Citizen AI – Intelligent Citizen Engagement Platform

Generative AI with IBM Granite-3.3-2B-Instruct

1. Introduction

• Project Title: Citizen AI- Intelligent Citizen Engagement Platform

• Team ID: LTVIP2025TMID29171

• Team Members:

- Member 1: Korupolu Sujana (Team Lead)

  - Guided the team members to work on the project by assigning tasks to each member.

  - Worked on model integration, prompt engineering, and sentiment analysis.

  - Designed and implemented the user interface using Streamlit with tabs, metrics, and dashboard components along with team.

  - Managed environment setup, deployment automation using ngrok, and prepared technical documentation along with the team.

- Member 2: Lalitha Devi Penke

  - Designed and implemented the user interface using Streamlit with tabs, metrics, and dashboard components along with team.

  - Worked on Documentation for this Project along with the team.

- Member 3: Srihari Gopu

- Managed environment setup, deployment automation using ngrok, and prepared technical documentation along with the team.

- Worked on final report for this Project.

2. Project Overview

• Purpose:

The Citizen AI Platform is a generative AI-powered web application designed to assist citizens with government-related queries through a real-time conversational assistant. It utilizes the IBM Granite-3.3-2B-Instruct model to generate intelligent, informative, and context-aware responses to civic questions.

• Features:

- Real-time Conversational Assistant

- Sentiment Analysis

- Topic Extraction

- Dynamic Dashboard Visualization

- WordCloud for topic visualization

- Personalized user context via sidebar input

- Streamlit-based UI

- GPU-accelerated IBM Granite model inference

3. Architecture

• Frontend:

The frontend is built using Streamlit, which provides an interactive and visually appealing user interface. It includes chat UI components, sentiment visualization, topic word clouds, and sidebar configurations.

• Backend:

The backend runs in a Google Colab notebook, where the IBM Granite model is loaded using the Hugging Face Transformers library. It also sets up pyngrok to provide public access to the Streamlit app via tunnel.

4. Setup Instructions

• Prerequisites: Python 3.10+, pip, Google Colab, ngrok account

• Installation:

1. Open the notebook in Google Colab
2. Install the required packages using pip
3. Setup ngrok authentication token
4. Run the Streamlit app/Gradio using the provided deployment script

5. Folder Structure

• Client (Streamlit App):

