**ECO ASSISTANT & POLICY ANALYZER**

**Project Documentation**

**1. Introduction**

* **Project Title** : Sustainable Smart City Assistant Using IBM Granite LLM
* **Team Member** :K. Karthika devi
* **Team Member** : T. Rasika
* **Team Member** : S. Omprabha
* **Team Member** : M. Angalaparameshwari

**2. Project Overview**

**Introduction :**

The Eco Assistant & Policy Analyzer is an AI-powered application built using Python, Gradio, and Hugging Face’s Transformers. It provides two main functionalities: generating actionable eco-friendly tips for sustainable living and summarizing lengthy policy documents (PDFs or text input). By combining AI with environmental awareness, this tool helps promote sustainability and policy understanding in an interactive way.

**Purpose :**

Environmental sustainability and policy awareness are two major challenges in today’s society. While people seek eco-friendly practices, they often lack actionable solutions. Similarly, policy documents are usually lengthy and complex, making it difficult for the general public to understand their implications.

This project bridges that gap by using a Large Language Model

1. Provide eco tips based on problem keywords.
2. Summarize policy documents into concise, understandable key points.

The project uses Gradio to create a simple and interactive web-based interface that anyone can use.

**Features :**

1. **Eco Tips Generator**

**Input :** Enter keywords (e.g., *plastic waste, solar energy*)

**Output :** Actionable eco tips

1. **Policy Summarization**

**Input :** Upload a policy PDF or paste the policy text

**Output :** Summarized key provisions & implications

**3. Architecture**

**Frontend :**

1. **Eco Tips Generator Tab :**

* Input textbox for keywords.
* Button to generate eco tips.
* Output textbox displaying AI-generated sustainable living solutions.

1. **Policy Summarization Tab :**

* File uploader for PDFs.
* Textbox for pasting raw policy text.
* Button to summarize the document.
* Output textbox showing summary, key points, and implications.

**Backend :**

1. **Model Initialization :**

* Loads the ibm-granite/granite-3.2-2b-instruct model and tokenizer.
* Optimized for GPU usage if available.

1. **Functions :**

* generate\_response() → Generates text responses from the model.
* extract\_text\_from\_pdf() → Reads and extracts text from PDF documents.
* eco\_tips\_generator() → Creates a prompt for eco tips and returns AI-generated suggestions.
* policy\_summarization() → Summarizes uploaded or pasted policy documents.

**4. Setup Instructions**

**Prerequisites :**

* **Python 3.8 or higher** installed → [Download Python](https://www.python.org/downloads/)
* **pip** (Python package manager) and virtual environment tools
* A working internet connection **GPU with CUDA support**
* GPU with CUDA support

**Installation Process :**

* Install Python (3.10+)
* Create project folder
* Create a virtual environment
* Activate the virtual environment
* Upgrade pip & packaging tools
* Install PyTorch
* Install remaining libraries
* Verify GPU availability
* Run the application
* Deactivate virtual environment when finished

**5. Folder Structure**

* + app.py - Your main code (the one you shared).
  + requirements.txt - Contains all dependencies so others can install
  + README.md - Project overview, usage guide, seminar documentation.
  + /models/ (optional) - If you want to store or cache Hugging Face models locally instead of downloading each time.
  + /data/ - For testing PDFs or sample text inputs.
  + /static/ - If you want to add any icons, logos, or images
  + /docs/ - Store formal documentation or reports in PDF/Word format for your seminar.

**6. Running the Application**

To Start the Project :

* + Python environment and install the required dependencies such as Torch, Transformers, Gradio, and PyPDF2.
  + After preparing the environment, the project is run by executing the main script app.py, which automatically launches a backend powered by the Hugging Face Granite model and a frontend created with Gradio.
  + The backend handles tasks such as text generation, eco-tip suggestions, and policy summarization, while the frontend provides a user-friendly interface accessible through a web browser.
  + Once the application is running, users can open the local URL provided in the terminal, where they can generate eco-friendly tips or upload policy documents for summarization.
  + This integration of backend intelligence and frontend simplicity makes the project easy to use and effective for both environmental awareness and policy analysis.

**Frontend**

The frontend is created using **Gradio**, which automatically launches a simple and interactive web interface in the browser. When users open the local URL, they are presented with two tabs — one for generating eco-friendly tips and another for summarizing policy documents. Users can type keywords, upload a PDF, or paste text directly into the interface. These inputs are sent to the backend for processing, and once the backend generates the results, they are displayed in the frontend textbox for the user to read.

