

Sustainable Smart City Assistant Project Documentation

1. Introduction

- Project Title: Sustainable Smart City Assistant Using IBM Granite LLM
- Team Members
 - Deepika T
 - Haripriya M
 - Hasna hasjida A

2. Project Overview

• Purpose:

The Sustainable Smart City Assistant is an AI-powered platform designed to empower cities and residents with eco-conscious tools and data-driven decision-making. It leverages IBM Watsonx Granite LLM and modern data pipelines to optimize energy, water, and waste usage while providing simplified policy summaries, citizen feedback loops, eco-advice, KPI forecasting, and anomaly detection. The platform bridges technology, governance, and citizen engagement to foster greener, inclusive, and resilient urban environments.

• Features:

Conversational Chat Assistant

Key Point: Natural language interaction

Functionality: Citizens and officials ask sustainability questions and receive AI-powered guidance.

Policy Summarization

Key Point: Simplified understanding

Functionality: Summarizes complex city policy documents into concise, actionable insights.

Resource Forecasting

Key Point: Predictive analytics

Functionality: Forecasts water, energy, and waste usage based on past data.

Eco-Tip Generator

Key Point: Sustainable lifestyle advice

Functionality: Recommends daily eco-friendly actions based on user input.

Citizen Feedback Reporting

Key Point: Real-time issue reporting

Functionality: Enables residents to log city issues instantly for government review.

KPI Forecasting & Anomaly Detection

Key Point: Strategic planning & early warnings

Functionality: Forecasts key indicators and detects irregularities in urban data.

Multimodal Input Support

Key Point: Flexible data handling

Functionality: Accepts text, PDFs, and CSVs for summarization, forecasting, and anomaly detection.

Streamlit Dashboard

Key Point: User-friendly UI

Functionality: Provides interactive dashboards for data visualization, reports, and eco insights.

Use Case Scenarios

Policy Search & Summarization: A municipal planner uploads a complex city policy document, and the assistant generates a simplified summary.

Citizen Feedback Reporting: A resident submits an issue such as a burst pipe, and the assistant logs it with category tagging for officials.

KPI Forecasting: A city administrator uploads last year's water usage data and receives AI-powered consumption forecasts for planning.

3. Architecture

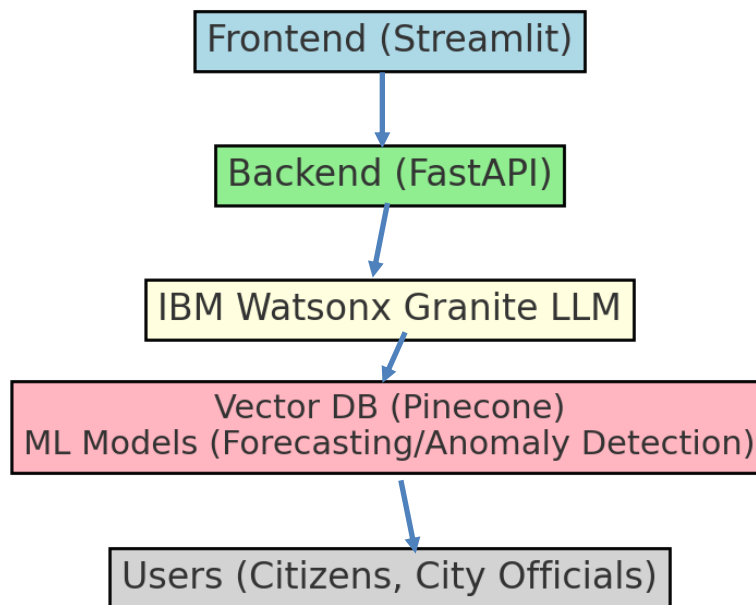
Frontend (Streamlit): Modular dashboard with chat, feedback, KPI views, policy search, eco tips, and anomaly detection.

Backend (FastAPI): RESTful APIs handling uploads, ML forecasting, anomaly detection, and LLM integration.

