**Sustainable Smart City Assistant using IBM Granite LLM**

**Team** **leader :** S.SRIKANTH

**Team** **Member :** J.DINESH BALAJI

**Team** **Member :** R.AMITHRAJ

**Team** **Member :** R.SELVA KARTHICK

**1.Introduction**

Artificial Intelligence (AI) is one of the most transformative technologies of our time. It refers to the ability of machines and computer systems to perform tasks that normally require human intelligence, such as understanding natural language, making decisions, solving problems, and learning from data. AI is now widely used across industries to improve efficiency, provide better user experiences, and support sustainable solutions.

IBM (International Business Machines Corporation) is a global leader in technology and innovation. For decades, IBM has been at the forefront of building advanced computing systems and AI-driven platforms. Its research and products are dedicated to solving real-world challenges, from enterprise solutions to sustainability and smart city development.

Granite is a family of Large Language Models (LLMs) developed by IBM. These models are designed to understand and generate human-like language, enabling applications such as chat assistants, document summarization, and problem-solving in multiple domains. The Granite LLM stands out for its enterprise-grade reliability, scalability, and ethical AI practices. It is built to help developers create AI applications that are secure, transparent, and tailored to business or societal needs.

**2. Project Overview**

* **Purpose/goal:**

The main purpose of this project is to develop a Sustainable Smart City Assistant that uses IBM Granite LLM to improve urban living and support eco-friendly practices. The goal is to create an intelligent system that can help both citizens and administrators by providing accurate information, useful suggestions, and decision-making support. Through natural language interaction, the assistant makes it easier for people to access important data and guidance in simple terms.

This project also aims to address key challenges faced by modern cities, such as energy management, waste reduction, and community engagement. By applying the capabilities of IBM Granite LLM, the assistant not only promotes sustainability but also enhances transparency, efficiency, and inclusiveness in city governance. Ultimately, the goal is to show how artificial intelligence can contribute to building smarter, greener, and more citizen-focused cities for the future.

Another important aspect of this project is its ability to generate personalized sustainability tips for citizens. The assistant analyzes user behavior and provides daily eco-friendly suggestions that can be easily followed in everyday life. These tips may include guidance on saving electricity, conserving water, reducing plastic usage, adopting public transport, and practicing proper waste segregation. By giving simple and practical recommendations, the system motivates individuals to make small lifestyle changes that, when combined across the community, create a large positive impact on the environment. This feature ensures that sustainability is not only a policy-level concept but also a personal habit that citizens can adopt to support the smart city vision.

* **Features:**

The Sustainable Smart City Assistant comes with a wide range of features designed to make city management more efficient and citizen-friendly. One of the key features is the conversational interface, which allows natural language interaction. This enables citizens and officials to ask questions, receive updates, and get guidance in plain and simple terms. Another important feature is policy summarization, which helps convert lengthy government documents into concise and actionable summaries, making complex policies easier to understand.

The system also includes resource forecasting, where predictive analytics are used to estimate future energy, water, and waste usage based on historical and real-time data. To promote eco-friendly living, the assistant provides an eco-tip generator that gives personalized sustainability advice and daily actions to reduce environmental impact. Additionally, the citizen feedback loop encourages community engagement by collecting and analyzing public opinions, which can then be used to improve city planning and services.

For administrative purposes, the assistant offers KPI forecasting to project key performance indicators, helping city officials track progress and plan ahead. It also includes anomaly detection, which acts as an early warning system by identifying unusual patterns in sensor or usage data that could signal potential problems. To ensure flexibility, the system supports multimodal input, meaning it can handle text, PDFs, and CSV files for document analysis and forecasting. Finally, the assistant provides a user-friendly interface through Streamlit or Gradio UI, offering an intuitive dashboard for both citizens and officials to interact easily with the system.

**3. Architecture**

The architecture of the Sustainable Smart City Assistant is structured to integrate artificial intelligence, machine learning, and data management in a single unified platform. It is divided into multiple layers, where each component has a clear role in ensuring smooth operation. The major layers include the frontend interface, backend services, large language model integration, vector database, and machine learning modules. These layers together create an intelligent system that delivers real-time guidance, forecasting, and decision support.

**Frontend (Streamlit)**

The frontend of the system is designed using Streamlit, which provides a clean and interactive web-based interface. In architectural terms, this component belongs to the presentation layer, responsible for how users interact with the assistant. The frontend includes dashboards for data visualization, chat windows for conversation with the AI model, forms for citizen feedback, and sections for uploading documents such as PDFs or CSV files. Its modular design makes it easy to extend the interface with new features in the future.

For end users, the frontend ensures accessibility and simplicity. Citizens use it to ask questions, receive sustainability tips, and explore policy summaries, while administrators rely on it to view forecasting reports, detect anomalies, and monitor city performance indicators. By keeping the interface intuitive, the system architecture ensures that technical complexity remains hidden behind a user-friendly layer.

**Backend (FastAPI)**

The backend is powered by FastAPI, which acts as the control center of the system. Architecturally, it forms the application layer that connects the frontend with the AI models and data modules. It manages all incoming user requests and processes them by coordinating with other components. For example, when a user uploads a policy document, the backend processes the file and sends it to the Granite LLM for summarization. Similarly, when forecasting is requested, the backend calls the machine learning modules and delivers the results to the frontend.

The backend also exposes various API endpoints for different services, such as eco-tip generation, document embedding, and anomaly detection. The choice of FastAPI adds performance efficiency, scalability, and ease of integration with external tools.