**Backend**

The backend of the **Eco Assistant & Policy Analyzer** is built with Python and powered by libraries such as PyTorch, Transformers, and PyPDF2. When the application starts, the backend loads the **Granite AI model**, which is responsible for generating eco-friendly tips and summarizing policies. It also processes inputs by tokenizing text, extracting data from uploaded PDFs, and creating responses based on user queries. The backend works silently in the background, handling all logic, computation, and AI-driven tasks.

**7. API Documentation**

Backend APIs available include:

* **POST /eco-tips-generator** – Accepts environmental problem keywords and responds with AI-generated sustainable living tips.
* **POST /policy-summarization** – Accepts either a PDF file or plain policy text and returns a summarized version highlighting key provisions, implications, and important points.
* **GET /health-check** – Simple health check endpoint to verify if the server is running.

Each endpoint is tested and can be extended with tools like **Swagger UI** or **Postman** for quick inspection and trial during development.

**8. Authentication**

This version of the project runs in an **open environment** for demonstration, so authentication is not required when running locally.

However, for secure deployments, authentication can be integrated using:

* **Token-based authentication** (API keys or JWT tokens)
* **OAuth2 authentication** (Google, GitHub, or IBM Cloud credentials)
* **Role-based access control** (admin, researcher, citizen)
* **Planned enhancements** such as usersessions, request history tracking, and access logging.

**9. User Interface**

The user interface is built with **Gradio** and is designed to be minimalist and functional, making it easy for both technical and non-technical users. It includes:

* **Tabbed layout** with two sections:
  + Eco Tips Generator *– Generates actionable sustainability tips.*
  + Policy Summarization – Summarizes uploaded policy PDFs or pasted text.
* **Real-time form handling** for instant interaction.
* **File upload support** for PDF policy documents.
* **Textbox outputs** for displaying AI-generated responses**.**
* **Cross-platform accessibility** via a web browser without additional installation.

**10. Testing**

Testing was done in multiple phases:

**Unit Testing :** Ensure that each individual function works correctly and returns expected outputs.

**Integration Testing** : That **functions work together as expected** in workflows.

**Interface Testing :** Ensure the user interface behaves correctly and is user-friendly.

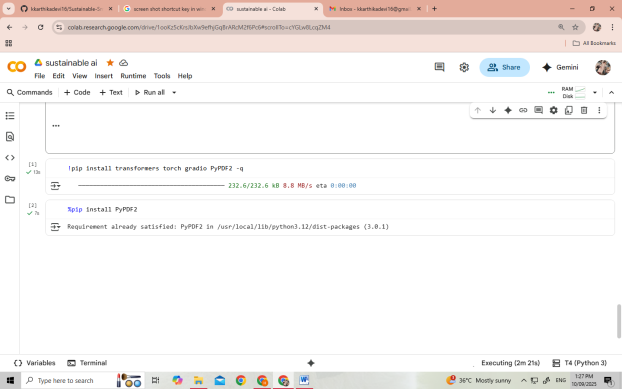
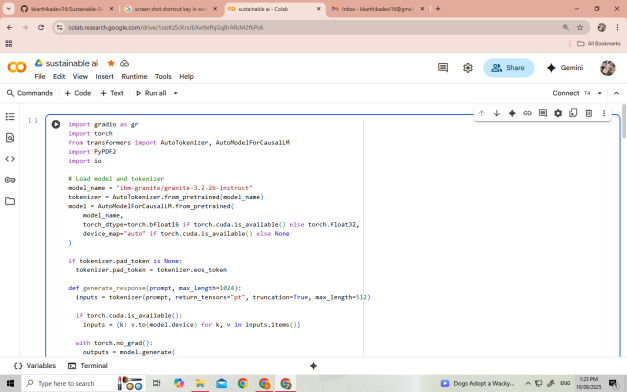
**Edge Case Testing :** Verify robustness against unusual or extreme inputs.

