Citizen AI with IBM

Project Documentation

1.Introduction

• Project Title: Citizen AI with IBM

- Team member: SARANYA.K
- Team member: SARANYA.S
- Team member: SARIBA.R
- Team member: SHAJITHA BANU.H

Citizen AI represents the approach of designing artificial intelligence that behaves like a responsible digital citizen. It prioritizes transparency, accountability, fairness, inclusivity, and sustainability. IBM has taken a leadership role in this space, embedding ethical practices in its AI platforms such as IBM Watson and Watsonx. This documentation explores Citizen AI with IBM, structured like a technical project, with purpose, architecture, APIs, and future enhancement.

1. Project Overview

• Purpose: The purpose of Citizen AI with IBM is to create AI systems that align with societal values, acting responsibly in decision-making processes. IBM’s platforms provide tools for explainability, bias detection, and responsible deployment, ensuring AI augments human intelligence rather than replacing it.

• Features: Conversational Interface - Key Point: Natural language interaction - Functionality: Enables citizens and institutions to ask questions and receive understandable guidance.

Policy Summarization –

Key Point: Simplified governance understanding - Functionality: Converts complex regulations into concise and actionable insights.

Resource Forecasting –

Key Point: Predictive analytics Functionality: Projects usage trends in domains like healthcare, education, and energy.

Ethical Audit Generator –

Key Point: AI accountability - Functionality: Provides reports on fairness, transparency, and compliance.

Citizen Feedback Loop –

Key Point: Inclusive engagement - Functionality: Collects and analyzes community input to improve AI systems.

KPI Forecasting –

Key Point: Strategic planning support - Functionality: Assists policymakers and businesses in tracking progress.

Bias & Anomaly Detection –

Key Point: Ethical safeguards - Functionality: Identifies irregularities or biases in datasets and AI outputs.

Multimodal Input Support –

Key Point: Flexible interaction - Functionality: Supports text, audio, and document uploads for AI-driven analysis.

IBM Watsonx Integration –

Key Point: Human-AI collaboration - Functionality: Uses IBM Watsonx Granite LLMs for ethical and explainable AI outcomes.

2. Architecture

The architecture of Citizen AI with IBM integrates multiple layers for responsible deployment:

Frontend: - Built with frameworks like Streamlit/Gradio, providing dashboards for monitoring AI fairness, transparency, and outputs.

Backend: - IBM Watsonx APIs and FastAPI serve as the backend, managing requests, ethical audits, and policy summaries.

LLM Integration (IBM Watsonx Granite): - Advanced language models trained to ensure inclusivity, bias checks, and explainable responses.

Vector Search (Pinecone/Watson Discovery): - Supports semantic search across laws, ethical guidelines, and datasets to guide responsible AI.

AI Governance Layer: - Integrates IBM AI Fairness 360 toolkit for bias detection. - Includes Explainable AI modules for transparency.

ML Modules: Forecasting societal trends (e.g., healthcare demand, education growth). - Anomaly detection to identify misuse of AI systems.

3. Setup Instructions

Prerequisites: -

- Python 3.9+
- IBM Watsonx API keys
- IBM Cloud account
- Internet connectivity

Installation Process:

1. Clone repository containing Citizen AI framework.
2. Install dependencies (requirements.txt).
3. Configure .env file with IBM Watsonx credentials.
4. Launch backend (FastAPI) for AI services.
5. Start frontend (Streamlit/Gradio) for dashboard interaction.
6. Upload datasets or documents for AI governance tasks.

4. Folder Structure

■ api/ – API endpoints for chat, fairness reports, audits

■ models/ – AI governance models, Watsonx integrations

■ ui/ – Streamlit/Gradio pages and dashboards

■ utils/ – Helper functions for preprocessing, embeddings

■ docs/ – Documentation, compliance reports

■ run_dashboard.py – Launch file for main UI

■ governance_layer.py – Bias detection and transparency modules

■ audit_reporter.py – Generates AI accountability reports

5. Running the Application

1. Start FastAPI server to enable backend services.
2. Run Streamlit dashboard for visualization.
3. Access sidebar for navigating to modules: -
    - Fairness & Bias Reports Policy Summarization
    - Resource Forecasting
    - Ethical Audit Logs
4. Interact with Watsonx LLM for AI-powered insights.
5. Generate PDF/HTML reports for compliance tracking.

6. API Documentation

Available APIs: -

POST /chat/ask → Query Watsonx AI for insights.

POST /upload-doc → Upload datasets for ethical auditing.

GET /fairness-report → Return bias analysis results.

GET /summarize-policy → Simplified summaries of regulations. POST /submit-feedback → Store citizen/community feedback. Each API is documented in Swagger UI for testing and integration.