**LLM Integration (IBM Watsonx Granite)**

At the core of the architecture lies IBM Watsonx Granite LLM, which provides the system’s intelligence. This component is responsible for understanding natural language, generating responses, and simplifying complex documents. It operates within the AI layer of the architecture, making the assistant capable of interactive conversations, policy summarization, and sustainability recommendations. By carefully designing prompts, the LLM generates meaningful outputs that can be used by both citizens and city administrators.

**Vector Search (Pinecone)**

To handle document retrieval efficiently, the architecture includes a Pinecone vector database. Uploaded policy documents are embedded into numerical vectors using Sentence Transformers and stored in Pinecone. When users make a query, Pinecone performs semantic search using cosine similarity to return the most relevant sections. This forms the data storage and retrieval layer of the architecture, ensuring that users can quickly access the right information without manual searching.

**ML Modules (Forecasting and Anomaly Detection)**

The architecture also integrates machine learning modules that support forecasting and anomaly detection. These modules analyze both historical and real-time data to predict trends such as future water, energy, or waste usage. At the same time, anomaly detection monitors data streams to identify irregular patterns that may signal problems. This predictive layer enhances the decision-making ability of city administrators, making the system proactive rather than reactive.

**Overall View**

In summary, the architecture is composed of five major layers: the presentation layer (Streamlit), the application layer (FastAPI), the intelligence layer (Granite LLM), the storage and retrieval layer (Pinecone), and the predictive layer (ML models). Each of these components is modular and can be independently upgraded, making the architecture flexible and scalable. This design ensures that the Sustainable Smart City Assistant can handle present requirements while also being adaptable to future enhancements.

**4. Setup Instructions**

Before running the Sustainable Smart City Assistant, certain setup requirements must be met to ensure smooth installation and execution. These setup instructions provide a step-by-step guide for preparing the environment, installing the necessary tools, and configuring the project for use. Since the project integrates multiple technologies, both software prerequisites and external dependencies must be satisfied.

**Prerequisites**

The project requires Python 3.9 or later as the primary programming language for development. Python was chosen because of its flexibility, large collection of libraries, and compatibility with data science and machine learning tools. Along with Python, package management tools such as pip and virtual environment managers are necessary to handle dependencies and isolate the project environment from the system.

API keys are another important requirement, as the system integrates with external services like IBM Watsonx Granite LLM and Pinecone Vector Database. These keys allow secure communication between the application and the cloud services. In addition, an active internet connection is required because the assistant relies on cloud-based models and APIs to process natural language, perform vector searches, and deliver results.

The project combines multiple software frameworks and libraries to achieve its functionality. For the frontend development, Streamlit is used to design an interactive dashboard where citizens and administrators can easily interact with the assistant. The backend is built with FastAPI, which serves as a high-performance REST framework to handle requests, manage data, and connect the frontend with AI modules.

For the AI integration, the project uses IBM Watsonx Granite LLM, which provides natural language understanding and text generation capabilities. Pinecone is used for vector storage and semantic search, enabling fast retrieval of relevant policy information. For predictive analytics and anomaly detection, Scikit-learn, Pandas, and Matplotlib are used to build lightweight machine learning modules that can forecast trends and visualize results.

Together, these software tools create a complete ecosystem where data flows seamlessly from the user interface to the backend, is processed by AI and machine learning models, and is finally presented back to users in a clear and user-friendly format.

**Installation Process**

Once the prerequisites are satisfied, the next step is to install and configure the project environment. The installation process begins with cloning the project repository from GitHub. This provides access to all the source code, including backend scripts, frontend components, and integration modules. After cloning, it is important to create a virtual environment to keep the dependencies organized and isolated from the system’s global environment.

With the environment prepared, the required libraries and frameworks are installed using the requirements.txt file provided in the repository. This file contains all the dependencies needed for running the project, including FastAPI for backend operations, Streamlit for the user interface, and machine learning libraries such as Scikit-learn and Pandas. Installing these packages ensures that the project runs consistently across different systems.

The next step is to configure credentials by creating a .env file. In this file, API keys for services such as IBM Watsonx Granite LLM and Pinecone are stored securely. These keys allow the system to connect with the cloud services for natural language processing, vector search, and policy summarization. Once the configuration is complete, the backend server can be launched using FastAPI. Running this server enables all the API endpoints, making it possible for the frontend to communicate with the backend.

Finally, the frontend is launched using Streamlit, which opens an interactive dashboard in the web browser. From this dashboard, users can upload documents, interact with the chat assistant, submit feedback, and view forecasting or anomaly detection results. At this stage, the Sustainable Smart City Assistant is fully functional, and users can begin exploring its features in real time.

**5. Folder Structure**

The Sustainable Smart City Assistant project follows a well-organized folder structure to separate different components of development and make the application easy to manage. Each folder and file has a specific purpose, ensuring that the project remains modular, scalable, and easy to understand for both developers and users.

At the root level of the project, the main app/ directory contains all the backend logic developed using FastAPI. Inside this directory, different modules are maintained, including routing functions, data models, and integration layers. The app/api/ subfolder is specifically designed to manage modular API routes. These include routes for chat interactions, citizen feedback, report generation, and document vectorization. By dividing routes into submodules, the project ensures clarity and maintainability.

The ui/ folder contains all the components related to the frontend interface. This section is built using Streamlit and includes layouts, forms, dashboards, and navigation menus. The UI folder is responsible for how the application looks and feels, allowing both citizens and city administrators to navigate the system easily and interact with the assistant.

In addition to these folders, several important Python files are included at the project root. The smart\_dashboard.py file serves as the entry point for launching the Streamlit dashboard. The granite\_llm.py file manages communication with the IBM Watsonx Granite model, handling tasks such as document summarization, eco-tip generation, and conversational responses. Similarly, the document\_embedder.py script converts uploaded documents into vector embeddings and stores them in Pinecone for fast and efficient retrieval.