- citizen_ai_app.py: Main application logic for frontend:

```
# Citizen AI - Intelligent Citizen Engagement Platform

import streamlit as st

import torch
```

```
from transformers import AutoTokenizer, AutoModelForCausalLM
import plotly.express as px
import plotly.graph_objects as go
from plotly.subplots import make_subplots
import pandas as pd
import numpy as np
from textblob import TextBlob
from wordcloud import WordCloud
import matplotlib.pyplot as plt
import seaborn as sns
from datetime import datetime, timedelta
import re
import json
from collections import defaultdict, Counter
import time

# Page configuration
st.set_page_config(
    page_title="Citizen AI - Intelligent Engagement Platform",
    page_icon="🏛",
    layout="wide",
    initial_sidebar_state="expanded"
```

```
)

# Custom CSS for better styling
st.markdown("""
<style>
    .main-header {
        background: linear-gradient(90deg, #1e3c72 0%, #2a5298 100%);
        padding: 2rem;
        border-radius: 10px;
        color: white;
        text-align: center;
        margin-bottom: 2rem;
    }
    .metric-card {
        background: #f8f9fa;
        padding: 1rem;
        border-radius: 10px;
        border-left: 4px solid #2a5298;
        margin: 0.5rem 0;
    }
    .chat-message {
        padding: 1rem;
```

```
        margin: 0.5rem 0;
        border-radius: 10px;
        border-left: 4px solid #2a5298;
    }
    .user-message {
        background-color: #e3f2fd;
        border-left-color: #1976d2;
        color: black;
    }
    .bot-message {
        background-color: #f3e5f5;
        border-left-color: #7b1fa2;
        color: black;
    }
    .sentiment-positive { color: #4caf50; font-weight: bold; }
    .sentiment-negative { color: #f44336; font-weight: bold; }
    .sentiment-neutral { color: #ff9800; font-weight: bold; }
</style>
""", unsafe_allow_html=True)

@st.cache_resource
def load_model():
```

```
    """Load the Granite-3.3-2B-Instruct model and tokenizer"""
    try:
        st.info("🤖 Loading IBM Granite-3.3-2B-Instruct model... This may take 2-3 minutes on first run.")
        model_name = "ibm-granite/granite-3.3-2b-instruct"

        # Load tokenizer
        tokenizer = AutoTokenizer.from_pretrained(model_name, trust_remote_code=True)
        if tokenizer.pad_token is None:
            tokenizer.pad_token = tokenizer.eos_token

        # Load model with optimizations for T4 GPU
        model = AutoModelForCausalLM.from_pretrained(
            model_name,
            torch_dtype=torch.float16,
            device_map="auto",
            trust_remote_code=True,
            low_cpu_mem_usage=True
        )

        st.success("☑ Model loaded successfully!")
        return model, tokenizer
```

```
    except Exception as e:
        st.error(f"❌ Error loading model: {str(e)}")
        return None, None

def analyze_sentiment(text):
    """Analyze sentiment using TextBlob"""
    blob = TextBlob(text)
    polarity = blob.sentiment.polarity

    if polarity > 0.1:
        return "Positive", polarity
    elif polarity < -0.1:
        return "Negative", polarity
    else:
        return "Neutral", polarity

def extract_topics(text):
    """Extract key topics from text using simple keyword extraction"""
    stop_words = {'the', 'a', 'an', 'and', 'or', 'but', 'in', 'on', 'at', 'to', 'for', 'of', 'with',
'by', 'is', 'are', 'was', 'were', 'be', 'been', 'being', 'have', 'has', 'had', 'do', 'does',
'did', 'can', 'could', 'should', 'would', 'will'}
    words = re.findall(r'\\b\\w+\\b', text.lower())
```

```
    topics = [word for word in words if word not in stop_words and len(word)
> 3]

    return Counter(topics).most_common(5)


def generate_response(model, tokenizer, prompt, context=""):

    """Generate response using Granite model"""

    if model is None or tokenizer is None:

        return "❌ Model not loaded. Please refresh the page and wait for model
loading to complete."


    try:

        # Create a system prompt for citizen engagement

        system_prompt = """You are a helpful AI assistant for citizen engagement.
You help citizens with:

- Government services information

- Policy explanations

- Civic processes and procedures

- Community engagement opportunities

- Local government contacts and resources


Please provide helpful, accurate, and constructive responses to citizen
inquiries. Keep responses concise but informative."""
```

```
    full_prompt = f"System: {system_prompt}\\n\\nContext: {context}\\n\\nCitizen: {prompt}\\n\\nAssistant:"

    # Tokenize input
    inputs = tokenizer(full_prompt, return_tensors="pt", truncation=True, max_length=400, padding=True)

    # Move to GPU
    inputs = {k: v.to(model.device) for k, v in inputs.items()}

    # Generate response
    with torch.no_grad():
        outputs = model.generate(
            **inputs,
            max_new_tokens=120,
            temperature=0.7,
            do_sample=True,
            pad_token_id=tokenizer.eos_token_id,
            top_p=0.9,
