Sustainable Smart City Assistant Using IBM Granite LLM

Category: Generative AI with IBM

Skills Required: Machine Learning, IBM Cloud, IBM Watson

1. Introduction

Project Title: Sustainable Smart City Assistant Using IBM Granite LLM

Purpose: To empower cities and residents with AI-powered tools for urban sustainability, governance, and citizen engagement.

Team Members: DHARSHINI K

Team Members: MAHASRI R

Team Members: EBIMAL MARAGATHAM M

Team Members: SAINTHAVI E

2. Project Overview

The Sustainable Smart City Assistant is an AI-powered platform leveraging IBM Watsonx Granite LLM and modern data pipelines to enhance urban sustainability and citizen engagement. It integrates features like policy summarization, KPI forecasting, eco-tips, anomaly detection, and a conversational chat assistant through a modular FastAPI backend and Streamlit/Gradio dashboard.

3. Features

• Conversational Interface: Natural language interaction for citizens and officials.

• Policy Summarization: Converts lengthy policy documents into concise summaries.

• Resource Forecasting: Predicts energy, water, and waste usage trends.

• Eco-Tip Generator: Recommends eco-friendly lifestyle practices.

• Citizen Feedback Loop: Collects and processes citizen feedback.

• KPI Forecasting: Projects future consumption trends and KPIs.

• Anomaly Detection: Detects unusual data patterns and issues.

• Multimodal Input Support: Accepts text, PDFs, and CSVs for analysis.

• Streamlit/Gradio UI: Interactive dashboard for officials and citizens.

4. Architecture

The platform uses a modular architecture with the following components:• Frontend (Streamlit / Gradio): Interactive dashboards, chat, feedback forms, and report viewers.• Backend (FastAPI): High-performance REST APIs to handle data and LLM requests.• LLM Integration (IBM Watsonx Granite): Core natural language understanding and generation.• Vector Search (Pinecone): Enables semantic search across documents.• ML Modules: Forecasting and anomaly detection using Scikit-learn and Pandas.

5. Use Case Scenarios

• Policy Search & Summarization: Municipal planners can upload policy documents and instantly receive concise, citizen-friendly summaries.

• Citizen Feedback Reporting: Residents report issues like water leaks via the assistant, which logs them with category tagging.

• KPI Forecasting: Administrators upload KPI data, and the assistant forecasts future resource usage.

• Eco Advice & Sustainability: Generates personalized eco-friendly tips to encourage sustainable living.

6. Setup Instructions

Prerequisites:• Python 3.9 or later• IBM Watsonx and Pinecone API keys• Internet accessSteps to Run:1. Clone the repository and install dependencies.2. Create a .env file and configure credentials.3. Launch FastAPI backend and Streamlit/Gradio frontend.4. Upload data and interact with the assistant modules.

7. API Documentation

Available API Endpoints:• POST /chat/ask → Responds with AI-generated answers.• POST /upload-doc → Upload and embed documents.• GET /search-docs → Semantic policy search.• GET /get-eco-tips → Provides sustainability tips.• POST /submit-feedback → Collects citizen feedback.

8. Authentication & Security

The demo version is open, but secure deployments can integrate:• Token-based authentication (JWT/API Keys)• OAuth2 with IBM Cloud• Role-based access (Admin, Citizen, Researcher)

9. User Interface Design

• Sidebar navigation• KPI visualizations and summaries• Tabs for chat, eco tips, and forecasting• PDF report downloads