Other specialized files include kpi\_file\_forecaster.py, which handles forecasting of resource consumption trends, and anomaly\_file\_checker.py, which identifies unusual patterns in data. The report\_generator.py script is responsible for creating AI-generated sustainability reports that can be shared with city officials for decision-making.

By maintaining this structured folder system, the project achieves better modularity and easier debugging. Each functionality is clearly separated into its respective file or folder, making it simple for developers to expand features in the future without disturbing the existing system.

The Sustainable Smart City Assistant project follows a well-organized folder structure to separate different components of development and make the application easy to manage. Each folder and file has a specific purpose, ensuring that the project remains modular, scalable, and easy to understand for both developers and users.

At the root level of the project, the main app/ directory contains all the backend logic developed using FastAPI. Inside this directory, different modules are maintained, including routing functions, data models, and integration layers. The app/api/ subfolder is specifically designed to manage modular API routes. These include routes for chat interactions, citizen feedback, report generation, and document vectorization. By dividing routes into submodules, the project ensures clarity and maintainability.

The ui/ folder contains all the components related to the frontend interface. This section is built using Streamlit and includes layouts, forms, dashboards, and navigation menus. The UI folder is responsible for how the application looks and feels, allowing both citizens and city administrators to navigate the system easily and interact with the assistant.

In addition to these folders, several important Python files are included at the project root. The smart\_dashboard.py file serves as the entry point for launching the Streamlit dashboard. The granite\_llm.py file manages communication with the IBM Watsonx Granite model, handling tasks such as document summarization, eco-tip generation, and conversational responses. Similarly, the document\_embedder.py script converts uploaded documents into vector embeddings and stores them in Pinecone for fast and efficient retrieval.

Other specialized files include kpi\_file\_forecaster.py, which handles forecasting of resource consumption trends, and anomaly\_file\_checker.py, which identifies unusual patterns in data. The report\_generator.py script is responsible for creating AI-generated sustainability reports that can be shared with city officials for decision-making.

By maintaining this structured folder system, the project achieves better modularity and easier debugging. Each functionality is clearly separated into its respective file or folder, making it simple for developers to expand features in the future without disturbing the existing system.

**6. Running the Application**

The Sustainable Smart City Assistant is designed to provide real-time insights and management support for urban services using IBM Granite and AI technologies. Running the application involves a series of steps that ensure the system functions correctly and the user can interact with all its features.

**Step 1:** Setting up the Environment

Before running the application, it is necessary to prepare the working environment. Ensure that all required software and libraries are installed, including Python, IBM Granite SDK, and other dependencies. This setup provides the foundation for smooth execution of the application.

**Step 2:** Loading the Application

The application is launched through its main script. By executing the script, all modules such as traffic management, energy monitoring, and citizen services are initialized. This ensures that the system can collect data from various sources and process it efficiently.

**Step 3:** Interacting with the System

Once the application is running, users can access the interface to input data or view results. The interface is designed to be user-friendly, allowing both administrators and citizens to monitor services, receive alerts, and make informed decisions.

**Step 4:** Testing and Verification

After the application is running, it is important to test each module to ensure proper functionality. This includes checking data accuracy, response time, and integration between modules. Any issues detected during this stage are corrected to maintain the reliability of the system.

**Step 5:** Closing the Application

Finally, after using the system, the application can be safely closed. Proper shutdown ensures that all data is saved, and no module is left in an inconsistent state.

**7. API Documentation**

The Sustainable Smart City Assistant makes use of Application Programming Interfaces (APIs) to connect different parts of the system and allow smooth data exchange. API documentation explains how these connections work, what type of data is used, and how other applications or users can interact with the system. It acts as a guide for developers and ensures that the project can be easily maintained or expanded in the future.

In this project, the APIs are mainly used to collect real-time data, process it using IBM Granite, and deliver useful outputs to the user interface. For example, traffic data can be fetched through one API, while energy usage or waste management details can be accessed through another. Each API has a clear description of its purpose, the type of request it accepts, and the format of the response it provides.

The documentation also highlights how authentication and security are handled so that only authorized users can access the system. In addition, details about possible error messages and how to handle them are included to make troubleshooting easier.

By maintaining clear and structured API documentation, the Sustainable Smart City Assistant ensures that developers, administrators, and future users can easily understand, use, and improve the system.

**8. Authentication**

Authentication is an important feature of the Sustainable Smart City Assistant, as it ensures that only authorized users can access and manage the system. Since the project deals with sensitive data such as traffic reports, energy consumption, and public service details, maintaining strong security is essential.

In this application, authentication is handled through secure login methods and access control policies. Each user, whether an administrator or a citizen, is required to create a valid account with proper credentials. IBM Granite services support token-based authentication, which provides an additional layer of security by generating unique access tokens for every session. This prevents unauthorized users from entering the system and protects data from misuse.

Different levels of access can also be assigned based on the user role. For example, administrators may have permission to manage system settings and view detailed reports, while citizens may only be allowed to access limited services such as receiving alerts or viewing city updates.

By implementing strong authentication practices, the Sustainable Smart City Assistant guarantees the privacy of users, the security of stored data, and the overall reliability of the system.

**9. User Interface**

The User Interface (UI) of the Sustainable Smart City Assistant is designed to provide a simple and interactive platform for both administrators and citizens. A well-structured interface is essential, as it allows users to easily access the system’s features and understand the insights generated by IBM Granite.

