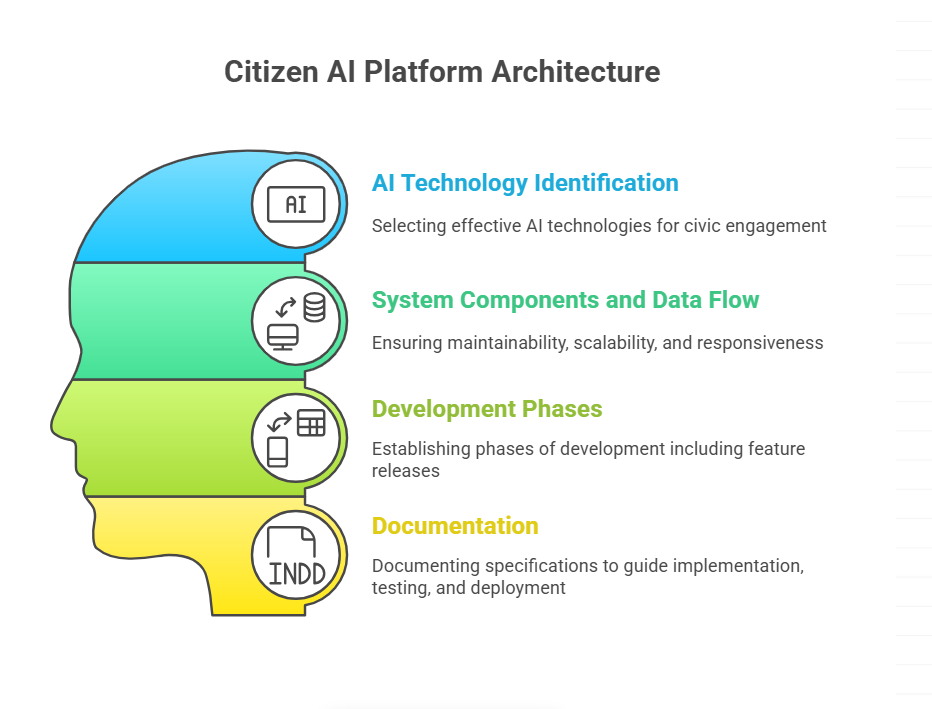
**4.1 Problem Solution Fit**

Citizen AI directly addresses the problems outlined in the problem statement, providing a solution that automates tasks, improves access, and allows for data-driven decision-making.

**4.2 Proposed Solution**

The solution will implement a conversational AI assistant powered by IBM Granite, analyzing citizens' inputs and responding in real time. Sentiment analysis will gauge public sentiment, and a dynamic dashboard will give policymakers a visual summary of interactions and sentiment.

**4.3 Solution Architecture**



**5. PROJECT PLANNING & SCHEDULING**

5.1 Project Planning

**5.1 Project Planning**

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Sprint / Epic | User Story Number | User Story / Task | Story Points | Priority | Team Members |
| Sprint-1: Chat Assistant | USN-1 | As a user, I can ask civic-related questions and get real-time responses using the AI chatbot. | 3 | High | Member A |
| Sprint-1: Chat Assistant | USN-2 | As a user, I should see the assistant response displayed clearly after submission. | 2 | Medium | Member A |
| Sprint-2: Sentiment Analysis | USN-3 | As a user, I can submit my feedback and view its sentiment classification. | 3 | High | Member B |
| Sprint-2: Dashboard | USN-4 | As an official, I can view a dynamic dashboard of feedback trends. | 4 | High | Member B |
| Sprint-3: Downloads | USN-5 | As an official, I can download sentiment logs as CSV or JSON. | 2 | Medium | Member C |
| Sprint-3: Error Handling | USN-6 | As a user, I am notified of invalid input or API issues gracefully. | 2 | Low | Member C |

