

Project Documentation

1.INTRODUCTION

Project Title: Intelligent Citizen Engagement Platform

Team Leader:

Nandhini R

Team Members:

Nanthini M

Noorul Irfana T

Pavithra M

2.PROJECT OVERVIEW

Purpose :

The purpose of the Intelligent Citizen Engagement Platform is to strengthen communication between citizens and government using AI-powered services. The platform provides real-time interaction, policy summarization, feedback collection, and safety analysis, making governance more transparent, inclusive, and efficient. It helps citizens access services and officials make data-driven decisions for better urban management.

Features:

Conversational Interface

- Key Point: Natural language interaction

- **Functionality:** Enables citizens and officials to ask questions, receive updates, and get guidance in plain language through an AI-powered chat assistant.

Policy Summarization

- **Key Point:** Simplified policy understanding
- **Functionality:** Converts lengthy government documents and policies into concise, actionable summaries for easy understanding by citizens and administrators.

Resource Forecasting

- **Key Point:** Predictive analytics
- **Functionality:** Estimates future demand for civic services such as water, energy, and waste management using historical and real-time data.

Eco-Tip Generator

- **Key Point:** Personalized sustainability advice
- **Functionality:** Provides daily eco-friendly suggestions to citizens, promoting sustainable lifestyle practices and reducing environmental impact.

Citizen Feedback Loop

- Key Point: Community engagement
- Functionality: Collects citizen input, analyzes feedback, and informs officials to improve city services, governance, and planning.

KPI Forecasting

- Key Point: Strategic planning support
- Functionality: Projects key performance indicators (KPIs) to help government officials track progress, measure outcomes, and plan ahead effectively.

Anomaly Detection

- Key Point: Early warning system
- Functionality: Identifies unusual or suspicious patterns in data such as traffic, safety, or service usage to flag potential issues for quick response.

Multimodal Input Support

- Key Point: Flexible data handling
- Functionality: Accepts inputs in multiple formats (text, PDFs, CSVs) for document analysis, forecasting, and decision support.

Streamlit or Gradio UI

- Key Point: User-friendly interface
- Functionality: Provides an interactive and intuitive dashboard for both citizens and officials to access insights, analysis, and services seamlessly.

3. Architecture

Frontend (Gradio/Streamlit): Provides a clean UI with chat, dashboards, and analysis tabs.

Backend (FastAPI): Powers API requests for document processing, feedback, and AI responses.

LLM Integration (IBM Granite): Handles natural language understanding and summarization.

Vector Search (Pinecone/FAISS): Enables semantic search on uploaded policies.

ML Modules:

- Forecasting: Predicts trends in services & KPIs
- Anomaly Detection: Detects unusual usage or issues

4. Setup Instructions

Prerequisites:

- Python 3.9+

- pip & virtual environment tools

- Hugging Face Transformers
- API keys for Watsonx Granite / Vector DB
- Internet connection

Installation Process:

- Clone the repository
- Install dependencies from requirements.txt
- Create a .env file and configure credentials
- Run the backend server using FastAPI
- Launch the frontend via Gradio or Streamlit
- Upload data and interact with the modules

5. Folder Structure

app/ – Contains all FastAPI backend logic including routers, models, and integration modules.

app/api/ – Subdirectory for modular API routes like chat, feedback, report, and document vectorization.

ui/ – Contains frontend components for Gradio/Streamlit pages, dashboards, and forms.

citizen_dashboard.py – Entry script for launching the main dashboard (Gradio/Streamlit).

llm_integration.py – Handles all communication with IBM Watsonx Granite model including summarization and chat.

document_embedder.py – Converts documents into embeddings and stores in Pinecone/FAISS.

forecasting.py – Forecasts future service and KPI trends using regression models.

anomaly_checker.py – Flags unusual patterns or anomalies in uploaded datasets.

report_generator.py – Constructs AI-generated governance and citizen engagement reports.

6. Running the Application

To start the project:

- Launch the FastAPI server to expose backend endpoints.
- Run the Gradio or Streamlit dashboard to access the web interface.
- Navigate through pages via the tabs or sidebar.
- Upload documents or CSVs, interact with the chat assistant, and view outputs such as reports, summaries, forecasts, and predictions.
- All interactions are real-time and use backend APIs to dynamically update the frontend.

Frontend (Gradio/Streamlit):

The frontend is built with Gradio or Streamlit, offering an interactive web UI with multiple modules including dashboards, file uploads, chat interface, feedback forms, and report viewers. Navigation is handled through tabs or a sidebar. Each page is modularized for scalability and user-friendliness.

Backend (FastAPI):

FastAPI serves as the backend REST framework powering API endpoints for document processing, citizen queries, feedback handling, forecasting, anomaly detection, and report creation. It is optimized for asynchronous performance and comes with easy Swagger integration for testing APIs.

7. API Documentation

Backend APIs available include:

POST /chat/ask – Accepts a citizen query and responds with an AI-generated message.

POST /upload-doc – Uploads and embeds government documents into the vector database.

