Generative AI with IBM

Project title : **SUSTAINABLE SMART CITY ASSISTANT USING IBM GRANITE LLM**

Team member : Nithyasri S
Team member : Vijaya Lakshmi M
Team member : Priya S
Team member : Abinaya M

Project Overview

Purpose :

The purpose of a Sustainable Smart City Assistant is to empower cities and their residents to thrive in a more eco-conscious and connected urban environment. By leveraging AI and real-time data, the assistant helps optimize essential resources like energy, water, and waste, while also guiding sustainable behaviors among citizens through personalized tips and services. For city officials, it serves as a decision- making partner—offering clear insights, forecasting tools, and summarizations of complex policies to support strategic planning. Ultimately, this assistant bridges technology, governance, and community engagement to foster greener cities that are more efficient, inclusive, and resilient.

Features:

Conversational Interface

Key Point: Natural language interaction

Functionality: Allows citizens and officials to ask questions, get updates, and receive guidance in plain language

Policy Summarization

Key Point: Simplified policy understanding

Functionality: Converts lengthy government documents into concise, actionable summaries.

Resource Forecasting
Key Point: Predictive analytics
Functionality: Estimates future energy, water, and waste usage using historical and real-time data.

Eco-Tip Generator
Key Point: Personalized sustainability advice
Functionality: Recommends daily actions to reduce environmental impact based on user behavior.

Citizen Feedback Loop
Key Point: Community engagement
Functionality: Collects and analyzes public input to inform city planning and service improvements.

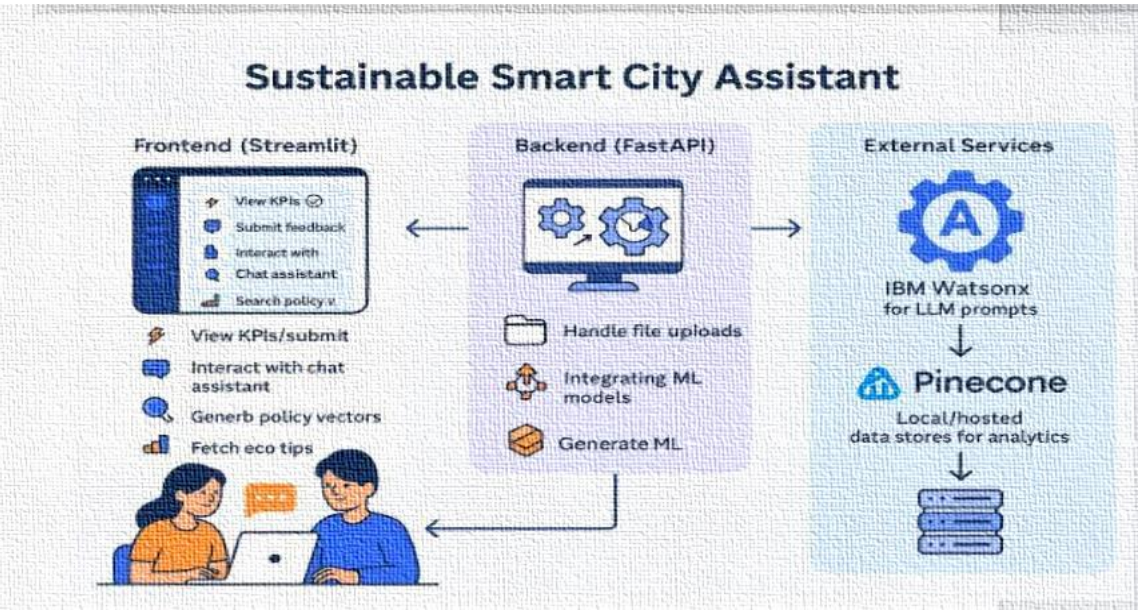
KPI Forecasting
Key Point: Strategic planning support
Functionality: Projects key performance indicators to help officials track progress and plan ahead.