**6. FUNCTIONAL AND PERFORMANCE TESTING**

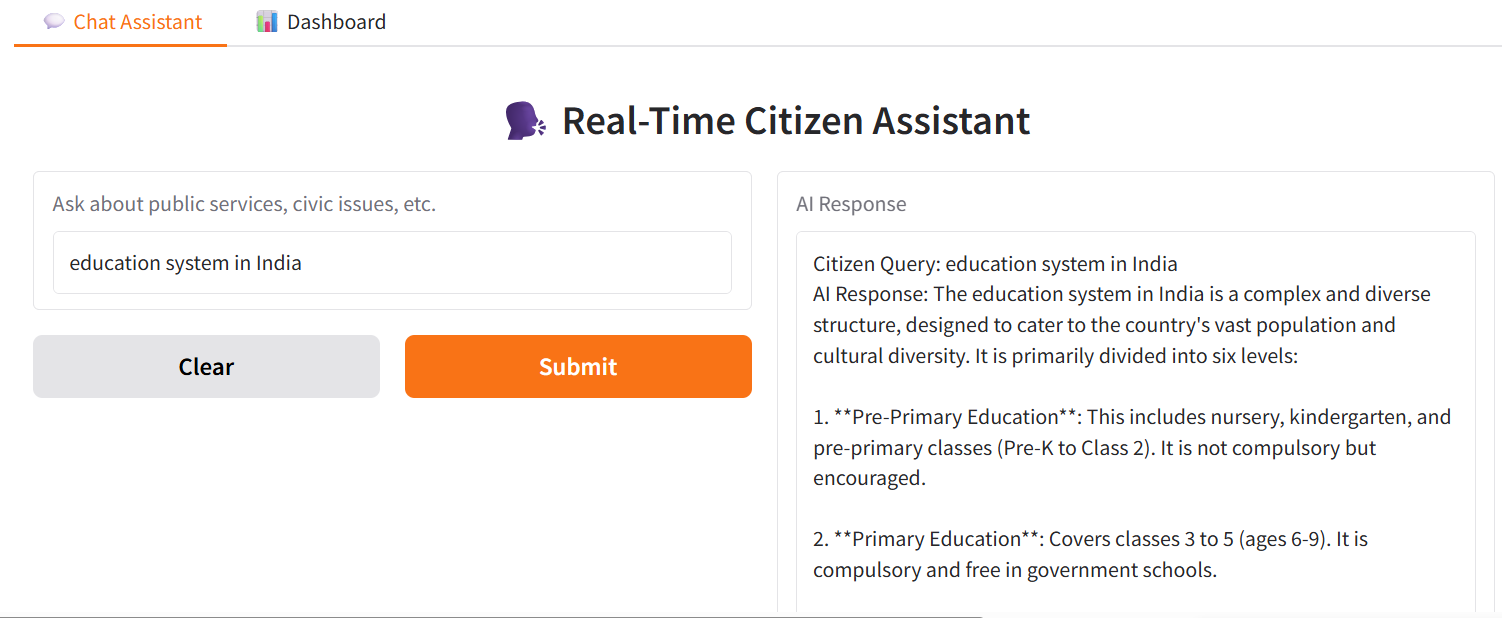
6.1 Performance Testing

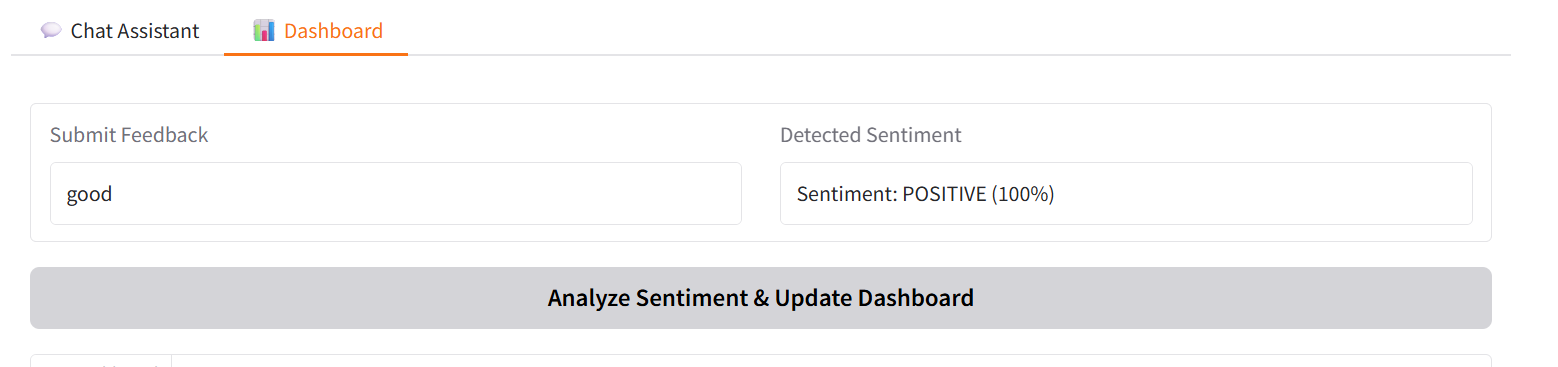
**6.1 Performance Testing**

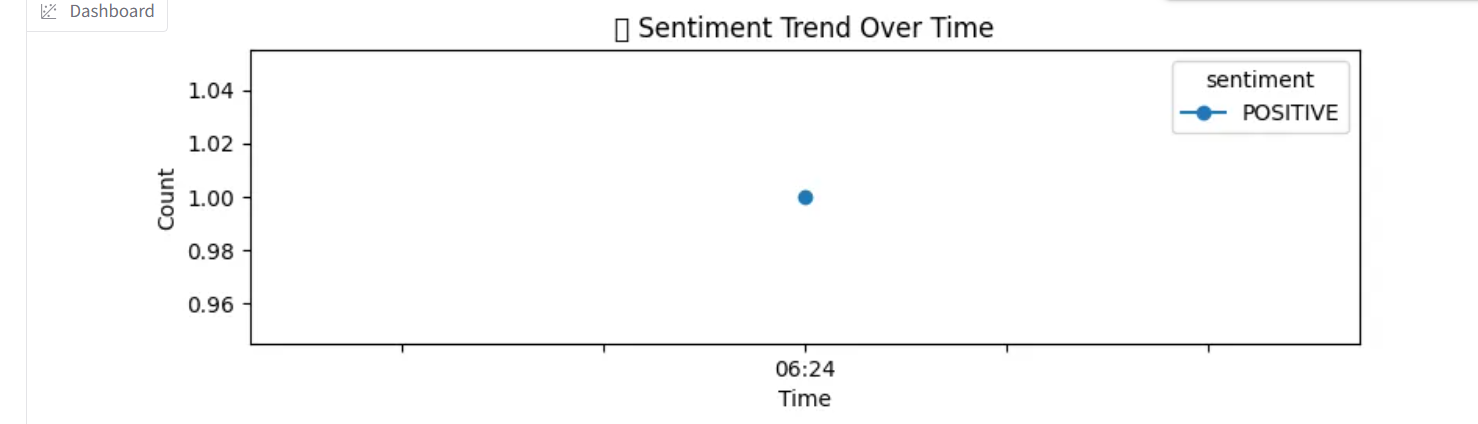
* **Load testing:** Test the system under various user loads to ensure responsiveness.