GET /search-docs – Returns semantically similar policies or documents based on a natural language query.

GET /get-insights – Provides governance or service-related insights (safety, resources, sustainability).

POST /submit-feedback – Stores citizen feedback for later review and analytics.

Each endpoint is tested and documented in Swagger UI for quick inspection and trial during development.

8. Authentication

This version of the project runs in an open environment for demonstration purposes.

However, secure deployments can integrate:

Token-based authentication (JWT or API keys)

OAuth2 with IBM Cloud or Government credentials

Role-based access (Admin, Citizen, Analyst)

Planned enhancements include user sessions and query history tracking.

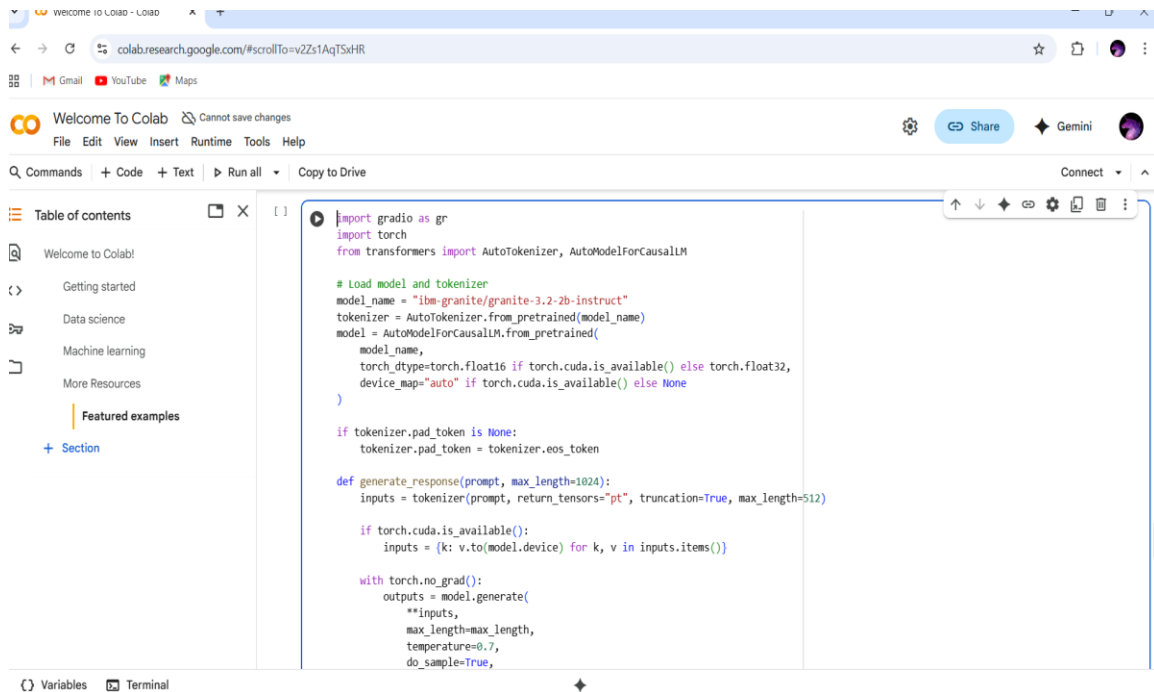
9. User Interface

- Tabs for City Analysis & Citizen Services
- Sidebar navigation
- Text inputs for queries & city names
- Real-time AI-generated responses
- Dashboard with visual insights
- PDF/Report download option

10. Testing

- Unit Testing: Prompt generation & AI functions
- API Testing: Swagger UI & Postman
- Manual Testing: File uploads, city analysis, chat
- Edge Cases: Invalid queries, large inputs

11. Screenshots



This screenshot shows the first code cell in a Google Colab notebook. The code imports the necessary libraries and loads the IBM Granite model. The interface includes a sidebar with a table of contents, a top navigation bar with 'Welcome To Colab' and 'Cannot save changes', and a bottom status bar with 'Variables' and 'Terminal' tabs.