            repetition_penalty=1.1
        )

    # Decode response
```

```
        response = tokenizer.decode(outputs[0], skip_special_tokens=True)

        # Extract only the assistant's response
        if "Assistant:" in response:
            response = response.split("Assistant:")[-1].strip()
        else:
            # Fallback: take text after the prompt
            response = response[len(full_prompt):].strip()

        return response if response else "I'm here to help with your civic questions. Could you please rephrase your question?"

    except Exception as e:
        return f"⚠ Error generating response: {str(e)}"

def initialize_session_state():
    """Initialize session state variables"""
    if 'chat_history' not in st.session_state:
        st.session_state.chat_history = []
    if 'sentiment_data' not in st.session_state:
        st.session_state.sentiment_data = []
    if 'topic_data' not in st.session_state:
```

```
        st.session_state.topic_data = defaultdict(int)
    if 'user_context' not in st.session_state:
        st.session_state.user_context = {}

def main():
    # Initialize session state
    initialize_session_state()

    # Main header
    st.markdown("""
    <div class="main-header">
        <h1>🏛️ Citizen AI</h1>
        <h3>Intelligent Citizen Engagement Platform</h3>
        <p>Empowering citizens through AI-driven government interaction</p>
    </div>
    """, unsafe_allow_html=True)

    # Load model
    model, tokenizer = load_model()

    if model is None:
```

```
    st.error("❌ Failed to load the AI model. Please refresh the page and try again.")
    st.stop()

# Sidebar for user context and settings
with st.sidebar:
    st.header("🔧 Configuration")

    # User context
    st.subheader("👤 User Profile")
    user_location = st.text_input("📍 Location (City/State)", placeholder="e.g., Boston, MA")
    user_interests = st.multiselect(
        "🎯 Areas of Interest",
        ["Transportation", "Healthcare", "Education", "Environment", "Housing", "Public Safety", "Taxation", "Employment", "Voting", "Permits"]
    )

    # Store user context
    st.session_state.user_context = {
        "location": user_location,
        "interests": user_interests
    }
```

```
        st.divider()


        # Model info
        st.subheader("🤖 AI Model Info")
        st.info("**Model:** IBM Granite-3.3-2B-Instruct\\n**GPU:** T4\\n**Status:** ☑ Active")


        # Clear chat button
        if st.button("🗑 Clear Chat History", type="secondary"):
            st.session_state.chat_history = []
            st.session_state.sentiment_data = []
            st.session_state.topic_data = defaultdict(int)
            st.rerun()


    # Main tabs
    tab1, tab2, tab3 = st.tabs(["💬 AI Assistant", "📊 Sentiment Analysis", "📈 Dashboard"])


    with tab1:
        st.header("🤖 Real-Time Conversational AI Assistant")
        st.write("Ask me anything about government services, policies, voting, permits, or civic processes!")
```

```
    # Display chat history
    for i, (user_msg, bot_msg, timestamp) in
enumerate(st.session_state.chat_history):
        st.markdown(f"""
        <div class="chat-message user-message">
            <strong>You ({timestamp}):</strong><br>{user_msg}
        </div>
        """, unsafe_allow_html=True)

        st.markdown(f"""
        <div class="chat-message bot-message">
            <strong>🤖 Citizen AI:</strong><br>{bot_msg}
        </div>
        """, unsafe_allow_html=True)

    # Input form
    with st.form("chat_form", clear_on_submit=True):
        user_input = st.text_area(
            "💬 Your Question:",
            placeholder="e.g., How do I register to vote? What permits do I need
to start a business? How can I report a pothole?",
            height=100
```

```
        )

        col1, col2 = st.columns([1, 4])
        with col1:
            submit_button = st.form_submit_button("📤 Send", type="primary")

        if submit_button and user_input:
            with st.spinner("🤔 Thinking... Generating response..."):
                # Create context from user profile
                context = f"User location: {user_location if user_location else 'Not specified'}, Interests: {', '.join(user_interests) if user_interests else 'General'}"

                # Generate response
                response = generate_response(model, tokenizer, user_input, context)

                # Analyze sentiment
                sentiment, polarity = analyze_sentiment(user_input)

                # Extract topics
                topics = extract_topics(user_input)

                # Store data
```

```
            timestamp = datetime.now().strftime("%H:%M")
            st.session_state.chat_history.append((user_input, response,
timestamp))
            st.session_state.sentiment_data.append({
              'timestamp': datetime.now(),
              'sentiment': sentiment,
              'polarity': polarity,
              'message': user_input
            })

            # Update topic data