Anomaly Detection
Key Point: Early warning system
Functionality: Identifies unusual patterns in sensor or usage data to flag potential issues.

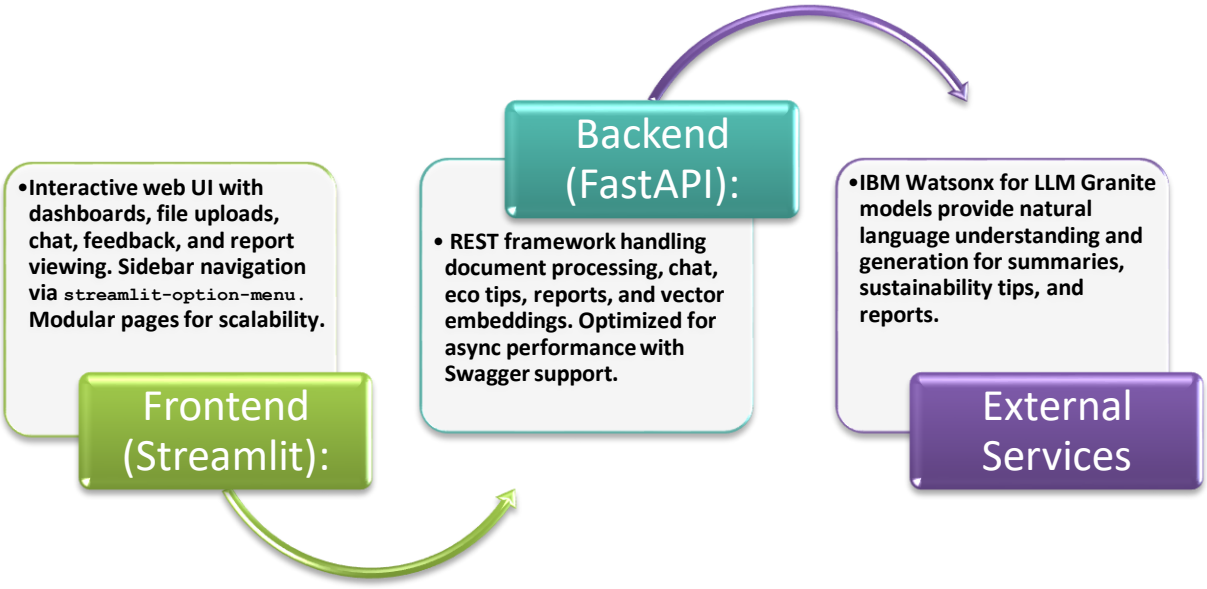
Multimodal Input Support
Key Point: Flexible data handling
Functionality: Accepts text, PDFs, and CSVs for document analysis and forecasting.

Streamlit or Gradio UI
Key Point: User-friendly interface
Functionality: Provides an intuitive dashboard for both citizens and city officials to interact with the assistant.

Architecture



The Architecture Consists Of Three Main Layers:



Vector Search (Pinecone):

Uploaded policy documents are embedded using Sentence Transformers and stored in Pinecone. Semantic search is implemented using cosine similarity to allow users to search documents using natural language queries.

ML Modules (Forecasting and Anomaly Detection):

Lightweight ML models are used for forecasting and anomaly detection using Scikit-learn. Time-series data is parsed, modeled, and visualized using pandas and matplotlib.

Setup Instructions

Prerequisites:

- Python 3.9 or later
- pip and virtual environment tools
- API keys for IBM Watsonx and Pinecone
- Internet access to access cloud services

Installation Process:

- Clone the repository
- Install dependencies from requirements.txt
- Create a .env file and configure credentials
- Run the backend server using Fast API
- Launch the frontend via Stream lit
- Upload data and interact with the modules

Project Milestones

- **Phase 1 – Initialization:** Set up modular folder structure, environment configs, and Pinecone vector index.
- **Phase 2 – Watsonx Integration:** Configured API keys, models, and validated endpoints via Swagger.
- **Phase 3 – Backend APIs:** Built modular routers (chat, feedback, eco tips, KPIs, vectors) with robust testing.
- **Phase 4 – Frontend UI:** Designed Streamlit dashboard with modular components, navigation, and styled UI.
- **Phase 5 – Pinecone & Embeddings:** Implemented document embedding and retrieval using sentence-transformers.
- **Phase 6 – Reports & Deployment:** Granite LLM-powered report generation, PDF/Markdown support, and full feature integration testing.