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

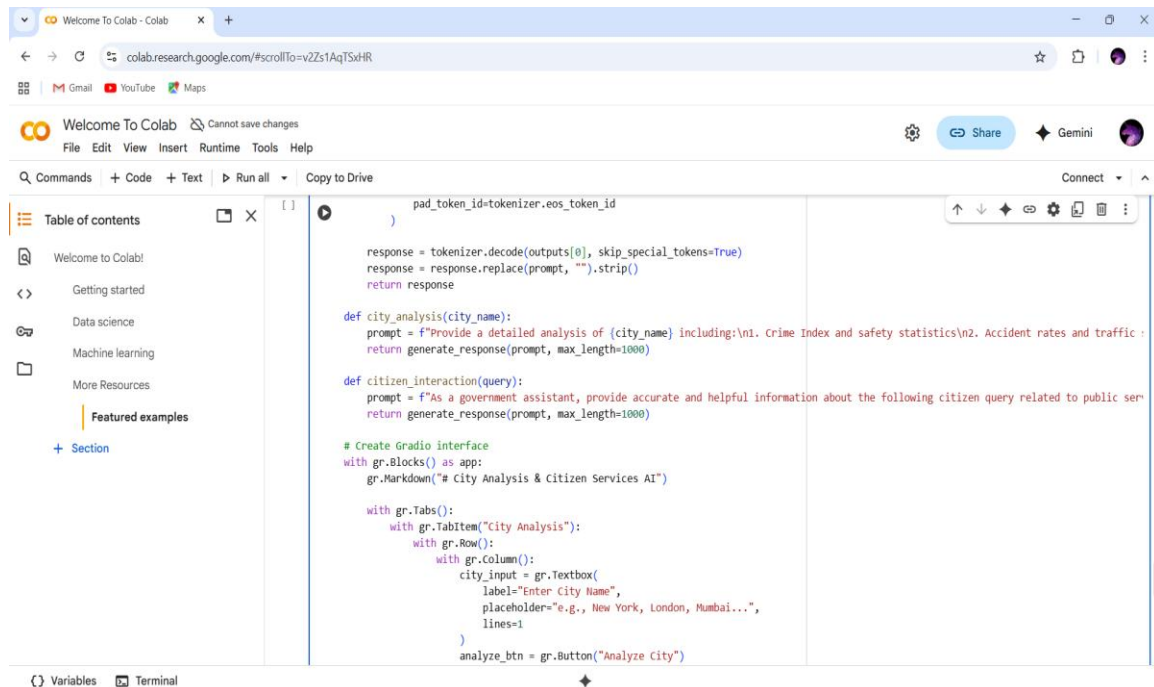
# Load model and tokenizer
model_name = "ibm-granite/granite-3.2-2b-instruct"
tokenizer = AutoTokenizer.from_pretrained(model_name)
model = AutoModelForCausalLM.from_pretrained(
    model_name,
    torch_dtype=torch.float16 if torch.cuda.is_available() else torch.float32,
    device_map="auto" if torch.cuda.is_available() else None
)

if tokenizer.pad_token is None:
    tokenizer.pad_token = tokenizer.eos_token

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,
```



This screenshot shows the continuation of the code in the second cell of the notebook. It defines functions for city analysis and citizen interaction, and then creates a Gradio interface with a text input and an 'Analyze City' button. The interface also includes the same sidebar and top navigation bar as the first screenshot.

```
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. Accident rates and traffic :
    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 query related to public ser
    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(
                        label="Enter City Name",
                        placeholder="e.g., New York, London, Mumbai...",
                        lines=1
                    )
                    analyze_btn = gr.Button("Analyze City")
```


Welcome To Colab - Colab

colab.research.google.com/#scrollTo=v2Zs1AqTSxHR

GmailYouTubeMaps

Welcome To Colab

Cannot save changes

FileEditViewInsertRuntimeToolsHelp

CommandsCodeTextRun allCopy to Drive

Table of contents

Welcome to Colab!

Getting started

Data science

Machine learning

More Resources

Featured examples

+ Section

model-00001-of-00002.safetensors: 100%

5.00G/5.00G [02:18<00:00, 68.0MB/s]

Loading checkpoint shards: 100%

2/2 [00:39<00:00, 16.49s/it]

generation_config.json: 100%

137/137 [00:00<00:00, 7.69kB/s]

Colab notebook detected. To show errors in colab notebook, set debug=True in launch()

* Running on public URL: <https://199b3f48cd41841631.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 direc

City Analysis & Citizen Services AI

City AnalysisCitizen Services

Enter City Name

e.g., New York, London, Mumbai...

Analyze City

City Analysis (Crime Index & Accidents)

VariablesTerminal

8:25 PMPython 3

Welcome To Colab - Colab

colab.research.google.com/#scrollTo=v2Zs1AqTSxHR

GmailYouTubeMaps

Welcome To Colab

Cannot save changes

FileEditViewInsertRuntimeToolsHelp

CommandsCodeTextRun allCopy to Drive

Table of contents

Welcome to Colab!

Getting started

Data science

Machine learning

More Resources

Featured examples

+ Section

This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from

City AnalysisCitizen Services

Enter City Name

e.g., New York, London, Mumbai...

Analyze City

City Analysis (Crime Index & Accidents)

Use via API · Built with Gradio · Settings

VariablesTerminal

8:25 PMPython 3

12. Known Issues

Limited support for very large documents and datasets

Requires stable internet connection for AI integration and real-time processing

High dependency on external APIs and cloud services

UI performance may slow down with heavy concurrent usage

13. Future Enhancements

Intelligent Resource Management: Smarter allocation of water, energy, and waste services.

Environmental Monitoring and Protection: Integration of IoT sensors for pollution, traffic, and climate tracking.

Data-Driven Urban Planning: AI-powered insights to support city development and infrastructure decisions.

Citizen Engagement and Governance: Multi-language support, wider participation tools, and transparent decision-making.

Smart Infrastructure Maintenance: Predictive maintenance for roads, utilities, and public facilities using AI and anomaly detection.