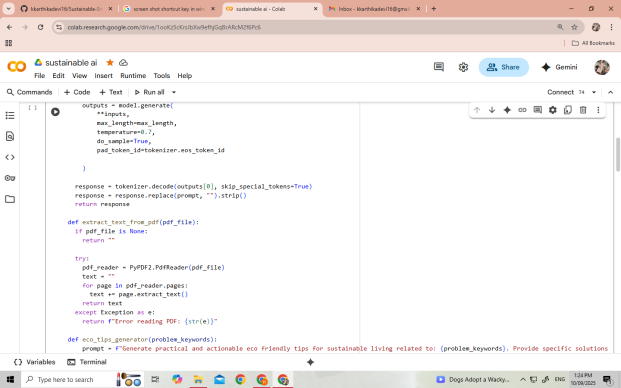
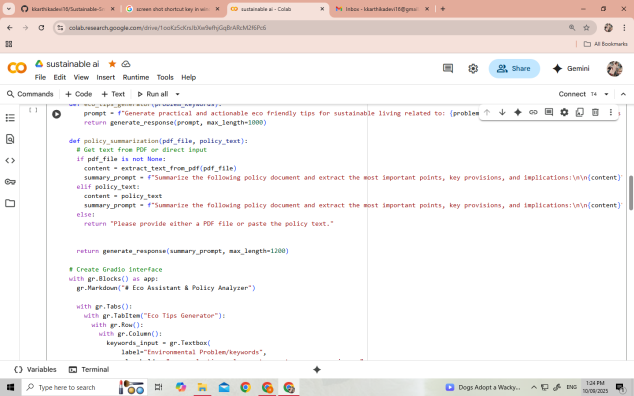
**Performance & Stress Testing :** Assess how the application performs under load or large inputs.

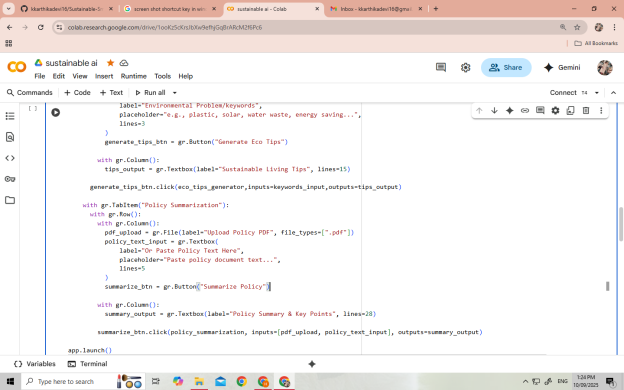
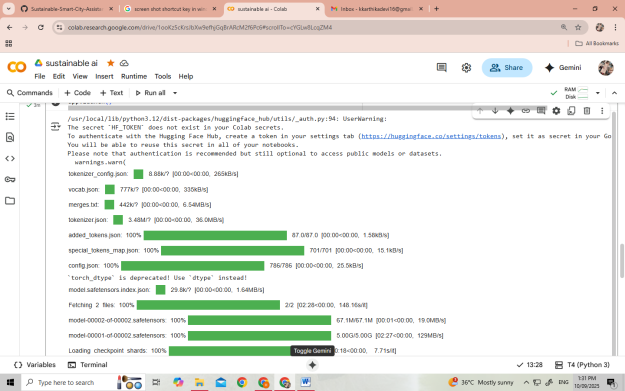
**Usability & User Acceptance Testing :** Evaluate user experience and clarity.

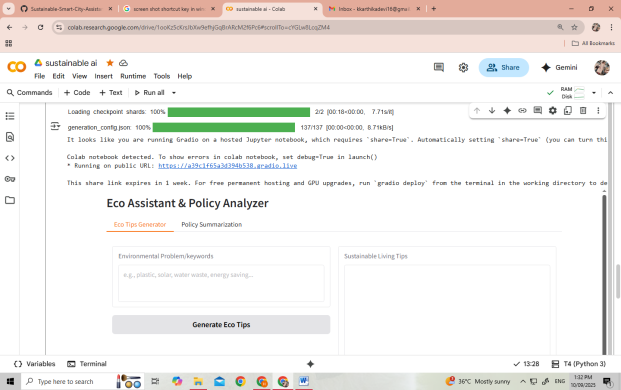
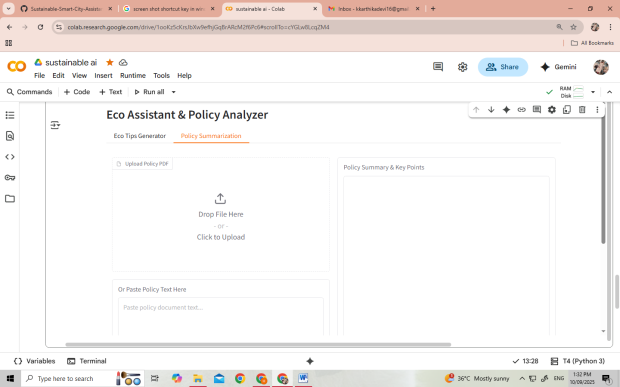
**Regression Testing :** Ensure new changes or updates do not break existing functionality.

**11. Screen Shots**

**12. Known issues**

**1. PDF Extraction Limitations**

* **Scanned PDFs or Image-based PDFs :** PyPDF2 cannot extract text from scanned documents or PDFs containing only images. Users may receive empty output or incomplete summaries.
* **Complex PDF Layouts :** Tables, footnotes, headers, and multi-column layouts may not be parsed correctly. Text order can be scrambled, affecting the summary quality.
* **Large PDFs :** PDFs exceeding 50–100 pages may take a long time to process or cause memory issues. Summaries may truncate due to max\_length restrictions in the language model.

