PROJECT TITLE

CITIZEN AI PROJECT DOCUMENTATION

1.Introduction

- **Project title : Citizen Ai**
- **Team member :**K. Priyadharshini
- **Team member :**K.S. Sangeetha
- **Team member :**S. Thamizharasi
- **Team member :**T. Teresa Helen Jyothi

2.project overview

- ***Purpose :***

  The purpose of Citizen AI is to empower individuals with a digital assistant that improves civic engagement, personal well-being, and access to public services. It assists people in navigating government services, staying informed about policies and rights, participating in community discussions, and receiving personalized recommendations for their well-being, education, and career. For governments, it serves as a tool to understand public sentiment, improve communication, and co-create solutions with citizens, thereby bridging the gap between civic engagement, digital inclusion, and community resilience.

- **Features:**

Conversational Interface

Key Point: Human-like interaction

Functionality: Citizens can ask questions about services, policies, or community programs in natural language and get actionable responses.

Service Navigator

Key Point: Easy access to public services

Functionality: Guides users step by step through applications (healthcare, IDs, permits, benefits).

Policy & Rights Simplifier

Key Point: Plain-language explanations

Functionality: Converts legal and government documents into easy-to-understand summaries.

Citizen Feedback & Polling

Key Point: Civic voice amplification

Functionality: Collects public opinions, conducts micro-surveys, and channels aggregated insights to policymakers.

Community Resource Finder

Key Point: Localized guidance

Functionality: Recommends nearby services like clinics, schools, shelters, or training centers.

Well-being & Education Coach

Key Point: Personal growth support

Functionality: Provides personalized tips on health, financial literacy, and learning resources.

Anomaly & Trend Detection

Key Point: Data-driven monitoring

Functionality: Identifies emerging community issues (e.g., rising unemployment or health complaints).

Multimodal Input Support

Key Point: Flexible interactions

Functionality: Accepts text, voice, PDFs, and images for service navigation or policy explanation.

Streamlit or Gradio UI

Key Point: Accessible design

Functionality: Offers a simple dashboard for citizens with tabs for chat, resources, polls, and reports.

3. Architecture

Frontend (Streamlit/Gradio):

Interactive web interface with navigation sidebar, chat window, service guides, survey forms, and local resource maps.

Backend (FastAPI):

Serves as the core REST API, handling chat interactions, service queries, feedback collection, and report generation.

LLM Integration (IBM Watsonx Granite or OpenAI GPT):

Natural language processing for summaries, translations, and dialogue. Prompts are tuned for clarity, inclusivity, and neutrality.

Vector Search (Pinecone):

Stores embedded government documents, FAQs, and service guides. Enables semantic search for policy-related queries.

ML Modules:

Forecasting: Tracks trends in citizen engagement or service demand.

Anomaly Detection: Flags unusual patterns in feedback or usage data.

4. Setup Instructions

Prerequisites:

Python 3.9+

pip and venv

API keys (IBM Watsonx / OpenAI, Pinecone, optional geolocation services)

Internet access

Installation Process:

1. Clone the repository

2. Install dependencies from requirements.txt

3. Configure .env with credentials

4. Run FastAPI backend

5. Launch Streamlit/Gradio frontend

6. Interact with chat, polls, and service navigation modules

5. Folder Structure

```
app/            # FastAPI backend
 └── api/         # Chat, feedback, services, resources
ui/             # Frontend pages
```

```
citizen_dashboard.py  # Streamlit/Gradio main entry
policy_helper.py      # Simplifies government docs
resource_locator.py   # Finds local services
trend_forecaster.py   # Forecasts community issues
anomaly_checker.py    # Detects irregular data patterns
feedback_analyzer.py  # Processes citizen input
```

6. Running the Application

- Start FastAPI server for backend endpoints
- Launch Streamlit/Gradio dashboard for UI
- Use sidebar navigation to:
- Chat with Citizen AI
- Access service navigator
- View summaries of policies
- Participate in polls/feedback
- Download reports

7. API Documentation

Key Endpoints:

- POST /chat/ask → Responds to citizen queries

- GET /get-services → Lists relevant government/public services

- POST /upload-doc → Summarizes or indexes a policy document

- GET /search-docs → Semantic search of policies

- POST /submit-feedback → Captures citizen feedback/poll responses

- GET /community-trends → Provides forecast insights

8. Authentication

- Demo Mode: Open access
- Secure Deployment:
- JWT tokens or API keys
- OAuth2 for government/NGO logins
- Role-based access (citizen, admin, researcher)
- Future: Personal accounts with history tracking