LLM Integration (IBM Watsonx Granite): Summarization, eco tips, sustainability reports, and conversational AI.

Vector Search (Pinecone): Semantic policy search with document embeddings.

ML Modules: Forecasting and anomaly detection using scikit-learn, pandas, and matplotlib.



4. Setup Instructions

Prerequisites:

- Python 3.9+
- FastAPI, Streamlit
- IBM Watsonx & Pinecone API keys
- scikit-learn, pandas, matplotlib
- Internet access

Installation Process:

- Clone the repository
- Install dependencies from requirements.txt
- Configure API credentials in .env
- Run FastAPI backend
- Launch Streamlit frontend
- Upload documents/data and explore modules

5. Folder Structure

app/ – FastAPI backend logic

app/api/ – Routers for chat, feedback, eco tips, policies, KPIs

ui/ – Streamlit frontend components

smart_dashboard.py – Entry script for dashboard

granite_llm.py – LLM service functions (summaries, eco tips, chat)

document_embedder.py – Converts documents to embeddings

kpi_file_forecaster.py – Forecasts urban KPIs

anomaly_file_checker.py – Flags anomalies in datasets

report_generator.py – Creates sustainability reports

6. Running the Application

- Start FastAPI backend
- Run Streamlit dashboard
- Navigate via sidebar
- Upload policies or KPI data
- Interact with chat assistant and eco tools
- View forecasts, anomalies, and sustainability reports

7. API Documentation

POST /chat/ask – AI-generated responses

POST /upload-doc – Uploads and embeds documents

GET /search-docs – Semantic policy search

GET /get-eco-tips – Provides sustainability tips

POST /submit-feedback – Stores citizen feedback

8. Authentication

- Token-based authentication (JWT / API keys)
- OAuth2 with IBM Cloud
- Role-based access (admin, citizen, researcher)
- Planned: session management & history tracking

9. User Interface

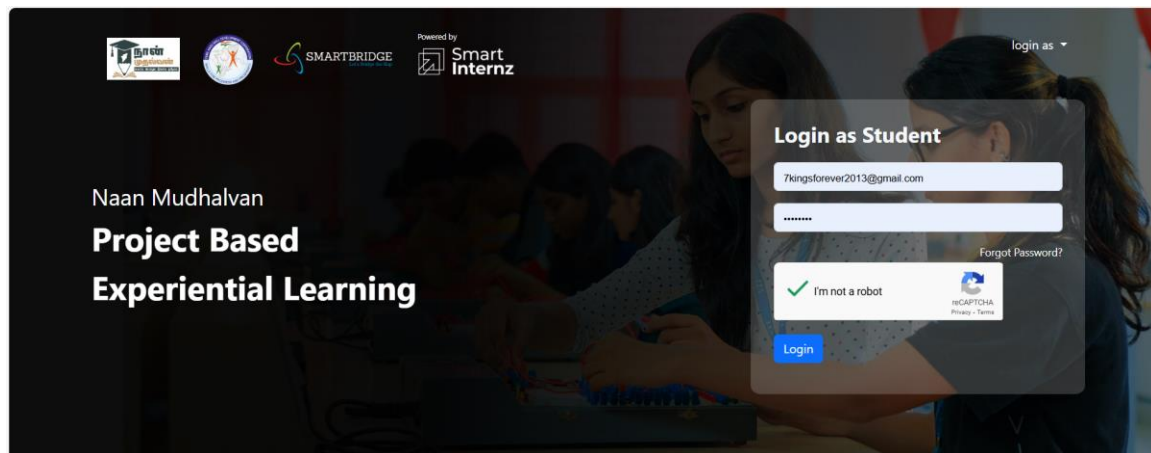
- Sidebar navigation with themed icons
- KPI visualizations with summary cards
- Chat assistant with real-time AI responses
- Feedback forms with issue categorization