* **Response time testing:** Measure the time it takes for the system to respond to user queries.
* **Scalability testing:** Evaluate the system's ability to handle increased user traffic.

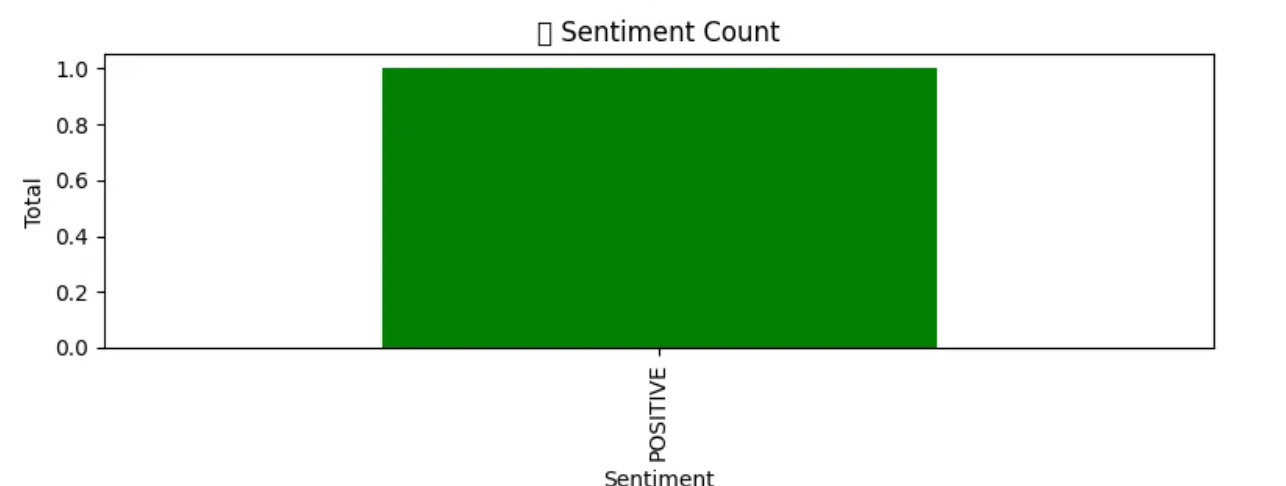
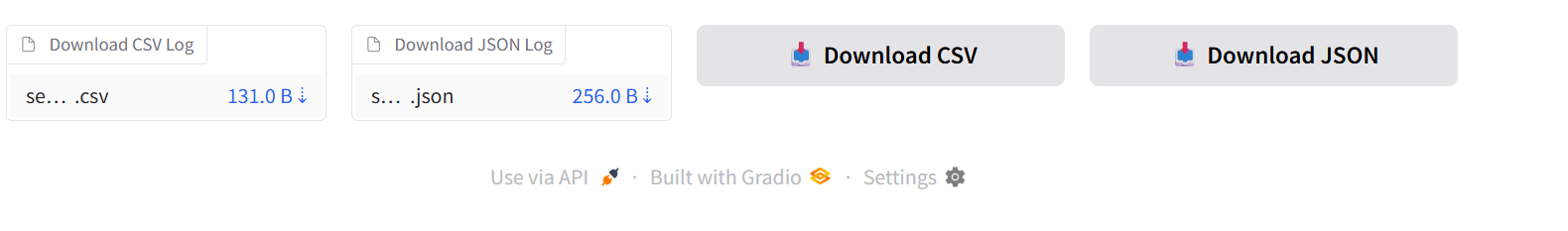
**7. RESULTS**

