# SUSTAINABLE SMART CITY ASSISTANT USING IBM GRANITE LLM

|  |  |
| --- | --- |
| DATE |  |
| TEAM ID | NM2025TMID00605 |
| PROJECT NAME | SUSTAINABLE SMART CITY ASSISTANT USING IBM GRANITE LLM |
| MAXIMUM MARKS |  |

## Introduction

The Sustainable Smart City Assistant is a state-of-the-art AI-driven platform designed to help communities, organizations, and government bodies create a greener, smarter, and more sustainable urban environment. This assistant combines advanced Natural Language Processing (NLP) from IBM Granite LLM to provide eco-friendly tips, policy analysis, and actionable solutions that empower users to take sustainable actions. Its goal is to bridge the gap between smart city initiatives and environmental awareness, making sustainability practices easy to understand and implement for citizens.

## Objectives

1. Promote eco-conscious living by delivering actionable sustainability tips.  
2. Simplify complex government policies and urban planning documents into concise summaries.  
3. Empower communities and policy makers with AI-driven insights.  
4. Showcase how AI can play a transformative role in building greener cities.

## Key Features

* Eco Tips Generator: Provides actionable solutions for environmental issues like waste management, solar energy adoption, and water conservation.
* Policy Summarization: Summarizes uploaded policy PDFs or text, extracting major points, clauses, and actionable insights.
* PDF Extraction: Reads and processes PDF documents for easy text retrieval and AI-based analysis.
* Interactive Interface: Uses Gradio for a visually appealing, easy-to-use interface.
* AI-Powered Insights: Leverages IBM Granite LLM, a state-of-the-art AI model, for text comprehension and response generation.

## Disclaimer

This tool is meant for educational and informational purposes only. While it helps users interpret policies and gain eco-friendly knowledge, official guidelines and professional consultation should always be prioritized for decision-making.

## Getting Started

Setting up the Sustainable Smart City Assistant is simple. The application is coded in Python and integrates popular libraries such as Hugging Face Transformers, PyTorch, Gradio, and PyPDF2. The assistant can run on local machines or cloud platforms.

### Prerequisites

- Python 3.8 or higher  
- PyTorch (with GPU support recommended)  
- Hugging Face Transformers library  
- Gradio for web-based interface  
- PyPDF2 for PDF extraction

### Installation Steps

Run the following command to install dependencies:  
  
pip install torch gradio transformers PyPDF2

## Project Code Explanation

The following section provides the full source code for the Sustainable Smart City Assistant. It includes functions for text generation, PDF reading, eco-tip generation, and policy summarization. Additionally, each function's purpose, input, and output are described.

# -\*- coding: utf-8 -\*-  
"""Smartcity Assistant"""  
  
import gradio as gr  
import torch  
from transformers import AutoTokenizer, AutoModelForCausalLM  
import PyPDF2

The program starts by importing essential libraries:   
- `gradio` for creating the web-based interface.  
- `torch` for PyTorch deep learning functionalities.  
- `transformers` for loading IBM Granite LLM.  
- `PyPDF2` for extracting text from uploaded PDF documents.

### Model Loading

The code loads IBM Granite LLM to generate AI responses. It intelligently detects GPU availability and optimizes performance.

model\_name = "ibm-granite/granite-3.2-2b-instruct"  
tokenizer = AutoTokenizer.from\_pretrained(model\_name)  
model = AutoModelForCausalLM.from\_pretrained(model\_name, ... )

### Response Generation

The `generate\_response()` function takes a text prompt and generates a contextual response. Parameters like `temperature` control the creativity of the responses, while `max\_length` restricts output length.  
Input: A string prompt.  
Output: An AI-generated string.

### PDF Extraction

The `extract\_text\_from\_pdf()` function reads text from PDF pages. It handles errors gracefully, ensuring that corrupted or invalid files don't crash the system.  
Input: PDF file.  
Output: Extracted text.

### Eco Tips Generator

This feature generates practical eco-friendly tips. Example:  
Input: 'plastic waste reduction'  
Output: 'Consider reducing single-use plastics, switching to biodegradable packaging, and promoting recycling drives.'

### Policy Summarization

This feature summarizes lengthy policies, highlighting important sections.  
Input: Urban development policy document.  
Output: Bullet points summarizing policies and actionable clauses.

### Gradio Interface

The Gradio interface contains two main tabs:  
- \*\*Eco Tips Generator Tab\*\*: Enter keywords and receive eco-friendly suggestions.  
- \*\*Policy Summarization Tab\*\*: Upload a PDF or paste policy text for AI-generated summaries.

The final application is launched using `app.launch(share=True)`, allowing public access through a shareable link.

### Coding

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,

max\_length=max\_length,

temperature=0.7,

do\_sample=True,

pad\_token\_id=tokenizer.eos\_token\_id

)

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)

# Create Gradio Interface

with gr.Blocks() as app:

gr.Markdown("Eco Assistant & Policy Analyzer")

with gr.Tabs():

with gr.TabItem("Eco Tips Generator"):

with gr.Row():

with gr.Column():

keywords\_input = gr.Textbox(

label="Environmental Problem/Keywords",

placeholder="e.g., plastic, solar, water waste, energy saving..."

)

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)

## Conclusion

The Sustainable Smart City Assistant represents the fusion of AI and sustainability, empowering users with actionable insights. It showcases the versatility of IBM Granite LLM in solving real-world challenges, particularly in environmental awareness and urban policy comprehension. By bridging the gap between technology and sustainability, this tool inspires citizens and organizations to adopt smart, eco-friendly practices. Future iterations can include features like live IoT data integration, predictive analytics for environmental risks, and personalized sustainability plans.