- Policy summarization and eco tips display
- Report generation and download

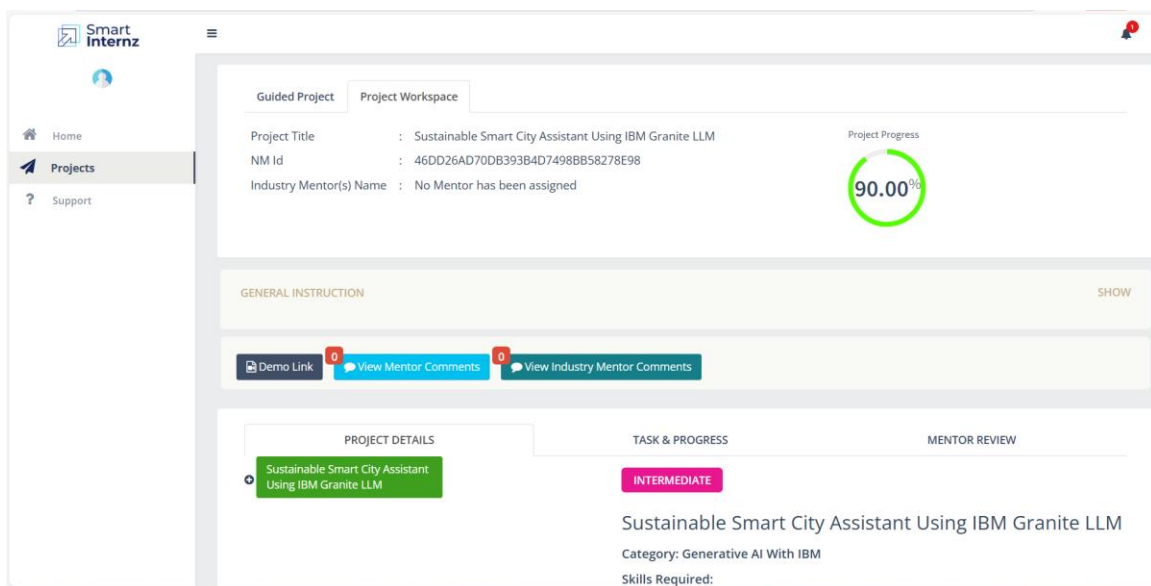
10. Testing

- Unit Testing: For backend services and ML functions
- API Testing: Swagger UI, Postman
- Manual Testing: For chat, policy search, forecasting
- Edge Case Handling: Large files, malformed inputs, invalid API keys