7.1 Output Screenshots







**8. ADVANTAGES & DISADVANTAGES**

**Advantages:**

* Improved citizen experience
* Increased government efficiency
* Data-driven decision making
* Reduced workload for government officials

**Disadvantages:**

* Potential for misinterpretation of citizen queries
* Dependency on accuracy of AI models
* Initial development costs

**9. CONCLUSION**

Citizen AI demonstrates a promising solution for improving government-citizen interactions. The platform addresses critical pain points through AI-driven solutions, offering a more efficient and user-friendly experience for all stakeholders.

**10. FUTURE SCOPE**

Future development could include:

* Integration with other government systems
* Expansion of language support
* Integration with social media data for sentiment analysis
* More sophisticated AI models for more nuanced responses

**11. APPENDIX**

**Source Code**

#Install Required Packages

!pip install -q gradio transformers torch accelerate matplotlib pandas requests

#Imports

import gradio as gr

import torch

import matplotlib.pyplot as plt

import pandas as pd

import json

import os

from datetime import datetime

from transformers import AutoTokenizer, AutoModelForCausalLM, pipeline

from huggingface\_hub import login

#Load IBM Granite Model

HF\_TOKEN = "hf\_token"  # Replace with your actual token

model\_id = "ibm-granite/granite-3.3-2b-instruct"

tokenizer = AutoTokenizer.from\_pretrained(model\_id, token=HF\_TOKEN)

model = AutoModelForCausalLM.from\_pretrained(

    model\_id,

    device\_map="auto",

    torch\_dtype=torch.float16,

    token=HF\_TOKEN

)

#Load Hugging Face Sentiment Model

sentiment\_pipeline = pipeline("sentiment-analysis")

#Real-Time Chat Function

def ask\_granite(prompt):

    inputs = tokenizer(prompt, return\_tensors="pt").to(model.device)

    outputs = model.generate(\*\*inputs, max\_new\_tokens=200)

    return tokenizer.decode(outputs[0], skip\_special\_tokens=True)

#Sentiment Analysis with Logging

sentiment\_counts = {"POSITIVE": 0, "NEGATIVE": 0, "NEUTRAL": 0}

interaction\_log = []

def submit\_feedback(text):

    result = sentiment\_pipeline(text)[0]

    sentiment\_label = result['label'].upper()

    score = round(result['score'] \* 100)

    if sentiment\_label not in sentiment\_counts:

        sentiment\_label = "NEUTRAL"

    sentiment\_counts[sentiment\_label] += 1

    # Log entry

    entry = {

        "timestamp": datetime.now().isoformat(),

        "text": text,

        "sentiment": sentiment\_label,

        "score": score

    }

    interaction\_log.append(entry)

    # Save to CSV

    csv\_file = "sentiment\_log.csv"

    df = pd.DataFrame([entry])

    df.to\_csv(csv\_file, mode='a', header=not os.path.exists(csv\_file), index=False)