            for topic, count in topics:
              st.session_state.topic_data[topic] += count

            st.rerun()

  with tab2:
    st.header("📊 Citizen Sentiment Analysis")

    if st.session_state.sentiment_data:
      # Create sentiment dataframe
      df_sentiment = pd.DataFrame(st.session_state.sentiment_data)
```

```
col1, col2 = st.columns(2)

with col1:
    # Sentiment distribution
    sentiment_counts = df_sentiment['sentiment'].value_counts()
    fig_pie = px.pie(
        values=sentiment_counts.values,
        names=sentiment_counts.index,
        title="📈 Overall Sentiment Distribution",
        color_discrete_map={
            'Positive': '#4caf50',
            'Negative': '#f44336',
            'Neutral': '#ff9800'
        }
    )
    st.plotly_chart(fig_pie, use_container_width=True)

with col2:
    # Sentiment metrics
    total_messages = len(df_sentiment)
```

```
            positive_pct = len(df_sentiment[df_sentiment['sentiment'] ==
'Positive']) / total_messages * 100

            negative_pct = len(df_sentiment[df_sentiment['sentiment'] ==
'Negative']) / total_messages * 100

            neutral_pct = len(df_sentiment[df_sentiment['sentiment'] ==
'Neutral']) / total_messages * 100


            st.metric("📈 Positive Sentiment", f"{positive_pct:.1f}%")

            st.metric("📉 Negative Sentiment", f"{negative_pct:.1f}%")

            st.metric("📊 Neutral Sentiment", f"{neutral_pct:.1f}%")

            st.metric("💬 Total Messages", total_messages)


        # Recent messages with sentiment

        st.subheader("📋 Recent Messages with Sentiment Analysis")

        for item in st.session_state.sentiment_data[-3:]:  # Show last 3

            sentiment_class = f"sentiment-{item['sentiment'].lower()}"

            st.markdown(f"""

            <div class="metric-card">

                <strong>📝 Message:</strong> {item['message'][:150]}{'...' if
len(item['message']) > 150 else ''}<br>

                <strong>🎭 Sentiment:</strong> <span
class="{sentiment_class}">{item['sentiment']}</span>

                (Score: {item['polarity']:.2f})
```

```
            </div>
            """, unsafe_allow_html=True)
        else:
            st.info("💡 Start chatting with the AI assistant to see sentiment analysis!")

    with tab3:
        st.header("📊 Dynamic Dashboard")

        # Key metrics
        col1, col2, col3, col4 = st.columns(4)

        with col1:
            st.metric("💬 Total Interactions", len(st.session_state.chat_history))

        with col2:
            if st.session_state.sentiment_data:
                positive_pct = len([s for s in st.session_state.sentiment_data if s['sentiment'] == 'Positive']) / len(st.session_state.sentiment_data) * 100
                st.metric("😊 Positive Sentiment", f"{positive_pct:.1f}%")
            else:
                st.metric("😊 Positive Sentiment", "0%")
```

```
        with col3:
            st.metric("🏷️ Active Topics", len(st.session_state.topic_data))

        with col4:
            if st.session_state.user_context['location']:
                st.metric("📍 User Location",
st.session_state.user_context['location'])
            else:
                st.metric("📍 User Location", "Not Set")

        # Topic analysis
        if st.session_state.topic_data:
            col1, col2 = st.columns(2)

            with col1:
                # Top topics bar chart
                topics_df = pd.DataFrame(
                    list(st.session_state.topic_data.items()),
                    columns=['Topic', 'Frequency']
                ).sort_values('Frequency', ascending=False).head(10)
```

```
    fig_bar = px.bar(
        topics_df,
        x='Frequency',
        y='Topic',
        orientation='h',
        title="🔥 Most Discussed Topics",
        color='Frequency',
        color_continuous_scale='viridis'
    )
    fig_bar.update_layout(yaxis={'categoryorder': 'total ascending'})
    st.plotly_chart(fig_bar, use_container_width=True)

with col2:
    # Word cloud
    if len(st.session_state.topic_data) > 0:
        wordcloud = WordCloud(
            width=400,
            height=300,
            background_color='white',
            colormap='viridis',
            max_words=50
        ).generate_from_frequencies(st.session_state.topic_data)
```

```
            fig, ax = plt.subplots(figsize=(8, 6))
            ax.imshow(wordcloud, interpolation='bilinear')
            ax.axis('off')
            ax.set_title('☁️ Topic Word Cloud')
            st.pyplot(fig)
        else:
            st.info("💡 Start asking questions to see topic analysis!")

    # Footer
    st.markdown("---")
    st.markdown("""
    <div style="text-align: center; color: #666;">
        <p>🏛️ <strong>Citizen AI</strong> - Powered by IBM Granite-3.3-2B-Instruct | Built with Streamlit</p>
        <p>Enhancing citizen-government interaction through artificial intelligence</p>
        <p>🚀 Running on Google Colab T4 GPU</p>
    </div>
    """, unsafe_allow_html=True)

if __name__ == "__main__":
```