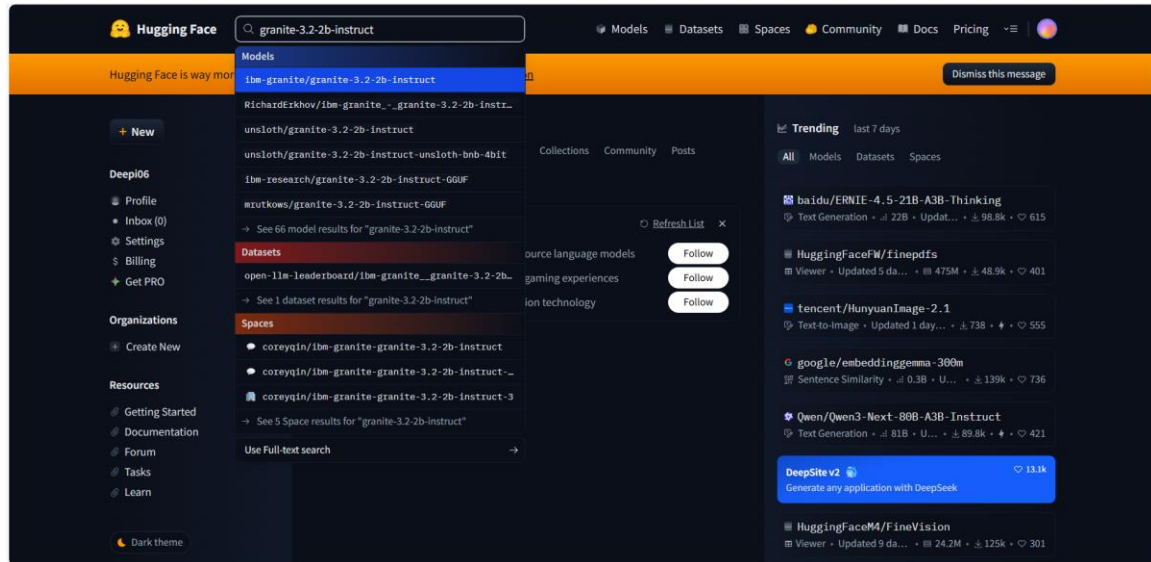
11. Screenshots



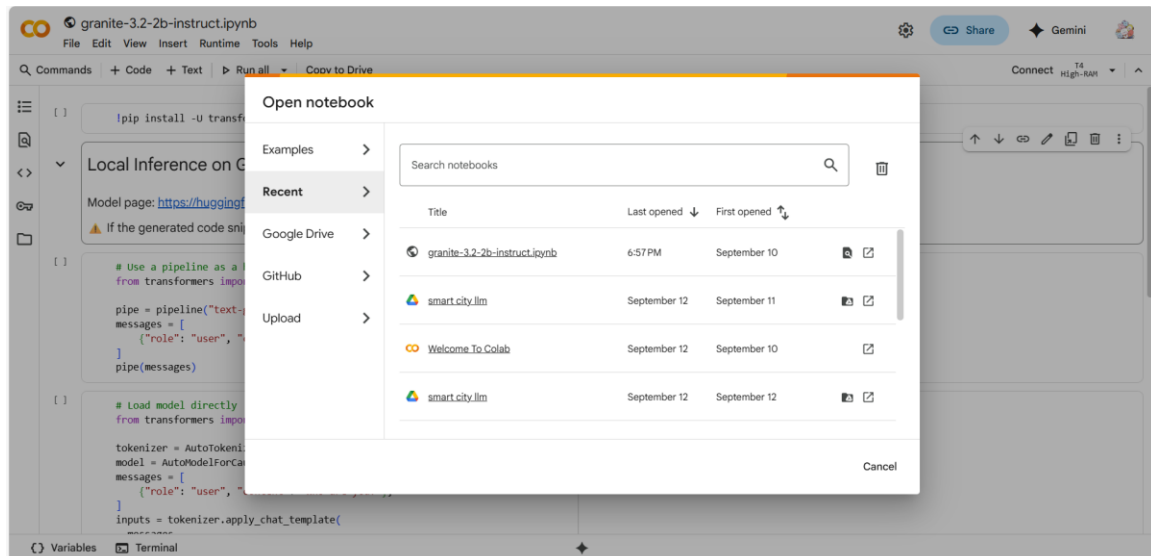
Smart Internz login page



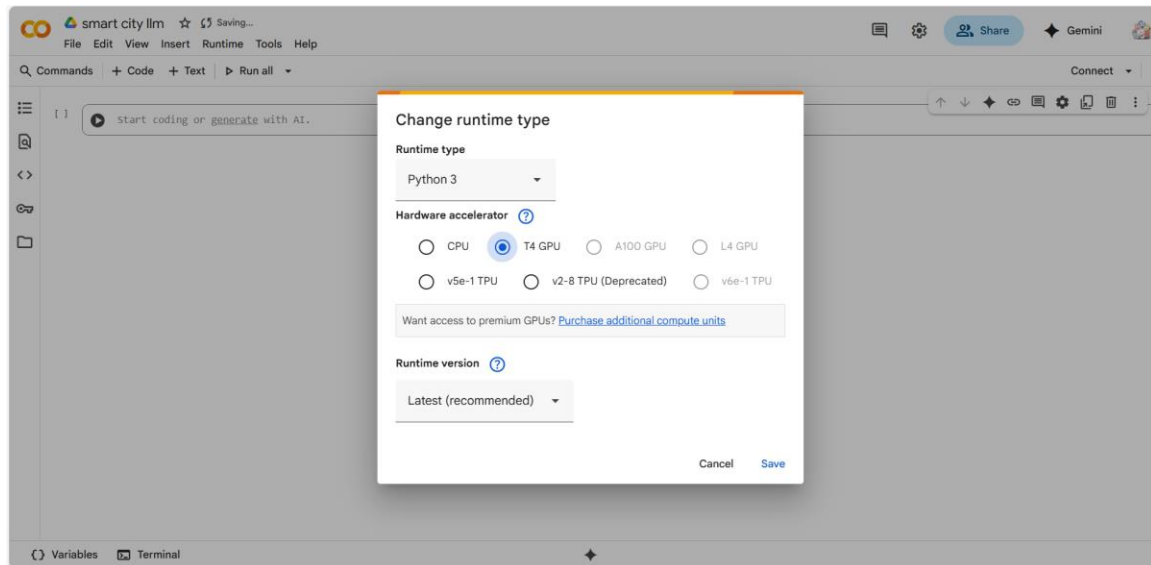
Smart Internz dashboard



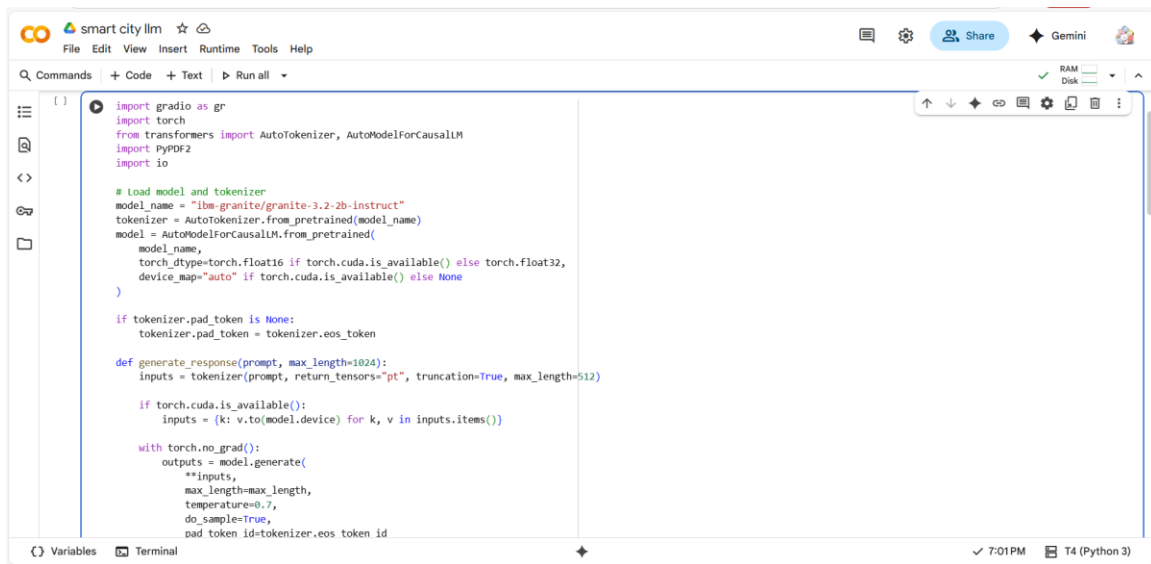
granite-3.2-2b-instruct in hugging face



New Notebook page in google colab



Choose "T4 GPU" and click on "save"



```
smart city llm
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
[ ]
response = tokenizer.decode(outputs[0], skip_special_tokens=True)
response = response.replace(prompt, "").strip()
return response

def extract_text_from_pdf(pdf_file):
    if pdf_file is None:
        return ""

    try:
        pdf_reader = PyPDF2.PdfReader(pdf_file)
        text = ""
        for page in pdf_reader.pages:
            text += page.extract_text() + "\n"
        return text
    except Exception as e:
        return f"Error reading PDF: {str(e)}"

def eco_tips_generator(problem_keywords):