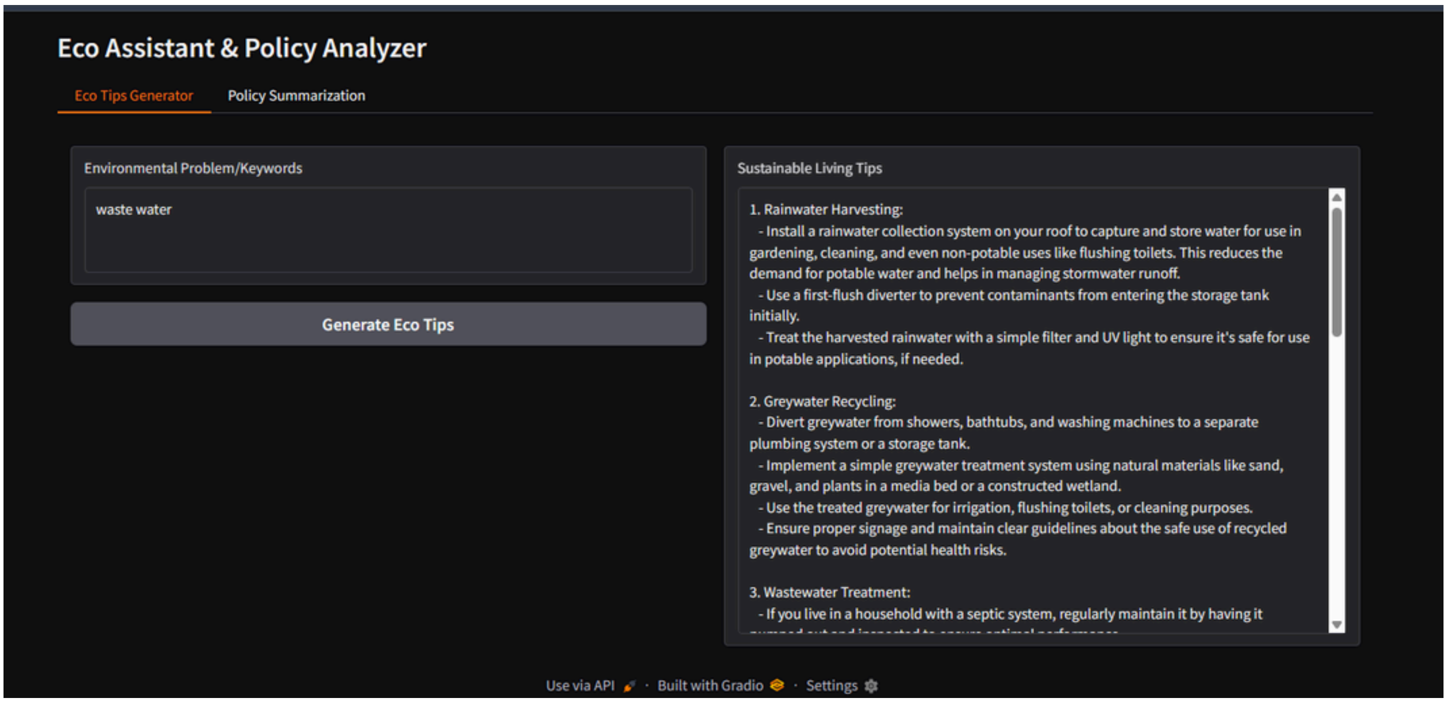
10. Testing & Validation

Testing includes:• Unit testing for prompt functions• API testing via Swagger/Postman• Manual testing for uploads and outputs• Edge-case handling for large files and invalid inputs

11. Future Enhancements

• Integration with IoT sensor networks• Advanced analytics dashboards• Support for multilingual city policies• Mobile app integration