    # Save to JSON

    json\_file = "sentiment\_log.json"

    if os.path.exists(json\_file):

        with open(json\_file, 'r') as f:

            data = json.load(f)

    else:

        data = []

    data.append(entry)

    with open(json\_file, 'w') as f:

        json.dump(data, f, indent=2)

    return f"Sentiment: {sentiment\_label} ({score}%)"

#Dashboard Charts

def plot\_dashboard():

    labels = list(sentiment\_counts.keys())

    sizes = list(sentiment\_counts.values())

    colors = ["green", "red", "gray"]

    fig, ax = plt.subplots()

    ax.pie(sizes, labels=labels, colors=colors, autopct="%1.1f%%")

    ax.set\_title("Overall Sentiment Distribution")

    return fig

def plot\_sentiment\_trends():

    if not interaction\_log:

        return None

    df = pd.DataFrame(interaction\_log)

    df['timestamp'] = pd.to\_datetime(df['timestamp'])

    df['minute'] = df['timestamp'].dt.strftime('%H:%M')

    fig, ax = plt.subplots(2, 1, figsize=(8, 6))

    # Line chart: Sentiment trend over time

    trend = df.groupby(['minute', 'sentiment']).size().unstack().fillna(0)

    trend.plot(ax=ax[0], marker='o')

    ax[0].set\_title("📈 Sentiment Trend Over Time")

    ax[0].set\_ylabel("Count")

    ax[0].set\_xlabel("Time")

    # Bar chart: Total sentiment count

    totals = df['sentiment'].value\_counts()

    totals.plot(kind='bar', ax=ax[1], color=["green", "red", "gray"])

    ax[1].set\_title("📊 Sentiment Count")

    ax[1].set\_ylabel("Total")

    ax[1].set\_xlabel("Sentiment")

    plt.tight\_layout()

    return fig

#File Download Helpers

def download\_csv():

    with open("sentiment\_log.csv", "r") as f:

        return f.read()

def download\_json():

    with open("sentiment\_log.json", "r") as f:

        return f.read()

#Gradio Interfaces

# Chat Assistant

def chat\_interface(user\_input):

    prompt = f"Citizen Query: {user\_input}\nAI Response:"

    return ask\_granite(prompt)

chat\_tab = gr.Interface(fn=chat\_interface,

                        inputs=gr.Textbox(label="Ask about public services, civic issues, etc."),

                        outputs=gr.Textbox(label="AI Response"),

                        title="🗣️ Real-Time Citizen Assistant")

# Dashboard

sentiment\_input = gr.Textbox(label="Submit Feedback")

sentiment\_output = gr.Textbox(label="Detected Sentiment")

dashboard\_plot = gr.Plot(label="Dashboard")

with gr.Blocks() as dashboard\_tab:

    with gr.Row():

        sentiment\_input.render()

        sentiment\_output.render()

    with gr.Row():

        submit\_btn = gr.Button("Analyze Sentiment & Update Dashboard")

        submit\_btn.click(fn=submit\_feedback, inputs=sentiment\_input, outputs=sentiment\_output)

        submit\_btn.click(fn=plot\_sentiment\_trends, outputs=dashboard\_plot)

    dashboard\_plot.render()

    # ✅ Download buttons (use File components)

    with gr.Row():

        csv\_file\_output = gr.File(label="Download CSV Log")

        json\_file\_output = gr.File(label="Download JSON Log")

        def serve\_csv(): return "sentiment\_log.csv"

        def serve\_json(): return "sentiment\_log.json"

        download\_csv\_btn = gr.Button("📥 Download CSV")

        download\_json\_btn = gr.Button("📥 Download JSON")

        download\_csv\_btn.click(fn=serve\_csv, outputs=csv\_file\_output)

        download\_json\_btn.click(fn=serve\_json, outputs=json\_file\_output)

# Launch the App (Only 2 tabs now)

app = gr.TabbedInterface(

    interface\_list=[chat\_tab, dashboard\_tab],

    tab\_names=["💬 Chat Assistant", "📊 Dashboard"]

)

app.launch()

**GitHub & Project Demo Link**

<https://github.com/vennelasonga/citizenai.git>

<https://drive.google.com/file/d/1GnMoSBgnoeujwP5E_nq6Q1Ai2WY_o8oB/view?usp=drive_link>