CHIDHAMBARAM PILLAI COLLEGE FOR WOMEN

Sustainable Smart City Assistant Using IBM Granite LLM

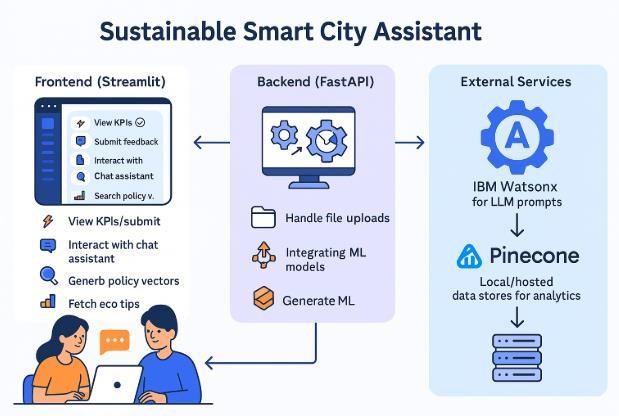
Student Leader : Naveena P

EXTERNAL INTERNAL

ABSTRACT

The Sustainable Smart City Assistant is an AI-powered platform that leverages IBM Watsonx's Granite LLM and modern data pipelines to support urban sustainability, governance, and citizen engagement. It integrates several modules like City Health Dashboard, Citizen Feedback, Document Summarization, Eco-Advice, Anomaly Detection, KPI forecasting and Chat Assistant through a modular FastAPI backend and a Streamlit- based frontend dashboard.

PROJECT DESCRIPTION



Introduction to Python

Python is a high-level, general-purpose programming language known for its readability and simplicity. Its design emphasizes code readability through the use of significant indentation, making it a popular choice for beginners and experienced developers alike.

Introduction to AI

AI works by analyzing data to recognize patterns and make informed decisions, enabling machines to understand human language, recognize objects, and learn from new information. Applications range from personalized recommendations and smart assistants to self-driving cars and advanced data analysis, enhancing efficiency and productivity across various fields like healthcare, finance, and entertainment.

Introduction to IBM GRANITE

IBM Granite is a powerful Large Language Model (LLM) designed for enterprise-grade applications, enabling businesses to leverage advanced natural language processing (NLP) capabilities for various use cases. Built to handle complex, large-scale data and generate accurate, context-aware responses, IBM Granite provides robust performance in tasks such as document summarization, question answering, content generation, and intelligent data analysis.

FRONTEND

STREAMLIT:

Streamlit is an open-source Python library designed to simplify the creation and sharing of custom web applications, particularly for machine learning and data science projects. It allows users to build interactive data-driven web apps, dashboards, and reporting tools using only Python, without requiring knowledge of front-end web development technologies like HTML, CSS, or JavaScript.

BACKEND

FAST API:

**FastAPI** is a modern, fast (high-performance) web framework for building APIs with Python, based on standard Python type hints. It is designed to create RESTful APIs quickly and efficiently, with automatic data validation, interactive API documentation, and high performance comparable to frameworks like Node.js and Go.

IBM WATSONX

IBM watsonx is an enterprise-focused, AI and data platform designed to help businesses accelerate the use of generative AI and machine learning in their core workflows. It comprises three main components for building, training, and deploying AI models; watsonx.data for managing and accessing trusted enterprise data; and watsonx.governance for ensuring AI applications are secure, transparent, and compliant with regulations. The platform offers flexibility by allowing users to leverage IBM, third-party, or open-source foundation models, and it supports various deployment environments, including hybrid and multi-cloud scenarios.

MODULES

**Policy Search & Summarization**

A municipal planner uploads a complex city policy document to the assistant’s interface. In seconds, the assistant summarizes it into a concise, citizen-friendly version using IBM Granite LLM. This empowers planners to quickly interpret key points and make informed urban decisions.

**Citizen Feedback Reporting**

A resident notices a burst water pipe on a city street. Instead of calling helplines, they submit a report through the assistant’s feedback form. The issue is logged instantly with category tagging (e.g., "Water") and can be reviewed by city administrators.

**KPI Forecasting**

A city administrator uploads last year’s water usage KPI CSV. The assistant forecasts next year’s consumption using built-in machine learning. This data is used in planning budgets and infrastructure upgrades.