12. Screen shots


Eco Assistant & Policy Analyzer

Environmental Problem/Keywords

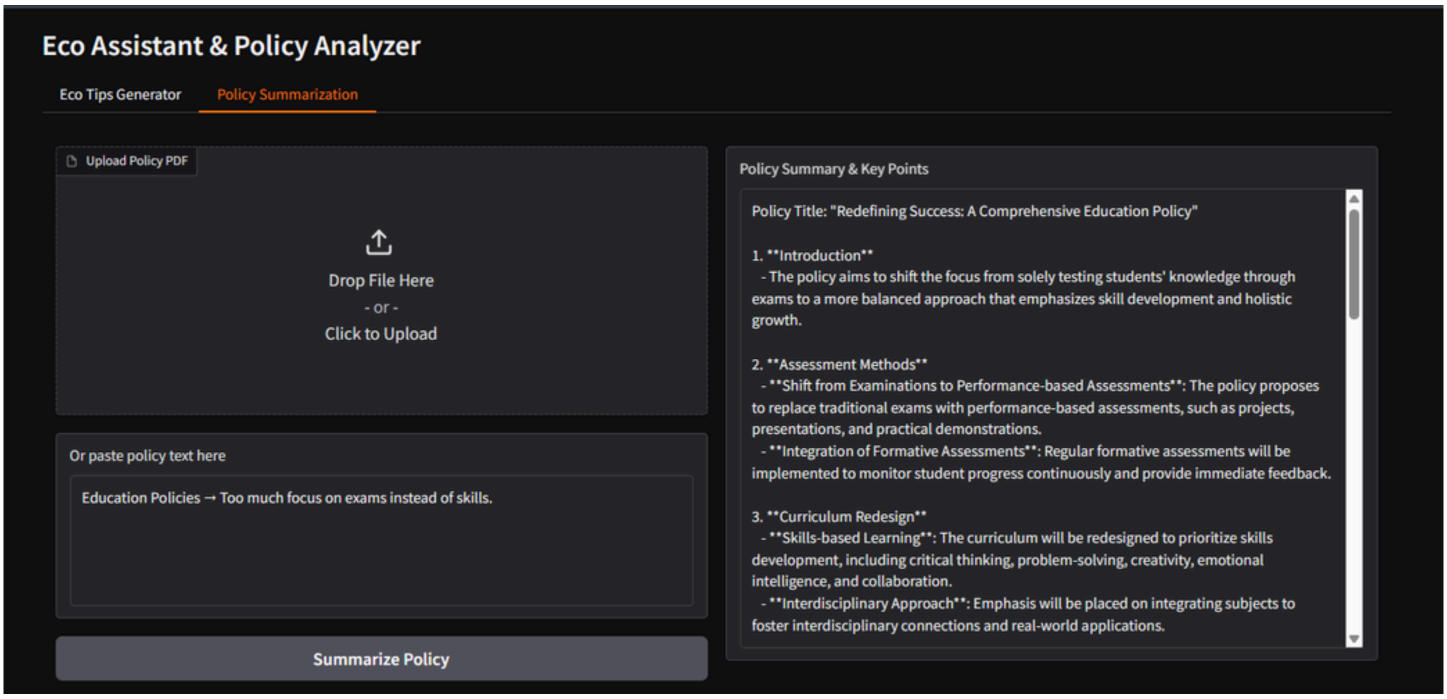
waste water

Sustainable Living Tips

1. Rainwater Harvesting:
   - Install a rainwater collection system on your roof to capture and store water for use in gardening, cleaning, and even non-potable uses like flushing toilets. This reduces the demand for potable water and helps in managing stormwater runoff.
   - Use a first-flush diverter to prevent contaminants from entering the storage tank initially.
   - Treat the harvested rainwater with a simple filter and UV light to ensure it's safe for use in potable applications, if needed.

2. Greywater Recycling:
   - Divert greywater from showers, bathtubs, and washing machines to a separate plumbing system or a storage tank.
   - Implement a simple greywater treatment system using natural materials like sand, gravel, and plants in a media bed or a constructed wetland.
   - Use the treated greywater for irrigation, flushing toilets, or cleaning purposes.
   - Ensure proper signage and maintain clear guidelines about the safe use of recycled greywater to avoid potential health risks.

3. Wastewater Treatment:
   - If you live in a household with a septic system, regularly maintain it by having it


Eco Assistant & Policy Analyzer

Or paste policy text here

Education Policies → Too much focus on exams instead of skills.

Policy Summary & Key Points

Policy Title: "Redefining Success: A Comprehensive Education Policy"

1. **Introduction**
   - The policy aims to shift the focus from solely testing students' knowledge through exams to a more balanced approach that emphasizes skill development and holistic growth.

2. **Assessment Methods**
   - **Shift from Examinations to Performance-based Assessments**: The policy proposes to replace traditional exams with performance-based assessments, such as projects, presentations, and practical demonstrations.
   - **Integration of Formative Assessments**: Regular formative assessments will be implemented to monitor student progress continuously and provide immediate feedback.

3. **Curriculum Redesign**
   - **Skills-based Learning**: The curriculum will be redesigned to prioritize skills development, including critical thinking, problem-solving, creativity, emotional intelligence, and collaboration.
   - **Interdisciplinary Approach**: Emphasis will be placed on integrating subjects to foster interdisciplinary connections and real-world applications.

13. Source 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,
            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)
```

14. GitHub & Project Demo Link GitHub Repo:

https://github.com/mahasri2007/sub-smart-city.git