

Sustainable Smart City AI using IBM Granite

Project Documentation

1. Introduction

Team Leader: Deepika P.

Team Member: Aasmitha P

Team Member: Harini R.

Team Member: Harini V.S.

Team Member: Nayana E.K

This project, Sustainable Smart City AI using IBM Granite, explores the use of artificial intelligence to solve real-world urban challenges. Modern cities face growing environmental concerns, including waste management, water scarcity, rising energy demands, and complex policy frameworks that are often inaccessible to the general public. By leveraging IBM's Granite large language models, our team aims to bridge the gap between government-level planning and individual action. The system provides eco-friendly lifestyle tips to citizens, summarizes complex policy documents into simple terms, and lays the groundwork for predictive tools such as resource forecasting and anomaly detection.

2. Object Overview

The purpose of the project is to democratize access to sustainability knowledge. Governments and organizations often release detailed reports and lengthy policy documents, but these are not always easy for ordinary people to understand. As a result, important measures—such as new recycling regulations or renewable energy incentives—do not always reach the communities they are meant to benefit. Our solution makes this information more accessible by using AI to automatically summarize policies into short, clear highlights. Citizens can then make decisions with better awareness of their rights, responsibilities, and available opportunities.

In addition to policy simplification, the system provides an **eco-tip generator** that delivers actionable advice for everyday sustainable living. For example, a user concerned about “plastic waste” can receive suggestions like replacing single-use plastics with biodegradable alternatives or organizing local recycling drives. Someone searching for “energy saving” can learn about switching to LED lights, monitoring home appliances, and making use of solar options. Beyond individuals, city planners can use this feature to promote large-scale initiatives and encourage citizen participation.

The project also sets the foundation for advanced features. Resource forecasting can help predict water and electricity demand in growing neighborhoods. KPI forecasting enables governments to measure progress against sustainability goals. Anomaly detection identifies sudden deviations in data, such as spikes in air pollution or waste generation, allowing

authorities to act quickly. Finally, a feedback loop will ensure that citizen suggestions are collected and integrated into planning, making the city more participatory and dynamic.

3. Architecture

The architecture of this system is modular, balancing simplicity in the prototype stage with scalability for future deployment. The **front-end** currently uses Gradio, which provides a lightweight interface where users can type queries, upload files, and view results instantly. This choice makes it easier to iterate quickly and demonstrate features to stakeholders. For more advanced dashboards, Streamlit is considered as an alternative since it supports interactive visualizations and richer layouts.

The **back-end** will be built using FastAPI, a modern Python framework that excels in speed and scalability. FastAPI will handle API endpoints for eco-tips, policy summaries, and forecasting services. It will also be responsible for user authentication, logging, and request handling. By separating front-end and back-end responsibilities, the architecture allows each component to evolve independently.

The **AI engine** relies on IBM WatsonX Granite, accessed via Hugging Face Transformers. Granite is a large language model capable of understanding human language, generating structured summaries, and producing context-aware advice. To improve accuracy and context, a vector database like Pinecone or FAISS will be used for retrieval-augmented

generation (RAG), allowing the system to reference policy documents or local datasets during response generation.

Finally, **machine learning modules** for forecasting and anomaly detection will be integrated. These modules will analyze historical trends in urban data—such as water consumption, electricity usage, or waste levels—to predict future demand and detect unusual changes. For production deployment, Docker containers and Kubernetes clusters will ensure that the system runs reliably with GPU acceleration and autoscaling support.

4. Setup Instructions

Before running the project, certain prerequisites are required. The user must have Python 3.10 or later installed, along with pip or conda for package management. If a machine with an NVIDIA GPU is available, CUDA drivers should be installed to take advantage of faster model inference. Hugging Face API tokens and IBM Granite keys are also necessary to access the model. It is recommended to use a virtual environment to avoid dependency conflicts.

The installation process begins by cloning the project repository and creating a virtual environment. Once activated, dependencies can be installed using `pip install -r requirements.txt`. GPU users should verify that the correct PyTorch build for CUDA is installed, as mismatches can cause runtime errors. Environment variables, including authentication tokens, must be configured before running the

app. Finally, the user can start the Gradio application by running `python app.py` locally or launch the provided Colab notebook for cloud-based testing.