main()

• Server (Google Colab Notebook):

- Setup cells for package installation and ngrok setup:

```
# CELL 1: Environment Setup and Package Installation

print("🔧 Setting up Citizen AI Platform environment...")

print("⚡ Using T4 GPU runtime")


# Install required packages

!pip install streamlit --quiet

!pip install transformers torch torchvision torchaudio --index-url
https://download.pytorch.org/whl/cu118 --quiet

!pip install accelerate --quiet

!pip install plotly --quiet

!pip install textblob --quiet

!pip install wordcloud --quiet

!pip install matplotlib seaborn --quiet

!pip install pandas numpy --quiet

!pip install pyngrok --quiet


# Download NLTK data for TextBlob

import nltk
```

```
nltk.download('punkt', quiet=True)

nltk.download('brown', quiet=True)


# Verify GPU availability

import torch

print(f"🎯 CUDA Available: {torch.cuda.is_available()}")

if torch.cuda.is_available():

   print(f"🚀 GPU Device: {torch.cuda.get_device_name(0)}")

   print(f"💾 GPU Memory:
{torch.cuda.get_device_properties(0).total_memory / 1e9:.1f} GB")


print("☑ Environment setup complete!")


# CELL 3: Setup ngrok for public access

#!pip install pyngrok

from pyngrok import ngrok

import os

NGROK_TOKEN = “My-Authtoken”


# Set authentication token

try:

   ngrok.set_auth_token(NGROK_TOKEN)
```

```
    print("☑ ngrok authentication token set successfully!")
except Exception as e:
    print(f"❌ Error setting ngrok token: {e}")
    print("🔑 Please make sure you've replaced 'YOUR_NGROK_TOKEN_HERE' with your actual token")
```