7. Authentication

Citizen AI with IBM ensures secure access via: -

Token-based authentication (JWT). OAuth2 with IBM Cloud credentials.

Role-based access (admin, policymaker, researcher, citizen). Future enhancements include blockchain-based audit trails for AI accountability.

8. User Interface

The interface emphasizes accessibility and transparency: - Sidebar navigation for modules.

Fairness and KPI visualizations.

Chat interface for natural queries.

Tabs for policy summaries, forecasts, and ethical reports.

Report download capability.

The UI is designed for inclusivity, ensuring citizens and non-technical users can engage with AI responsibly.

9. Testing

Testing phases: -

Unit Testing → AI fairness modules, transparency checks.

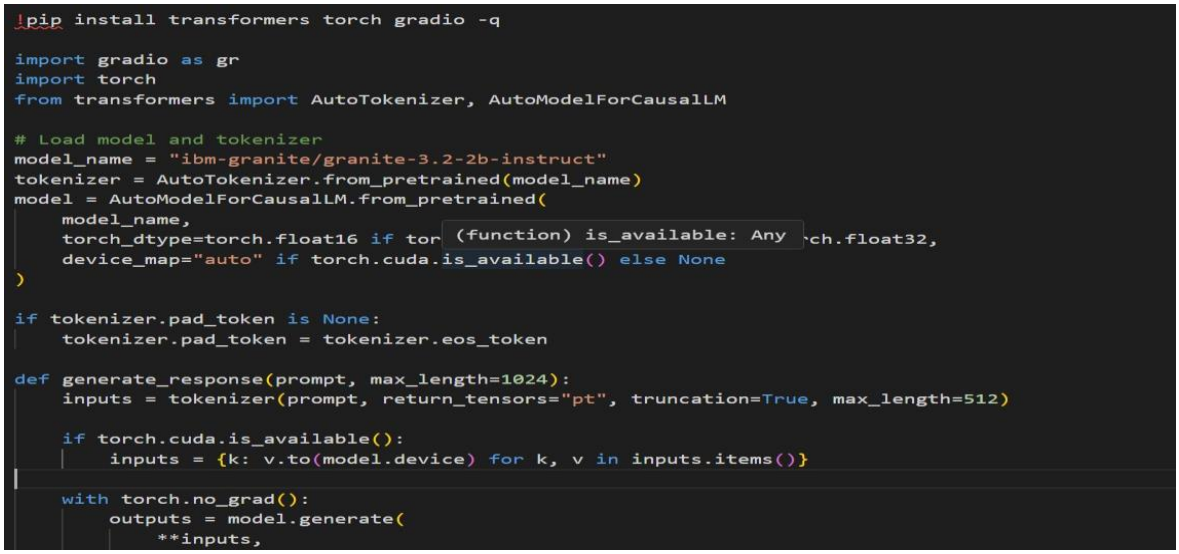
API Testing → Swagger, Postman validation. –

Manual Testing → File uploads, fairness dashboard results.

Stress Testing → Large datasets and policy documents.

Edge case handling includes invalid inputs, incomplete documents, and adversarial data.