The interface presents data in a clear and organized manner, often through dashboards, charts, and reports. For administrators, the UI offers advanced options such as monitoring city-wide energy usage, analyzing traffic conditions, and managing waste services. For citizens, it provides real-time updates, alerts, and easy-to-use options for reporting issues or receiving notifications.

Accessibility and ease of use are key priorities. The design follows a user-friendly approach with intuitive navigation, simple menus, and visually appealing layouts. This ensures that even non-technical users can benefit from the system without confusion.

By combining clarity, interactivity, and functionality, the User Interface plays a vital role in making the Sustainable Smart City Assistant effective, engaging, and accessible to all types of users.

**10. Testing**

Testing plays a crucial role in ensuring the Sustainable Smart City Assistant works efficiently and delivers accurate results. Since the project combines multiple modules such as traffic monitoring, energy management, and waste tracking, each part of the system must be verified individually and also as a whole.

**Unit Testing**

Each module is tested separately to confirm that it performs its specific function correctly. For example, the traffic monitoring module is checked to ensure it receives real-time data and displays accurate results.

**Integration Testing**

After verifying individual modules, the next step is to test how they work together. The integration testing phase ensures smooth communication between APIs, the database, and the user interface.

**System Testing**

In this stage, the complete application is tested in a real-world environment. It verifies whether the assistant can handle multiple requests, manage large datasets, and respond within acceptable time limits.

**User Testing**

Finally, the system is tested with real users such as administrators or citizens. Their feedback is used to check the usability, clarity of the interface, and reliability of the outputs.

**11. Future Enhancement**

The Sustainable Smart City Assistant is a powerful step toward building intelligent and eco-friendly urban systems. However, like any evolving technology, there is scope for continuous improvement and expansion. Future enhancements can focus on making the system more advanced, scalable, and citizen-friendly.

One major enhancement is the integration of advanced AI models for predictive analysis. By using IBM Granite with machine learning, the system could forecast traffic congestion, energy demand, or waste generation before they occur, allowing preventive actions.

Another improvement is the use of IoT-enabled sensors across the city. These sensors can provide more accurate real-time data, such as air quality, noise levels, and water usage, which would strengthen decision-making.

The assistant can also be enhanced with mobile applications for citizens, enabling them to report issues instantly, receive alerts, and participate actively in sustainable development. Additionally, adopting cloud scalability will help the system manage larger populations and more complex data without performance issues.

Finally, future versions may include multi-language support, data visualization dashboards, and integration with renewable energy grids, ensuring the system remains relevant and impactful for future smart cities.

**12. Conclusion**

The Sustainable Smart City Assistant using IBM Granite is a practical approach to modern urban development, combining AI, IoT, and data analytics to improve city services such as traffic, energy, waste, and water management. With its secure authentication, user-friendly interface, and real-time monitoring, the system provides reliable support for administrators and citizens alike. This project shows how technology can make cities smarter, greener, and more sustainable, while also offering opportunities for future enhancements like predictive analytics and mobile integration.

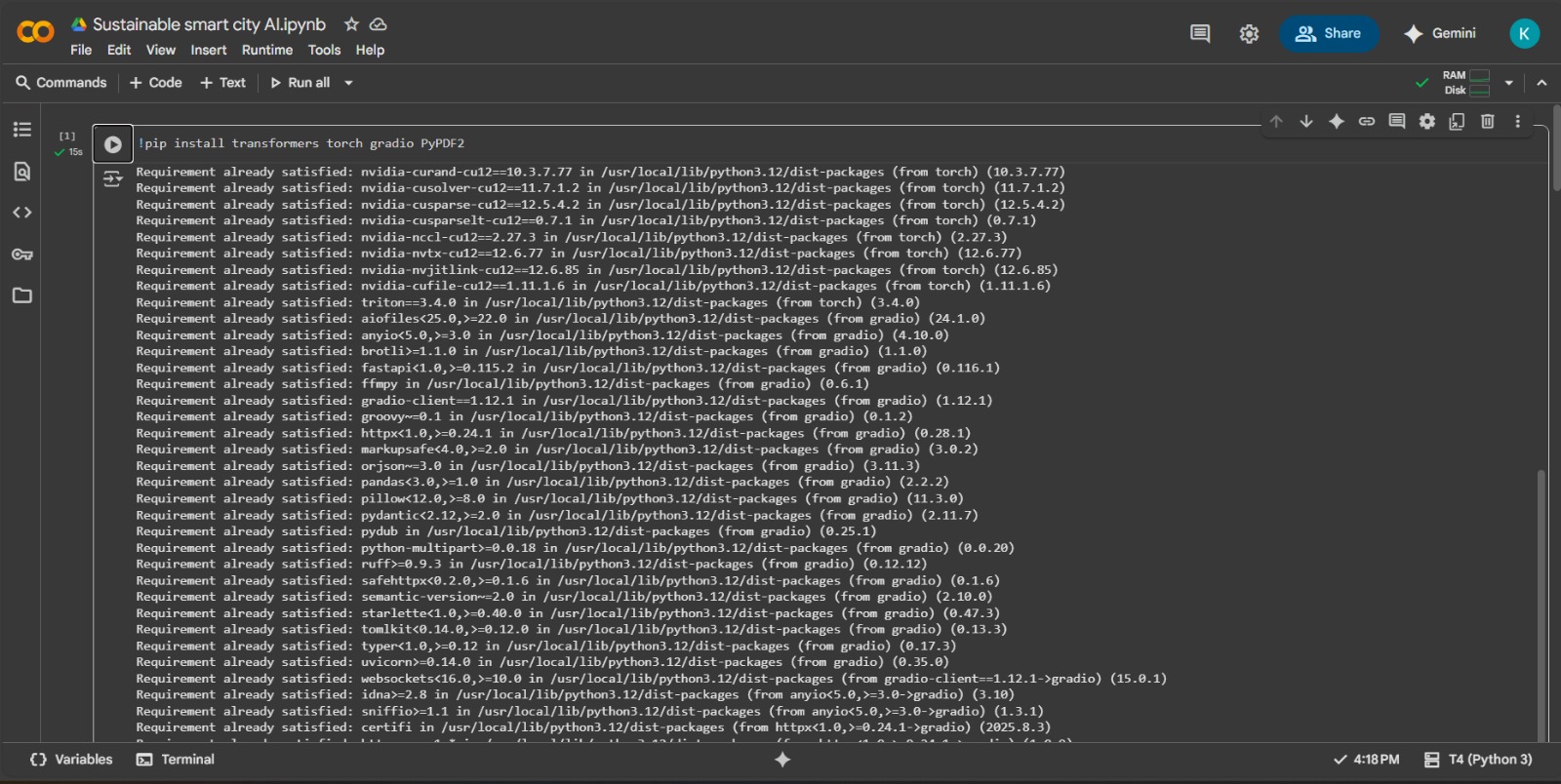
This project emphasizes the importance of citizen participation in creating smarter cities. By generating eco-friendly tips and summarizing government policies, the assistant empowers individuals to take meaningful actions in their daily lives. It bridges the gap between technology and people by providing simple, accessible guidance that contributes to a sustainable future.