1. Folder Structure

app/ – Contains all Fast API backend logic including routers, models, and integration modules.

app/api/ – Subdirectory for modular API routes like chat, feedback, report, and document vectorization.

ui/ – Contains frontend components for Stream lit pages, card layouts, and form UIs.

smart_dashboard.py – Entry script for launching the main Stream lit dashboard.

granite_llm.py – Handles all communication with IBM Watsonx Granite model including summarization and chat.

document_embedder.py – Converts documents to embeddings and stores in Pinecone.

kpi_file_forecaster.py – Forecasts future energy/water trends using regression.

anomaly_file_checker.py – Flags unusual values in uploaded KPI data.

report_generator.py – Constructs AI-generated sustainability reports.

2. Running the Application

To start the project:

- Launch the FastAPI server to expose backend endpoints.
- Run the Streamlit dashboard to access the web interface.
- Navigate through pages via the sidebar.
- Upload documents or CSVs, interact with the chat assistant, and view outputs like reports, summaries, and predictions.
- All interactions are real-time and use backend APIs to dynamically update the frontend.

3. API Documentation

Backend APIs available include:

POST /chat/ask – Accepts a user query and responds with an AI-generated message

POST /upload-doc – Uploads and embeds documents in Pinecone

GET /search-docs – Returns semantically similar policies to the input query

GET /get-eco-tips – Provides sustainability tips for selected topics like energy, water, or waste

POST /submit-feedback – Stores citizen feedback for later review or analytics

Each endpoint is tested and documented in Swagger UI for quick inspection and trial during development.

Authentication

Implemented endpoint testing and documentation via Swagger UI for easy validation during development. The demo runs in an open environment for accessibility, while production deployments can integrate authentication and access controls to ensure security.

- ✓ Token-based authentication (JWT or API keys)
- ✓ OAuth2 with IBM Cloud credentials
- ✓ Role-based access (admin, citizen, researcher)
- ✓ Planned enhancements include user sessions and history tracking.

User Interface

The interface is minimalist and functional, focusing on accessibility for non-technical users. It includes:

- Sidebar with navigation
- KPI visualizations with summary cards
- Tabbed layouts for chat, eco tips, and forecasting
- Real-time form handling
- PDF report download capability

The design prioritizes clarity, speed, and user guidance with help texts and intuitive flows.

Testing

Testing was done in multiple phases:

Unit Testing: For prompt engineering functions and utility scripts

API Testing: Via Swagger UI, Postman, and test scripts

Manual Testing: For file uploads, chat responses, and output consistency

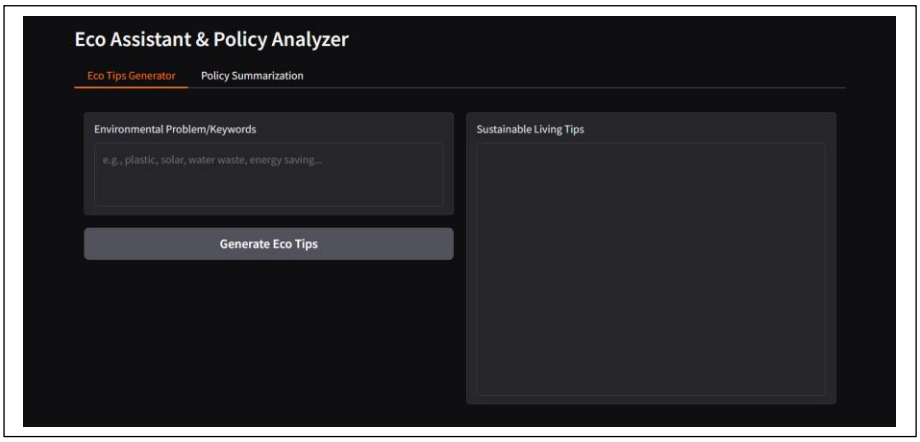
Edge Case Handling: Malformed inputs, large files, invalid API keys