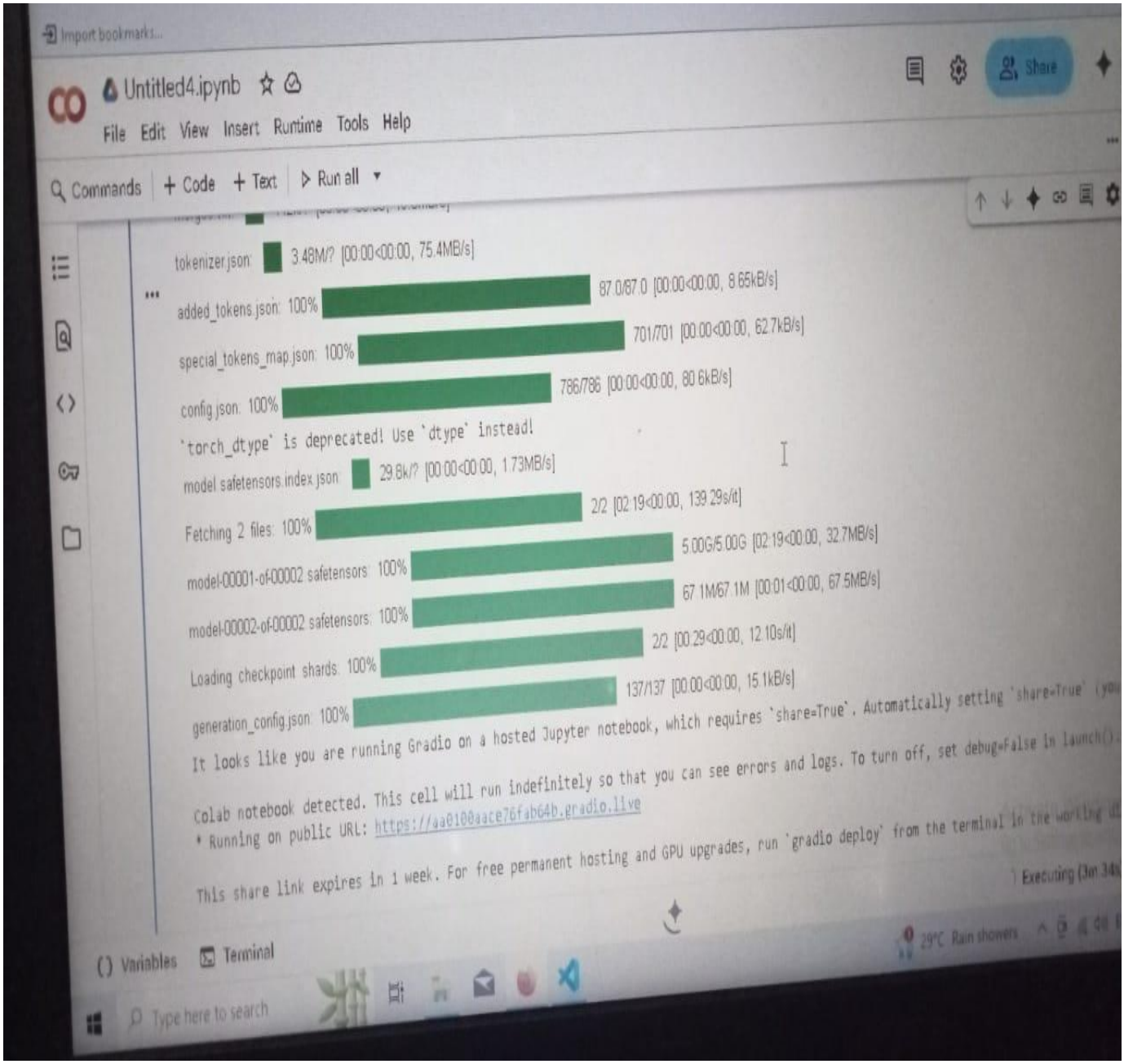
10. Screenshots



```
!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 tor (function) is_available: Any rch.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 statistics\n2. A
    return generate_response(prompt, max_length=1000)

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

```
                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=15)

        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)
```


Import bookmarks...
Untitled4.ipynb
File Edit View Insert Runtime Tools Help
Share
Commands + Code + Text Run all
tokenizer.json: 3.48M/? [00:00<00:00, 75.4MB/s]
added_tokens.json: 100% 87.0/87.0 [00:00<00:00, 8.65kB/s]
special_tokens_map.json: 100% 701/701 [00:00<00:00, 62.7kB/s]
config.json: 100% 786/786 [00:00<00:00, 80.6kB/s]
`torch_dtype` is deprecated! Use `dtype` instead!
model.safetensors.index.json: 29.8k/? [00:00<00:00, 1.73MB/s]
Fetching 2 files: 100% 2/2 [02:19<00:00, 139.29s/it]
model-00001-of-00002.safetensors: 100% 5.00G/5.00G [02:19<00:00, 32.7MB/s]
model-00002-of-00002.safetensors: 100% 67.1M/67.1M [00:01<00:00, 67.5MB/s]
Loading checkpoint shards: 100% 2/2 [00:29<00:00, 12.10s/it]
generation_config.json: 100% 137/137 [00:00<00:00, 15.1kB/s]
It looks like you are running Gradio on a hosted Jupyter notebook, which requires `share=True`. Automatically setting `share=True` (yo
Colab notebook detected. This cell will run indefinitely so that you can see errors and logs. To turn off, set debug=False in launch().
* Running on public URL: https://aa0100aace76fab64b.gradio.live
This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the working d
Executing (3m 34s
Variables Terminal
29°C Rain showers
Type here to search

12.Known Issues

- High dependency on quality of training data.
- Explainability trade-offs in large-scale LLMs.
- Limited availability of domain-specific ethical datasets.
- Scalability challenges for real-time auditing with massive datasets.

11. Future Enhancement

- Integration with blockchain for immutable audit logs.
- Expanded dataset diversity for bias reduction.
- More advanced visualization tools for citizen engagement.
- Global collaboration to standardize Citizen AI frameworks. Integration of quantum-safe AI governance with IBM Quantum research.