- Deployment script to start Streamlit server:

```
#  DEPLOYMENT SCRIPT
import subprocess
import threading
import time
from pyngrok import ngrok
import sys
import os

print("🔄 Stopping previous deployment...")
ngrok.kill()

print("🔧 Fixing Streamlit installation...")
# Ensure Streamlit is properly installed and accessible
try:
    # Reinstall streamlit with explicit path
```

```
    result = subprocess.run([sys.executable, "-m", "pip", "install", "streamlit", "--
upgrade", "--force-reinstall"],
                            capture_output=True, text=True, timeout=120)
    print("☑ Streamlit installation completed")

    # Verify streamlit installation
    result = subprocess.run([sys.executable, "-m", "streamlit", "--version"],
                            capture_output=True, text=True, timeout=30)
    if result.returncode == 0:
        print(f"☑ Streamlit version: {result.stdout.strip()}")
    else:
        print(f"⚠  Streamlit verification: {result.stderr}")

except Exception as e:
    print(f"❌ Error with Streamlit installation: {e}")

print("🚀 Starting Citizen AI Platform...")

# Function to run Streamlit using Python module approach
def run_streamlit():
    """Run Streamlit using python -m streamlit approach"""
    try:
```

```
    # Use python -m streamlit instead of direct streamlit command
    cmd = [
        sys.executable, "-m", "streamlit", "run", "citizen_ai_app.py",
        "--server.port", "8501",
        "--server.headless", "true",
        "--server.fileWatcherType", "none",
        "--browser.gatherUsageStats", "false",
        "--server.enableCORS", "false",
        "--server.enableXsrfProtection", "false"
    ]

    print(f"🔧 Running command: {' '.join(cmd)}")
    process = subprocess.Popen(cmd, stdout=subprocess.PIPE,
stderr=subprocess.PIPE, text=True)

    # Monitor the process
    for line in iter(process.stdout.readline, ''):
        if line.strip():
            print(f"📡 Streamlit: {line.strip()}")
        if "Network URL" in line or "External URL" in line:
            break
```

```
    except Exception as e:
        print(f"❌ Error running Streamlit: {e}")

# Start Streamlit in background thread
print("📡 Starting Streamlit server...")
streamlit_thread = threading.Thread(target=run_streamlit, daemon=True)
streamlit_thread.start()

# Wait longer for Streamlit to fully start
print("⏳ Waiting for Streamlit to initialize...")
time.sleep(25)

# Test if Streamlit is running by checking the port
import socket
def check_port(host, port):
    try:
        sock = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
        sock.settimeout(3)
        result = sock.connect_ex((host, port))
        sock.close()
        return result == 0
    except:
```

```
        return False


# Check if Streamlit is running
if check_port('localhost', 8501):
    print("☑ Streamlit server is responding on port 8501")
else:
    print("❌ Streamlit server not responding. Trying alternative approach...")


    # Alternative: Run streamlit directly with full path
    streamlit_path = subprocess.run([sys.executable, "-c", "import streamlit;
print(streamlit.__file__)"],
                                    capture_output=True, text=True)
    if streamlit_path.returncode == 0:
        streamlit_dir = os.path.dirname(streamlit_path.stdout.strip())
        streamlit_script = os.path.join(streamlit_dir, "web", "cli.py")


        # Try running streamlit CLI directly
        def run_streamlit_alt():
            try:
                subprocess.run([sys.executable, streamlit_script, "run",
"citizen_ai_app.py",
                                "--server.port", "8501", "--server.headless", "true"])
            except Exception as e:
```

```
            print(f"Alternative streamlit failed: {e}")

    alt_thread = threading.Thread(target=run_streamlit_alt, daemon=True)
    alt_thread.start()
    time.sleep(15)

# Create ngrok tunnel
try:
  print("🌐 Creating public tunnel...")

  # Create tunnel with retry logic
  max_retries = 3
  tunnel = None

  for attempt in range(max_retries):
    try:
      tunnel = ngrok.connect(8501, proto="http", bind_tls=True)
      break
    except Exception as e:
      print(f"❌ Tunnel attempt {attempt + 1} failed: {e}")
      if attempt < max_retries - 1:
        time.sleep(5)
```

```
            continue
        else:
            raise e

if tunnel:
    print("\n" + "="*70)
    print("🎉 CITIZEN AI PLATFORM IS DEPLOYED!")
    print("="*70)
    print(f"🌐 Public URL: {tunnel.public_url}")
    print(f"🔗 Click here: {tunnel.public_url}")
    print("="*70)
    print("\n🎯 Application Features:")
    print("☑ Real-Time AI Assistant (IBM Granite-3.3-2B)")
    print("☑ Citizen Sentiment Analysis")
    print("☑ Dynamic Dashboard & Analytics")
    print("☑ Personalized Contextual Responses")
    print("☑ T4 GPU Acceleration")
    print("\n💡 First-Time Setup:")
    print("• The AI model will load when you first visit (2-3 minutes)")
    print("• Set your location and interests in the sidebar")
    print("• Try asking: 'How do I register to vote?'")
```

```
print("\n⚠️ IMPORTANT: Keep this cell running to maintain access!")
print("="*70)

# Keep tunnel alive with better error handling
try:
    while True:
        time.sleep(30)
        # Check if tunnel is still active
        try:
            tunnels = ngrok.get_tunnels()
            if not tunnels:
                print("🔄 Tunnel disconnected, recreating...")
                tunnel = ngrok.connect(8501, proto="http", bind_tls=True)
                print(f"🌐 New URL: {tunnel.public_url}")
            else:
                print(f"🔄 App running... Access: {tunnel.public_url}")
        except Exception as tunnel_check_error:
            print(f"⚠️ Tunnel check error: {tunnel_check_error}")

except KeyboardInterrupt:
    print("\n🛑 Shutting down Citizen AI Platform...")
```

```
        ngrok.kill()


except Exception as e:
   print(f"❌ Error creating tunnel: {e}")
   print("\n🔧 Troubleshooting Steps:")
   print("1. Make sure your ngrok token is set correctly")
   print("2. Try restarting the runtime: Runtime → Restart and run all")
   print("3. Check if all packages installed correctly")


   # Show alternative local access
   print(f"\n💡 Alternative: If you're running locally, try:
http://localhost:8501")
```

6. Running the Application

Commands:

```
from pyngrok import ngrok
ngrok.set_auth_token("MY_NGROK_TOKEN")
tunnel = ngrok.connect(8501, bind_tls=True)
```

Streamlit Server:

```
!streamlit run citizen_ai_app.py --server.port 8501 --server.headless true
```

If Gradio framework is used then it automatically loads the link.