    prompt = f"Generate practical and actionable eco-friendly tips for sustainable living related to: {problem_keywords}. Provide specific solutions and suggestions:"
    return generate_response(prompt, max_length=1000)

def policy_summarization(pdf_file, policy_text):
    # Get text from pdf or direct input
    if pdf_file is not None:
        content = extract_text_from_pdf(pdf_file)
        summary_prompt = f"Summarize the following policy document and extract the most important points, key provisions, and implications:\n\n{content}"
    else:
        summary_prompt = f"Summarize the following policy document and extract the most important points, key provisions, and implications:\n\n{policy_text}"
    return generate_response(summary_prompt, max_length=1200)

Variables Terminal 7:01 PM T4 (Python 3)
```

```
smart city llm
File Edit View Insert Runtime Tools Help
Commands + Code + Text Run all
[ ]
with gr.Row():
    with gr.Column():
        keywords_input = gr.Textbox(
            label="Environmental Problem/Keywords",
            placeholder="e.g., plastic, solar, water waste, energy saving...",
            lines=3
        )
        generate_tips_btn = gr.Button("Generate Eco Tips")

    with gr.Column():
        tips_output = gr.Textbox(label="Sustainable Living Tips", lines=15)

    generate_tips_btn.click(eco_tips_generator, inputs=keywords_input, outputs=tips_output)

with gr.Tabitem("Policy Summarization"):
    with gr.Row():
        with gr.Column():
            pdf_upload = gr.File(label="Upload Policy PDF", file_types=[".pdf"])
            policy_text_input = gr.Textbox(
                label="Or paste policy text here",
                placeholder="Paste policy document text...",
                lines=5
            )
            summarize_btn = gr.Button("Summarize Policy")

        with gr.Column():
            summary_output = gr.Textbox(label="Policy Summary & Key Points", lines=20)

    summarize_btn.click(policy_summarization, inputs=[pdf_upload, policy_text_input], outputs=summary_output)

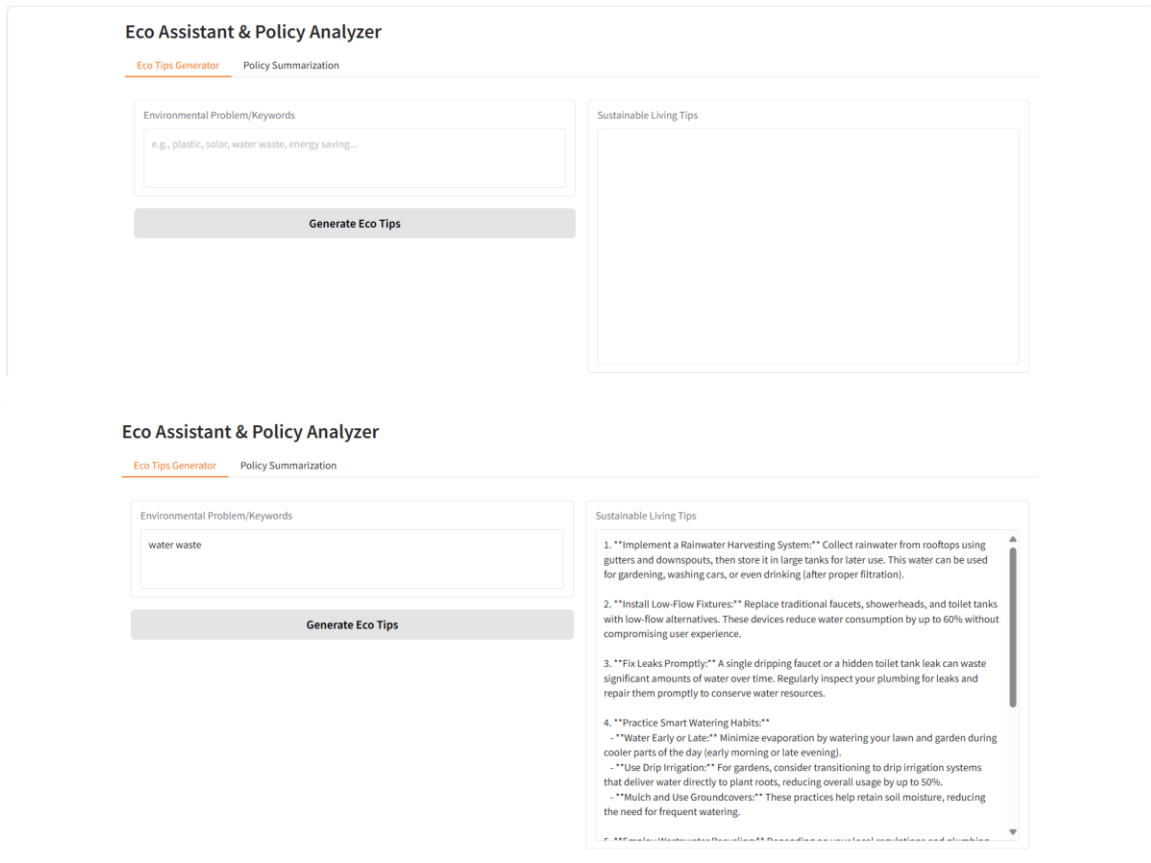
app.launch(share=True)
```