**2. Model Limitations**

* **Token Length Constraints :** The model has a maximum token limit. Very long prompts or PDFs may result in truncated responses. Important points could be missed in the output if content exceeds token limits.
* **Accuracy of Summaries :** AI-generated summaries may occasionally misinterpret policy nuances or miss subtle provisions. Fact-checking is recommended for critical policy documents.
* **Language Support :** While the model supports English primarily, other languages may generate poor or irrelevant responses. Special characters or uncommon symbols may produce unexpected output.
* **Determinism :** temperature=0.7 introduces randomness in responses. The same prompt may yield slightly different outputs each time.

**3. Interface / UX Issues**

* **File Upload Validation :** Non-PDF uploads are handled with a generic error, but there is no pre-validation before file processing. Users may be confused if they upload unsupported file types.
* **Empty Inputs :** If both PDF and text inputs are empty, a message appears, but there is no interactive prompt guiding users to correct it.
* **Output Formatting :** Long outputs in the textboxes may become hard to read without scrolling. No pagination or collapsible sections for very large summaries.

**4. Performance Issues**

* **GPU vs CPU :** On machines without GPU, model inference is slower, which may result in delays for long prompts or PDFs.
* **Multiple Requests / Concurrency :** Gradio in local mode is single-threaded; simultaneous requests may slow down the app. High concurrency is not fully supported in local deployment.

**5. Edge Case Behavior**

* **Extreme Inputs :** Extremely long strings (thousands of characters) may cause truncation or slow response times. Empty strings, special characters, or unsupported languages may produce unexpected or empty outputs.
* **Policy Context Misinterpretation :** Summaries may miss critical legal or technical terms if the policy contains complex jargon.

**6. Other Known Limitations**

* **No Persistence :** Generated tips or summaries are not saved; refreshing the app loses all data.
* **No Multi-File Handling :** Only one PDF can be uploaded at a time; batch summarization is not supported.
* **No Offline PDF OCR :** Image-based PDFs cannot be summarized unless pre-processed with an OCR tool.

**13. Future Enhancement**

**1. PDF Handling & Text Extraction**

* **OCR Integration :** Support scanned or image-based PDFs using OCR (e.g., Tesseract, EasyOCR) to extract text.
* **Improved Layout Parsing :** Handle tables, columns, headers, and footnotes correctly to improve summarization quality.
* **Batch PDF Uploads :** Allow multiple PDFs to be uploaded and summarized in a single session.
* **Large PDF Support :** Implement chunking to process long PDFs in segments without hitting token limits.

**2. Model & AI Improvements**

* **Multi-Language Support :** Enhance AI capabilities for non-English input and generate tips or summaries in multiple languages.
* **Customizable Output :** Allow users to select summary length, detail level, or tone (formal/informal).
* **Domain-Specific Fine-Tuning :** Fine-tune the model for environmental and policy-specific contexts for more accurate outputs.
* **Context Retention :** Support multi-turn conversation for eco tips to remember previous queries and provide consistent advice.

**3. Interface & UX Enhancements**

* **Rich Output Display :** Add collapsible sections, bullet points, or tables for better readability of summaries.
* **Interactive Guidance :** Provide real-time suggestions if input is missing or invalid.
* **Download Option :** Allow users to download summaries and tips as PDF or Word documents.
* **Mobile-Friendly Design :** Optimize Gradio interface for mobile devices.

**4. Performance & Scalability**

* **Asynchronous Processing :** Improve response times for large documents using async processing or background tasks.
* **GPU / Cloud Deployment :** Deploy on cloud services with GPU support for faster inference.
* **Concurrency Handling :** Enable multi-user support for simultaneous requests without slowing down the application.

**5. Additional Features**

* **Eco Tips Personalization :** Tailor tips based on user location, lifestyle, or resources available.
* **Policy Comparison :** Compare multiple policy documents and highlight differences or conflicts.
* **Historical Data Tracking :** Maintain a record of past inputs and summaries for user reference.
* **Integration with External APIs :** Pull real-time environmental data or policy updates for more relevant recommendations.

**6. Analytics & Reporting**

* **Usage Statistics :** Track most common keywords or policies analyzed.
* **Effectiveness Metrics :** Allow users to rate generated tips and summaries to improve AI accuracy over time.
* **Visualization :** Generate charts or infographics summarizing eco-friendly tips or policy insights.