While the current system provides strong results, the potential for future enhancements is vast. Expanding the assistant with predictive analytics, IoT integration, and real-time city data would make it even more effective for urban planners and citizens alike. This project lays a foundation for smarter city development, ensuring that innovation continues to support sustainability in the years ahead.

**13. Source Code Implementation**

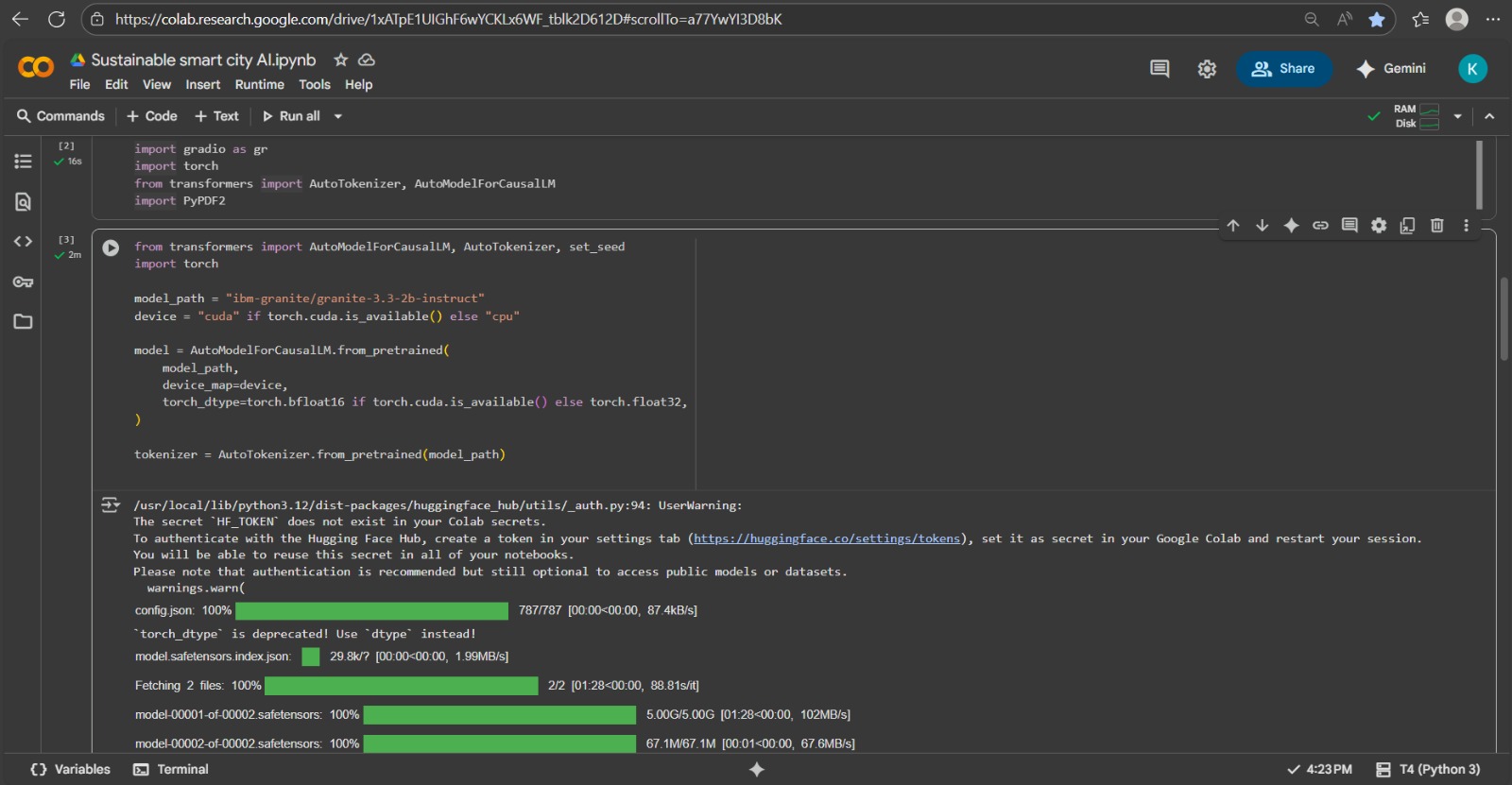
**Step** **1 :**  Install Dependencies.