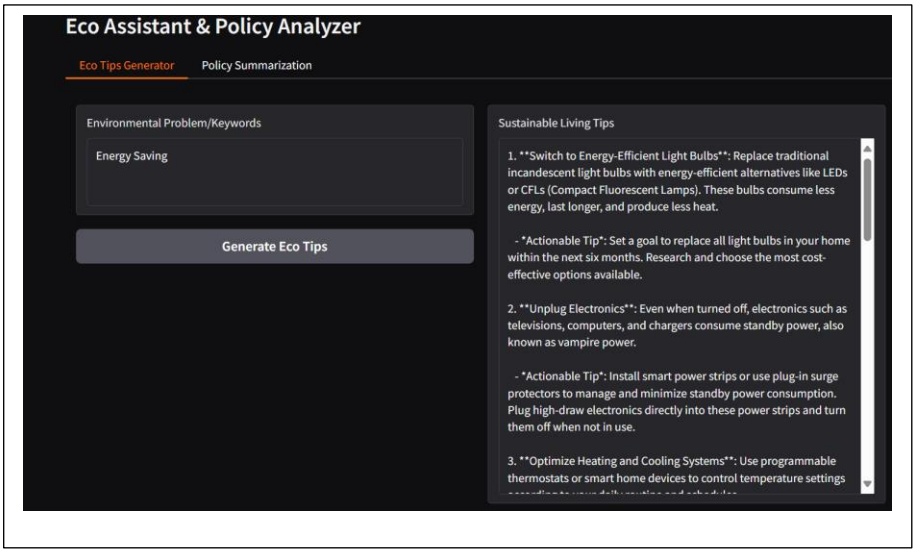
Each function was validated to ensure reliability in both ofline and API-connected modes.

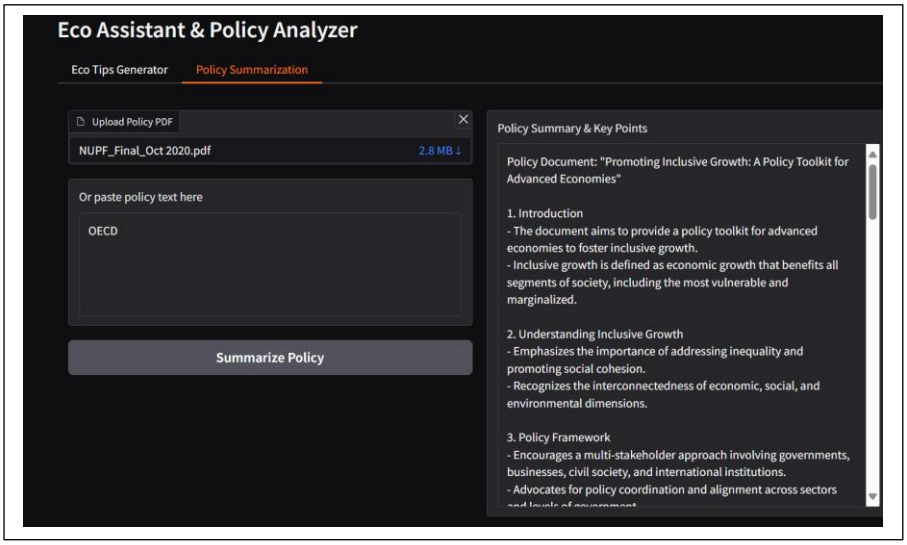
CODE

```
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...",
                        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)
```