This modular setup allows flexibility. Developers can work in Colab for quick prototyping, use Hugging Face Spaces for sharing lightweight demos, or deploy on private cloud infrastructure for production workloads. Each environment has different strengths—Colab provides GPU access for free, Hugging Face Spaces allows easy sharing, and cloud servers offer long-term stability and scalability.

5. Folder Structure

The project repository is structured to promote maintainability. At the top level, `app.py` serves as the main entry point for the Gradio prototype. The `backend` folder will contain FastAPI routes and business logic. The `models` directory is reserved for AI-related utilities, such as model loading, prompt engineering, and forecasting algorithms. The `utils` folder includes helper scripts for PDF extraction, error handling, and response post-processing.

A `data` folder stores test policy documents, while the `docs` folder contains formal documentation and screenshots. The `tests` folder ensures code quality by providing both unit tests and integration tests. Finally, supporting files like `requirements.txt`, `README.md`, and configuration scripts ensure that setup and execution are straightforward for new developers.

6. Running the Application

To run the project locally, the user activates their virtual environment, ensures environment variables are set, and executes `python app.py`. This launches a Gradio interface accessible through the local URL `http://127.0.0.1:7860`. Users can then interact with the eco-tip generator or upload a PDF to test policy summarization.

In Google Colab, the notebook must first install all required libraries. After logging into Hugging Face, the IBM Granite model can be downloaded and used for inference. Gradio's `share=True` option provides a public URL, making it easy to demonstrate the application to others without setting up a server.

For deployment, Hugging Face Spaces is ideal for small-scale demos but is limited by hardware availability. In production, Docker containers ensure portability, while Kubernetes manages scaling and fault tolerance. With this approach, the project can grow from a simple demo to a city-scale AI platform.

7. API Documentation (Planned)

The project will soon provide a dedicated API layer. The `/eco-tips` endpoint will accept keywords and context parameters, returning a JSON response with actionable suggestions ranked by priority. The `/policy-summary` endpoint will process

uploaded PDFs or raw text and output a structured summary highlighting key provisions, affected stakeholders, and implications. Advanced endpoints like /forecast and /anomaly-detect will process time-series data and generate predictions or highlight irregularities.

Each endpoint will follow REST conventions, using JSON payloads and HTTP status codes. For example, a request to /eco-tips might look like:

```
{ "keywords": ["plastic waste", "recycling"], "context": "urban households" }
```

And the response would return a structured list of prioritized tips.

This design ensures that the assistant is not just an application but also a reusable service for integration with city dashboards, mobile apps, or third-party systems.

8. Authentication

Currently, the prototype does not enforce authentication to allow easier testing. However, production deployment will include strict security measures. API access will be controlled using API keys or OAuth tokens, which must be securely stored in a secrets manager. TLS encryption will ensure secure communication, and role-based access control (RBAC) can be implemented for different user groups. Logging and audit trails will be added to monitor activity and prevent misuse. Rate

limiting will protect the system from denial-of-service attacks while maintaining reliable access for legitimate users.

9. User Interface

The user interface is intentionally simple to encourage adoption. It provides two tabs: one for generating eco-tips and another for summarizing policies. Textboxes allow users to type queries or upload files, while clear buttons guide actions such as “Generate Tips” or “Summarize Policy.”

Future improvements will focus on usability and accessibility. Export options will let users save results as PDF or CSV. Multi-language support will help non-English speakers interact with the system. Progress indicators will make it easier to follow long-running tasks. Additionally, accessibility features like screen reader compatibility will ensure that people with disabilities can also use the system effectively.