We install the required Python libraries in Google Colab to run the project.



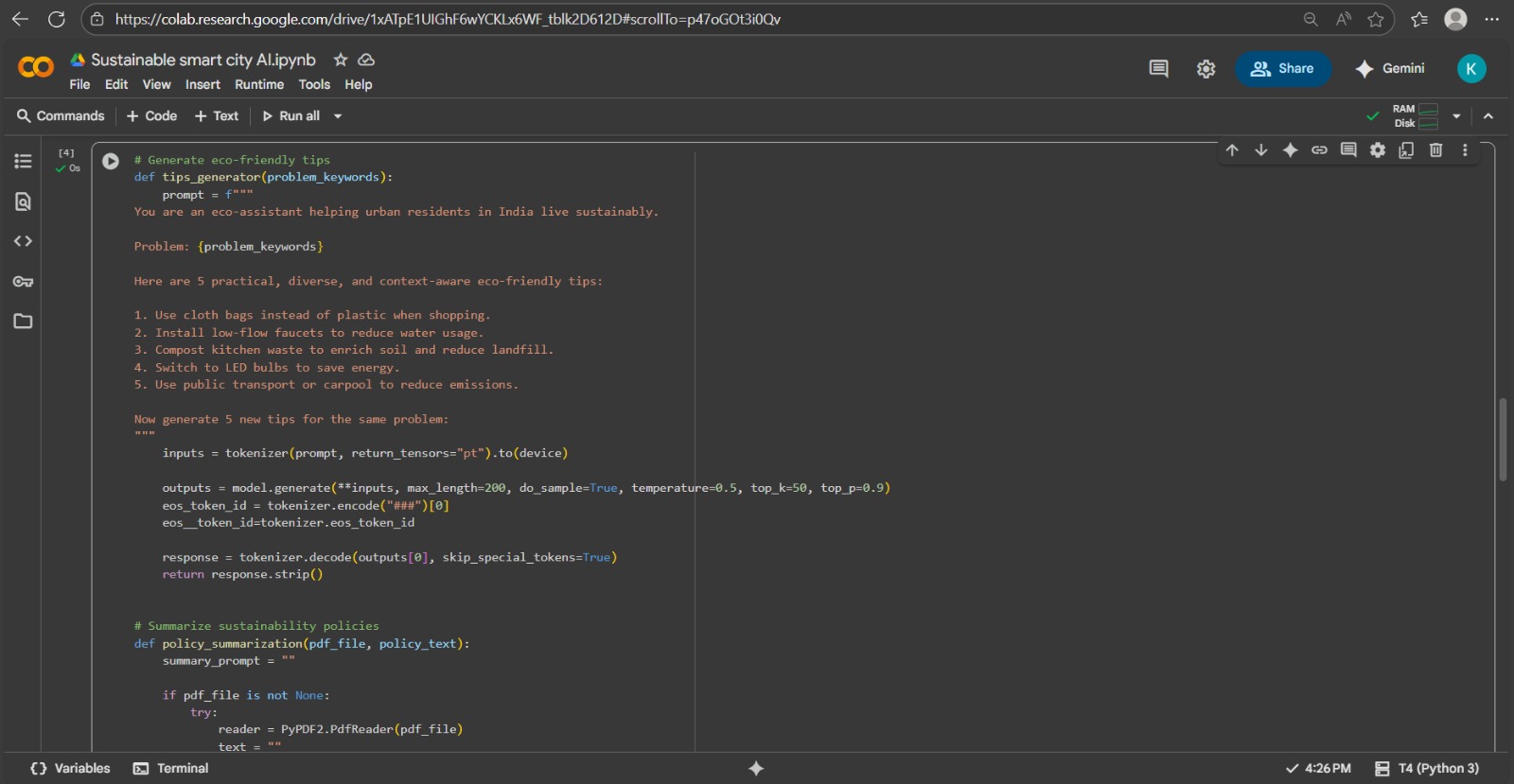
**Step** **2 :** Import Libraries and Configure Device.

We import all necessary libraries and set the device to GPU (CUDA) if available, otherwise CPU.