OUTPUT


Eco Assistant & Policy Analyzer
Eco Tips Generator
Policy Summarization
Environmental Problem/Keywords
e.g., plastic, solar, water waste, energy saving...
Generate Eco Tips
Sustainable Living Tips


Eco Assistant & Policy Analyzer
Eco Tips Generator
Policy Summarization
Environmental Problem/Keywords
Energy Saving
Generate Eco Tips
Sustainable Living Tips
1. **Switch to Energy-Efficient Light Bulbs**: Replace traditional incandescent light bulbs with energy-efficient alternatives like LEDs or CFLs (Compact Fluorescent Lamps). These bulbs consume less energy, last longer, and produce less heat.
- *Actionable Tip*: Set a goal to replace all light bulbs in your home within the next six months. Research and choose the most cost-effective options available.
2. **Unplug Electronics**: Even when turned off, electronics such as televisions, computers, and chargers consume standby power, also known as vampire power.
- *Actionable Tip*: Install smart power strips or use plug-in surge protectors to manage and minimize standby power consumption. Plug high-draw electronics directly into these power strips and turn them off when not in use.
3. **Optimize Heating and Cooling Systems**: Use programmable thermostats or smart home devices to control temperature settings


Eco Assistant & Policy Analyzer
Eco Tips Generator
Policy Summarization
Upload Policy PDF
NUPF_Final_Oct 2020.pdf
2.8 MB
Or paste policy text here
OECD
Summarize Policy
Policy Summary & Key Points
Policy Document: "Promoting Inclusive Growth: A Policy Toolkit for Advanced Economies"
1. Introduction
- The document aims to provide a policy toolkit for advanced economies to foster inclusive growth.
- Inclusive growth is defined as economic growth that benefits all segments of society, including the most vulnerable and marginalized.
2. Understanding Inclusive Growth
- Emphasizes the importance of addressing inequality and promoting social cohesion.
- Recognizes the interconnectedness of economic, social, and environmental dimensions.
3. Policy Framework
- Encourages a multi-stakeholder approach involving governments, businesses, civil society, and international institutions.
- Advocates for policy coordination and alignment across sectors

Future enhancement

Functionality Enhancements

1. **Multi-language Support**
   - Enable summarization and eco tips in multiple languages using translation models.
2. **Advanced Summarization**
   - Offer both short and detailed summaries, highlight key provisions in bullet points, and extract named entities (e.g., organizations, dates, locations).
3. **Contextual Eco Tips**
   - Tailor tips based on region, climate, or sector (e.g., urban vs. rural, energy vs. water).
4. **Report Generation**
   - Export eco tips and summaries as **PDF/Word reports** with formatting, charts, and visuals.

Integration Enhancements

5. **Database/Vector Store Integration**
   - Store policies in **Pinecone, FAISS, or Weaviate** for semantic search and retrieval.
6. **Watsonx Granite LLM API**
   - Instead of a local Hugging Face model, connect to **IBM Watsonx Granite LLM service** for scalable and more accurate outputs.
7. **External APIs**
   - Link with real-time sustainability datasets (carbon footprint APIs, energy usage stats, etc.) to give data-driven eco tips.

UI/UX Improvements

8. **Interactive Dashboard**
   - Add charts/graphs (e.g., matplotlib, Plotly) for visual policy analysis.
   - Provide comparison view between two policy documents.
9. **Authentication & Roles**
   - Add secure login with user roles (e.g., citizens, policymakers, researchers).
10. **Mobile-Friendly UI**
    - Optimize Gradio or migrate to **Streamlit/React-based dashboards** for better responsiveness.

Deployment Enhancements

11. **Cloud Deployment**
    - Deploy on **AWS/GCP/Azure** with GPU support for scalability.
12. **CI/CD Pipeline**
    - Add automated testing, linting, and containerization (Docker/Kubernetes).
13. **Offline Mode**
    - Allow local inference for small documents when internet is not available.