9. User Interface

- Sidebar navigation (chat, services, resources, feedback)
- Card-based service highlights (health, education, ID services)
- Real-time forms for polls and surveys
- Downloadable summaries/reports
- Accessibility-first design (large fonts, multilingual support, voice input)

10. Testing

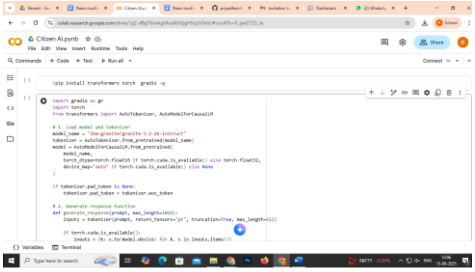
- Unit Testing: NLP summarization, utility functions

- API Testing: Swagger UI, Postman

- Manual Testing: Service navigation, chat quality, feedback forms

- Edge Cases: Complex legal docs, invalid inputs, large uploads

11.screen shots


Citizen AI.pynb
File Edit View Insert Runtime Tools Help
Commands + Code + Text ▷ Run all
Connect
Variables Terminal
Type here to search

```
!pip install transformers torch  gradio -q
```

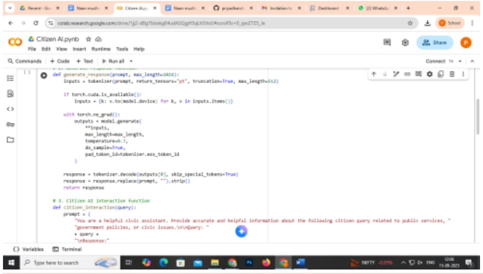
```
import gradio as gr
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM

# 1. 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

# 2. Generate response function
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()}
```



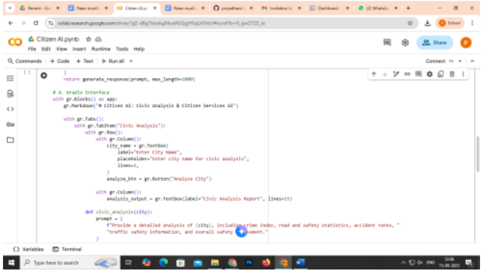
```
def generate_response(prompt, max_length=1024):
    inputs = tokenizer(prompt, return_tensors="pt", truncation=True, max_length=512)

    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

# 3. Citizen AI interaction function
def citizen_interaction(query):
    prompt = (
        "You are a helpful civic assistant. Provide accurate and helpful information about the following citizen query related to public services, "
        "government policies, or civic issues.\n\nQuery: "
        + query +
        "\nResponse:"
```



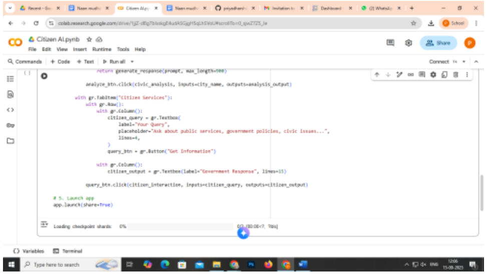
```
    )
    return generate_response(prompt, max_length=1000)

# 4. Gradio Interface
with gr.Blocks() as app:
    gr.Markdown("# Citizen AI: Civic Analysis & Citizen Services AI")

    with gr.Tabs():
        with gr.TabItem("Civic Analysis"):
            with gr.Row():
                with gr.Column():
                    city_name = gr.Textbox(
                        label="Enter City Name",
                        placeholder="Enter city name for civic analysis",
                        lines=1,
                    )
                    analyze_btn = gr.Button("Analyze City")

                with gr.Column():
                    analysis_output = gr.Textbox(label="Civic Analysis Report", lines=15)

            def civic_analysis(city):
                prompt = (
                    f"Provide a detailed analysis of {city}, including crime index, road and safety statistics, accident rates, "
                    "traffic safety information, and overall safety assessment."
                )
```



```
                return generate_response(prompt, max_length=1000)

            analyze_btn.click(civic_analysis, inputs=city_name, outputs=analysis_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)

# 5. Launch app
app.launch(share=True)
```

```
Loading checkpoint shards:   0%|          | 0/2 [00:00<?, ?it/s]
```

12.Known Issues

- Limited offline capability

- Some complex legal texts may lose nuance in summarization

- Local service locator dependent on available geodata

13. Future Enhancements

- Voice-based assistant (speech-to-text + TTS)

- Mobile app integration
- Multi-language support for all major regional languages
- AI-driven community dashboards for policymakers
- Integration with smart city IoT data for real-time citizen alerts