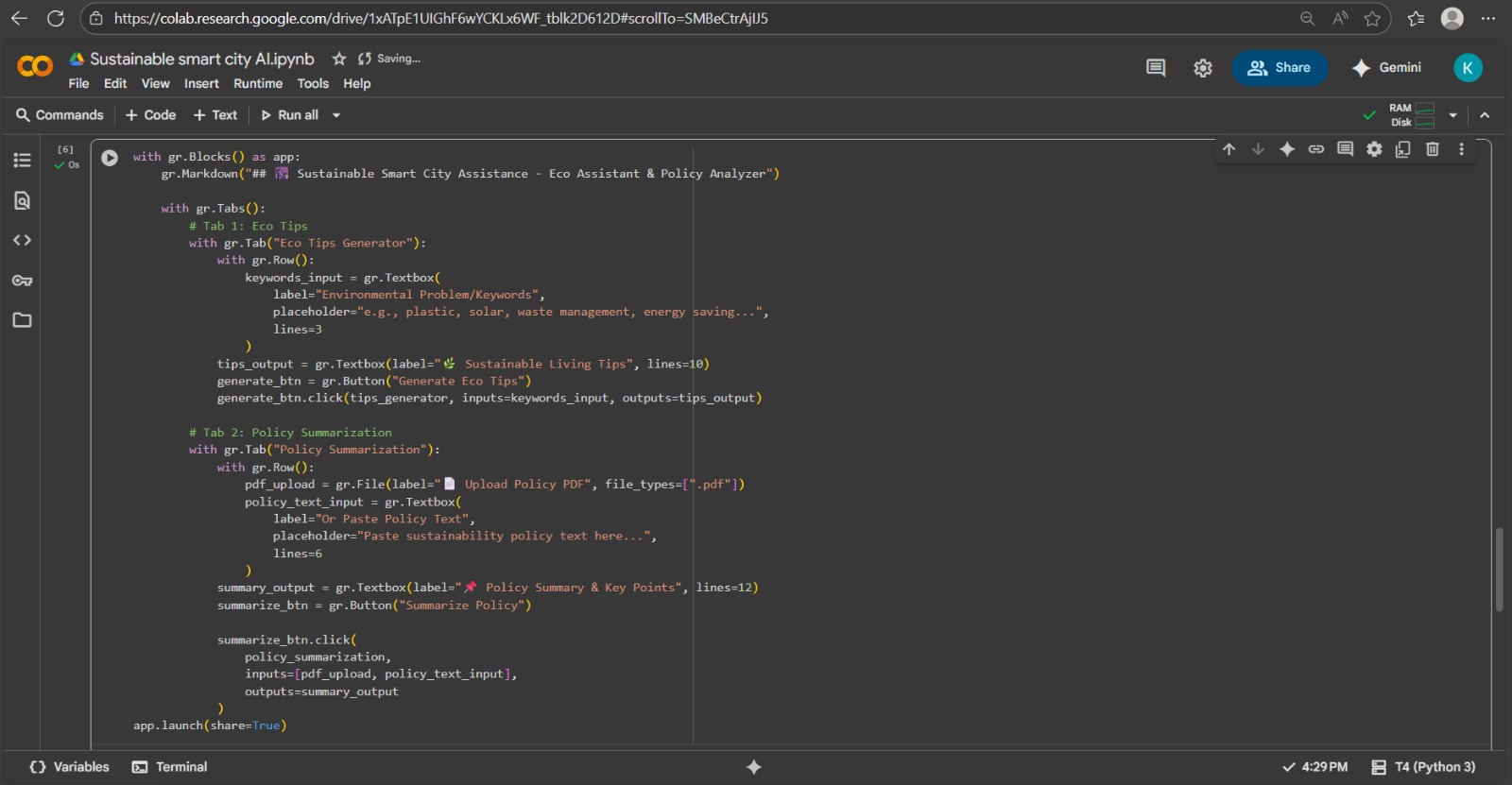
**Step** **3 :** Load Model and Tokenizer.

We load the IBM Granite model and its tokenizer to generate text and summaries.



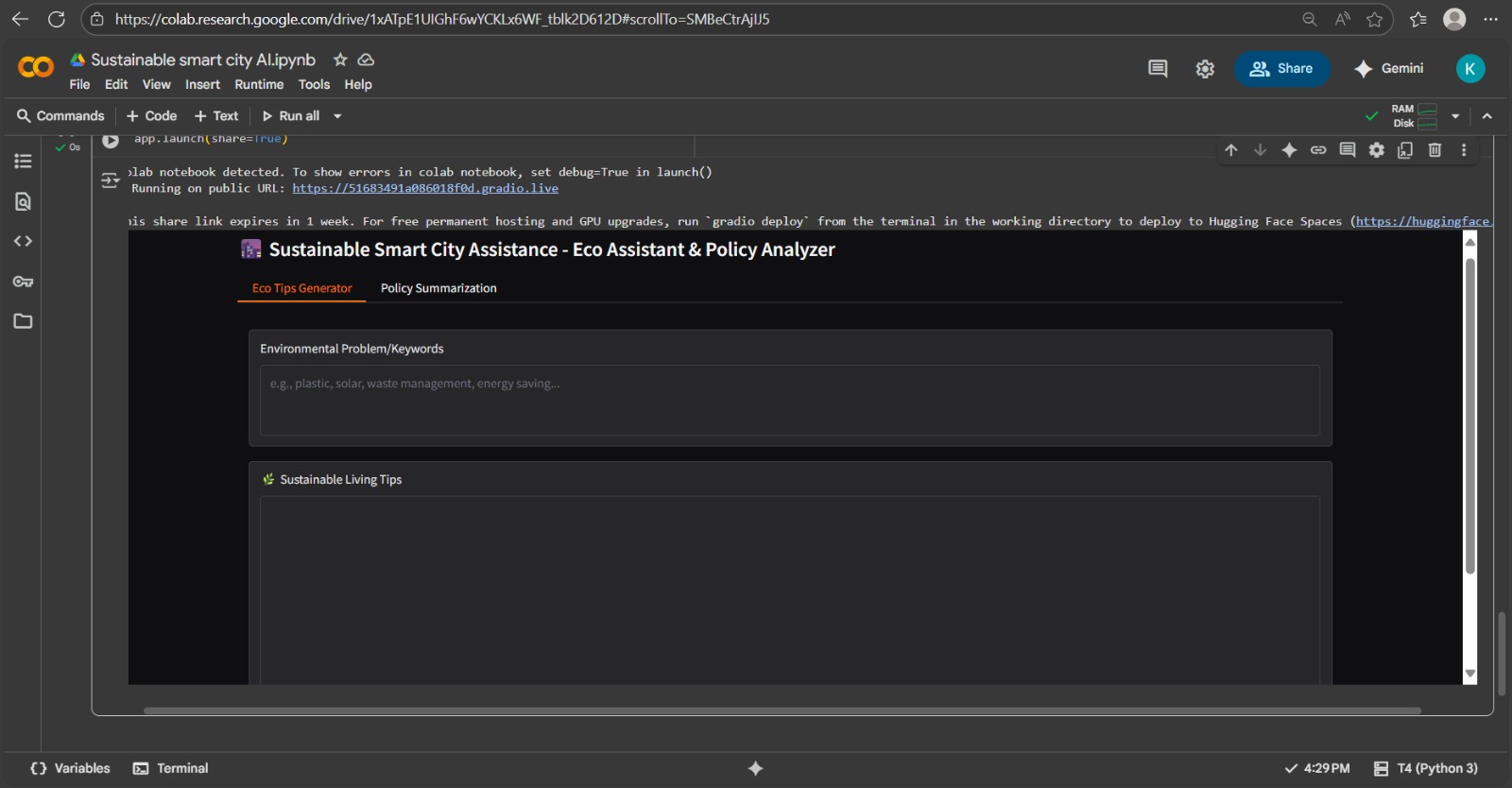
**Step** **4 :** Define Core Functions.