10. Testing

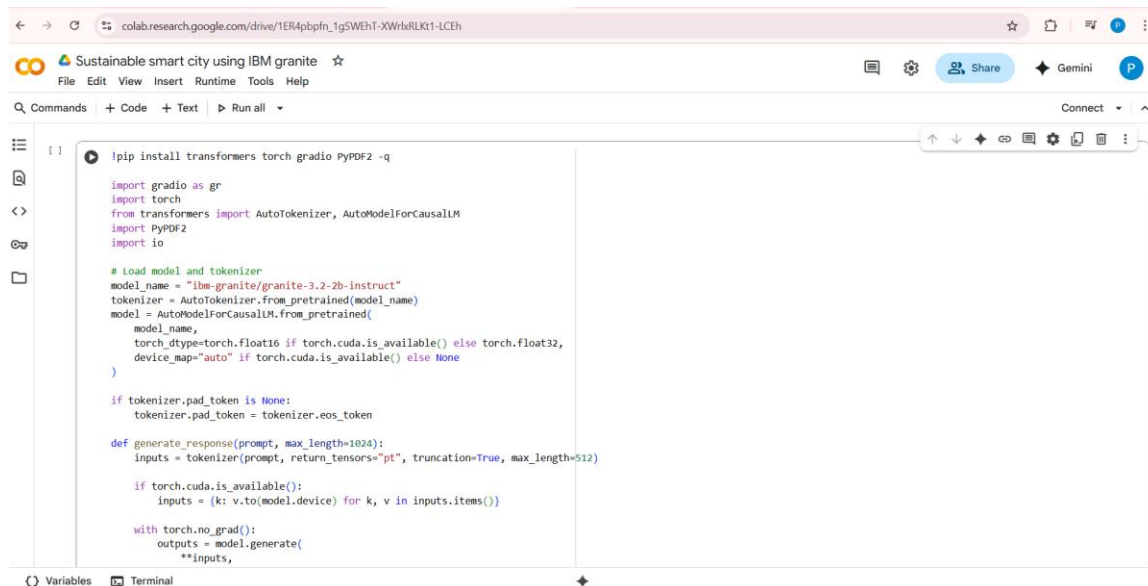
A strong testing framework is vital for reliability. Unit tests will cover smaller components, such as text extraction from PDFs or proper formatting of model responses. Integration tests will simulate end-to-end workflows, ensuring that entering text in the UI produces the expected model output. Performance testing will simulate hundreds of users to check system stability, while security testing will confirm that authentication and authorization are correctly enforced once implemented.

Continuous Integration (CI) pipelines will automate these tests whenever new code is pushed. This approach ensures that the system remains stable as features are added and refined.

11. Screenshots

1. Running the Code in Google Colab

The first screenshot demonstrates the environment setup, including installation of dependencies, model loading, and test queries. It confirms that the IBM Granite model runs successfully in Colab.



The screenshot shows a Google Colab notebook interface. The browser address bar displays the URL: `colab.research.google.com/drive/1ER4pbpfm_1gSWEhT-XWlxRLK11-LCEh`. The notebook title is "Sustainable smart city using IBM granite". The code editor contains the following Python code:

```
!pip install transformers torch gradio PyPDF2 -q

import gradio as gr
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM
import PyPDF2
import io

# 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

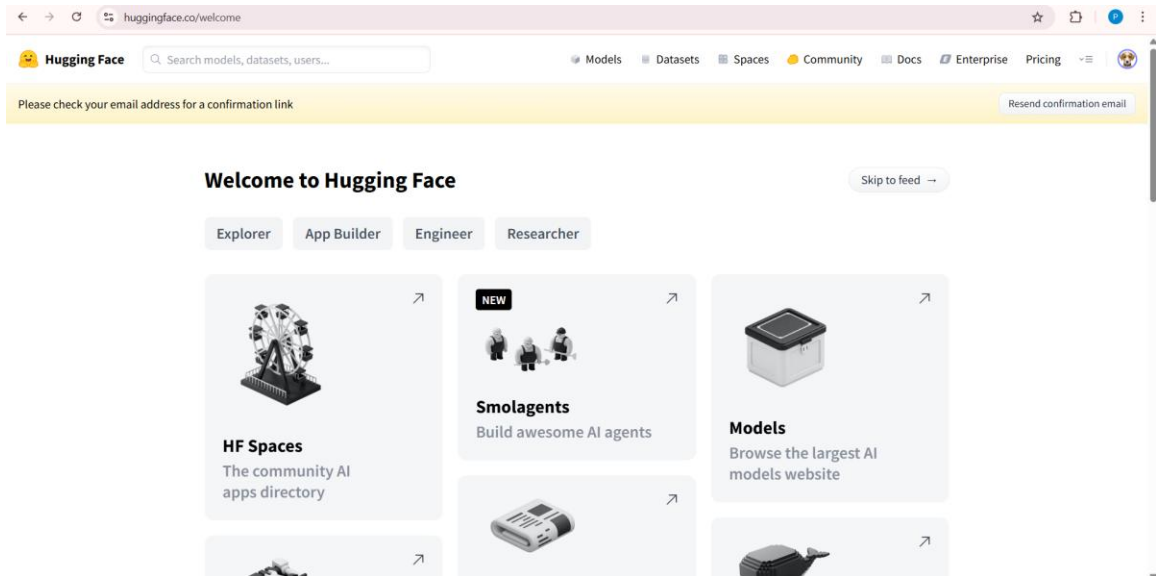
def generate_response(prompt, max_length=1024):
    inputs = tokenizer(prompt, return_tensors="pt", truncation=True, max_length=512)

    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,
```