Sustainable Smart City Assistant Using IBM Granite LLM coding page



```
/usr/local/lib/python3.12/dist-packages/huggingface_hub/utils/_auth.py:94: UserWarning:
The secret 'HF_TOKEN' does not exist in your Colab secrets.
To authenticate with the Hugging Face Hub, create a token in your settings tab (https://huggingface.co/settings/tokens), set it as secret in your Google Colab and restart your notebook.
You will be able to reuse this secret in all of your notebooks.
Please note that authentication is recommended but still optional to access public models or datasets.
warnings.warn(
tokenizer_config.json: 8.88k/? [00:00<00:00, 872kB/s]
vocab.json: 777k/? [00:00<00:00, 25.4MB/s]
merges.txt: 442k/? [00:00<00:00, 16.3MB/s]
tokenizer.json: 3.48M/? [00:00<00:00, 79.1MB/s]
added_tokens.json: 100% [00:00<00:00, 87.0/87.0, 7.06kB/s]
special_tokens_map.json: 100% [00:00<00:00, 701/701, 86.0kB/s]
config.json: 100% [00:00<00:00, 786/786, 53.6kB/s]
`torch_dtype` is deprecated! Use `dtype` instead!
model.safetensors.index.json: 29.8k/? [00:00<00:00, 1.70MB/s]
Fetching 2 files: 100% [00:24<00:00, 84.17s/it]
model-00001-of-00002.safetensors: 100% [00:23<00:00, 5.00GiB/s]
model-00002-of-00002.safetensors: 100% [00:01<00:00, 67.1M/67.1M, 20.8MB/s]
Loading checkpoint shards: 100% [00:19<00:00, 8.00s/it]
generation_config.json: 100% [00:00<00:00, 137/137, 8.81kB/s]
Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://4c489e9a0123246ea3.gradio.live
```

● Click on the URL to open the Gradio Application click on the link



Sustainable Smart City Assistant Home page

12. Known Issues

- Limited language support
- Requires stable cloud connectivity
- Dependent on API quota limits

13. Future Enhancements

- Multi-language support
- Integration with IoT and city sensors
- Advanced anomaly detection (deep learning)
- Mobile-friendly dashboard
- Doctoral/official verification of eco policies