We define two main functions: one to generate eco-friendly tips and another to summarize sustainability policies.



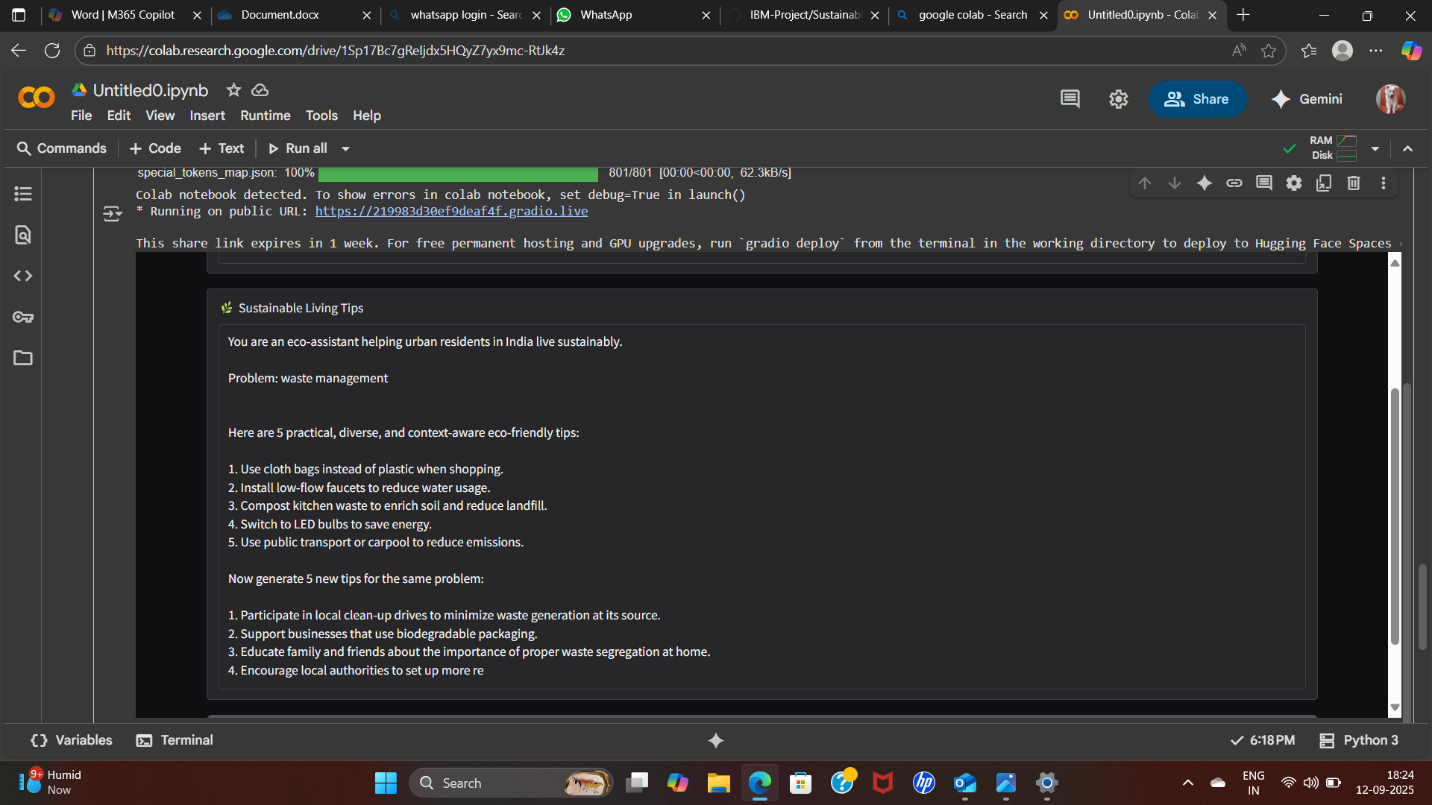
**Step** **5 :** Build and Launch Gradio Interface.

We create a user interface with Gradio where users can generate tips and summarize policies.



**14.** **Output :**

“The Sustainable Smart City Assistant successfully generates eco-friendly tips and summarizes policies through an interactive interface.”



**15.** **Reference :**

This project was developed using multiple open-source tools and research materials. The IBM Granite model served as the foundation for building the AI assistant, supported by the Hugging Face Transformers library and the PyTorch deep learning framework. Gradio was used to design the user interface, while PyPDF2 enabled the extraction and summarization of policy documents. In addition, reference was made to the United Nations Sustainable Development Goals, particularly Goal 11, which emphasizes the importance of creating sustainable cities and communities. Together, these resources provided the technical and conceptual background for implementing the Sustainable Smart City Assistant using IBM.