7. API Documentation

This Streamlit app is interactive and does not expose traditional REST API endpoints. However, key internal APIs:

Internal Methods:

- generate_response(prompt, context) — Returns response using Granite model
- analyze_sentiment(text) — Uses TextBlob to return polarity and sentiment class
- extract_topics(text) — Extracts common keywords from input

8. Authentication

Currently, there is no user-level authentication. Planned enhancements include:

- Session token-based login
- Role-based access for administrators
- OAuth integration with Google/Microsoft

9. User Interface

Components:

- Tabs: "AI Assistant", "Sentiment Analysis", "Dashboard"
- Forms: Chat input, user profile configuration
- Charts: Pie (Sentiment), Bar (Topics)
- Word Cloud: From keyword extraction

10. Testing

Strategy:

- Manual UI testing via Google Colab Public URL
- Chat response validation
- GPU model loading confirmation
- Session state validation

Tools:

- Streamlit debug logs
- Google Colab cell output

11. Screenshots or Demo

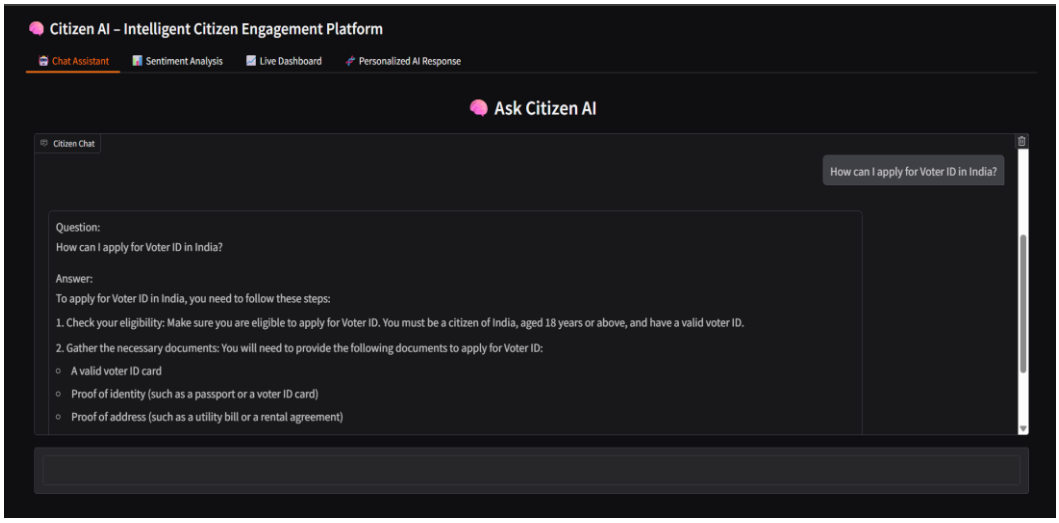
Demo Link:

Public ngrok URL shown during runtime:

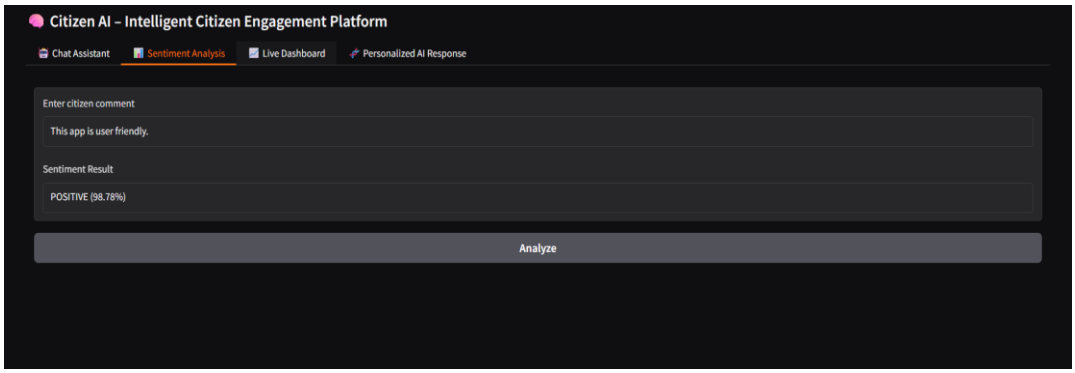
https://3805-34-125-155-0.ngrok-free.app/

Screenshots:

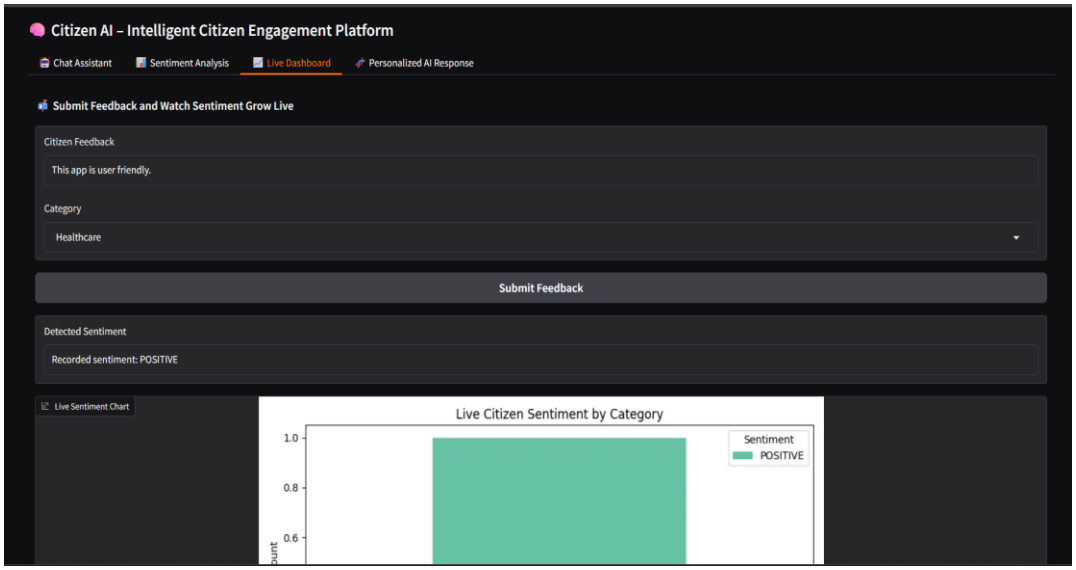
1. Chat UI with user and AI messages



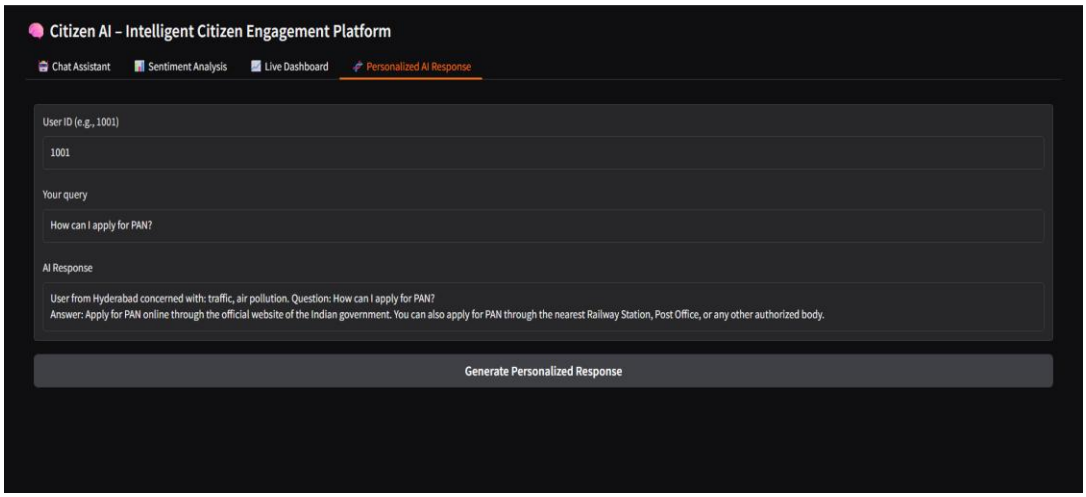
2. Sentiment Analysis



3. Live Dashboard



4. Personalized AI Response



12. Known Issues

- No persistence: Data resets after runtime ends
- Initial model loading takes ~2-3 minutes
- No authentication support
- Limited to session-based state

13. Future Enhancements

- Add user registration & authentication
- Persistent database storage for chats
- Admin dashboard for trend analysis
- Support for multilingual queries
- Deployment via Hugging Face Spaces / Docker
- Voice input integration for accessibility