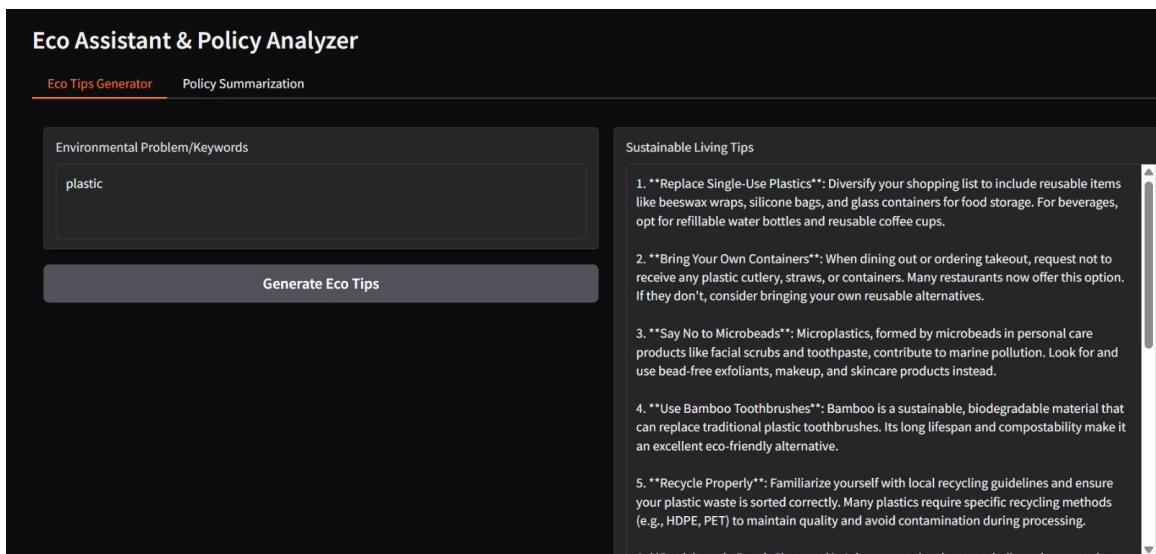
2. Signing in to Hugging Face

The second screenshot shows the integration step, where the user authenticates with Hugging Face to access the Granite model. This step is critical for connecting local or Colab environments with Hugging Face's model hub.



3. Output with AI Eco Tips

The final screenshot captures the Gradio application in action. After entering keywords such as “water conservation” or “plastic waste,” the system generates customized eco-friendly suggestions, proving that the AI assistant works as intended.



12. Known Issues

While functional, the prototype has limitations. Model responses sometimes echo the input text, which requires

additional post-processing. Scanned PDFs are not supported since the system lacks OCR capabilities, though this can be solved with libraries like Tesseract. Performance on Hugging Face Spaces is limited by hardware, leading to slow responses for large documents. Without authentication, the current setup cannot be safely deployed in a real-world municipal environment. These limitations are known and will be addressed in future iterations.

13. Future Enhancements

Future plans include fully separating the front-end and back-end, integrating FastAPI for API endpoints, and using Pinecone for retrieval-augmented generation. Advanced ML models for forecasting and anomaly detection will be added, helping cities predict demand and spot unusual trends. Streamlit dashboards will offer interactive charts and analytics for decision-makers. Long-term goals include integrating IoT sensors for real-time monitoring, expanding multilingual support to serve global communities, and piloting the system with local governments. These enhancements will transform the project from a prototype into a comprehensive AI-powered sustainability assistant.