**Coding**

import gradio as gr

import torch

from transformers import AutoTokenizer, AutoModelForCausalLM

import io

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):

    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)

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():

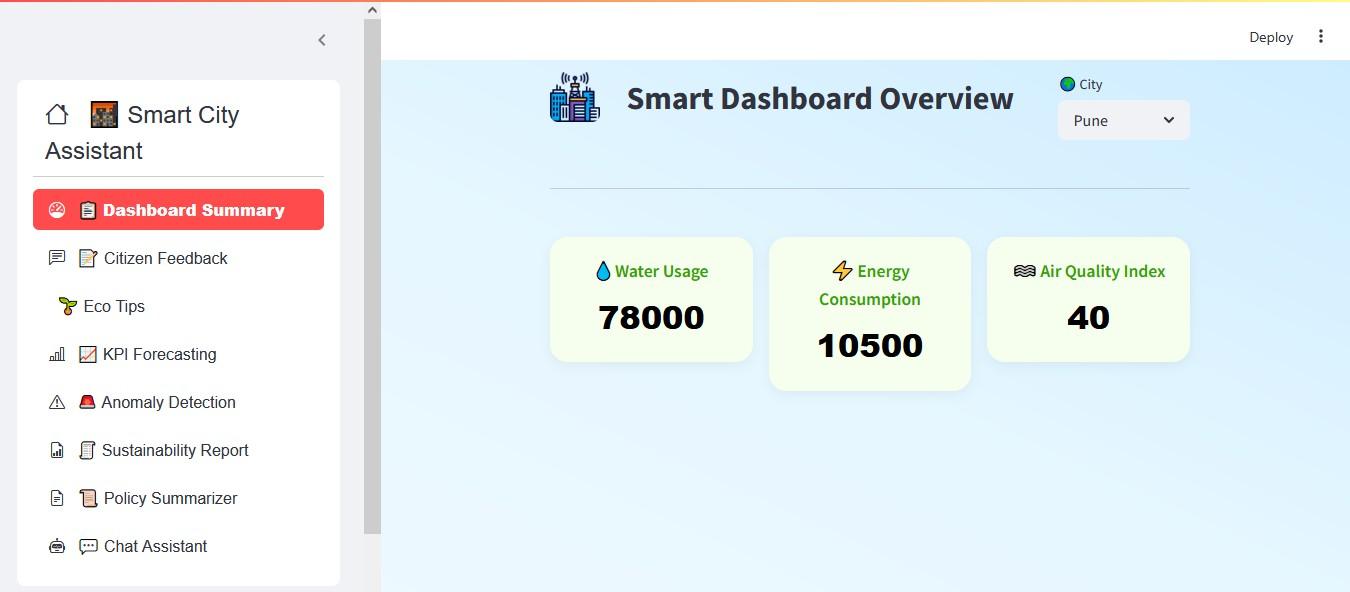
  summary\_output = gr.Textbox(label="Policy Summary & Key Points", lines=20)

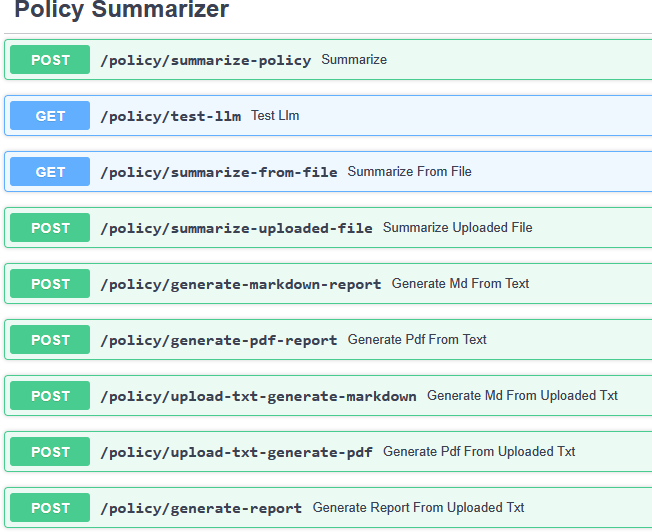
summarize\_btn.click(policy\_summarization, inputs=[pdf\_upload,

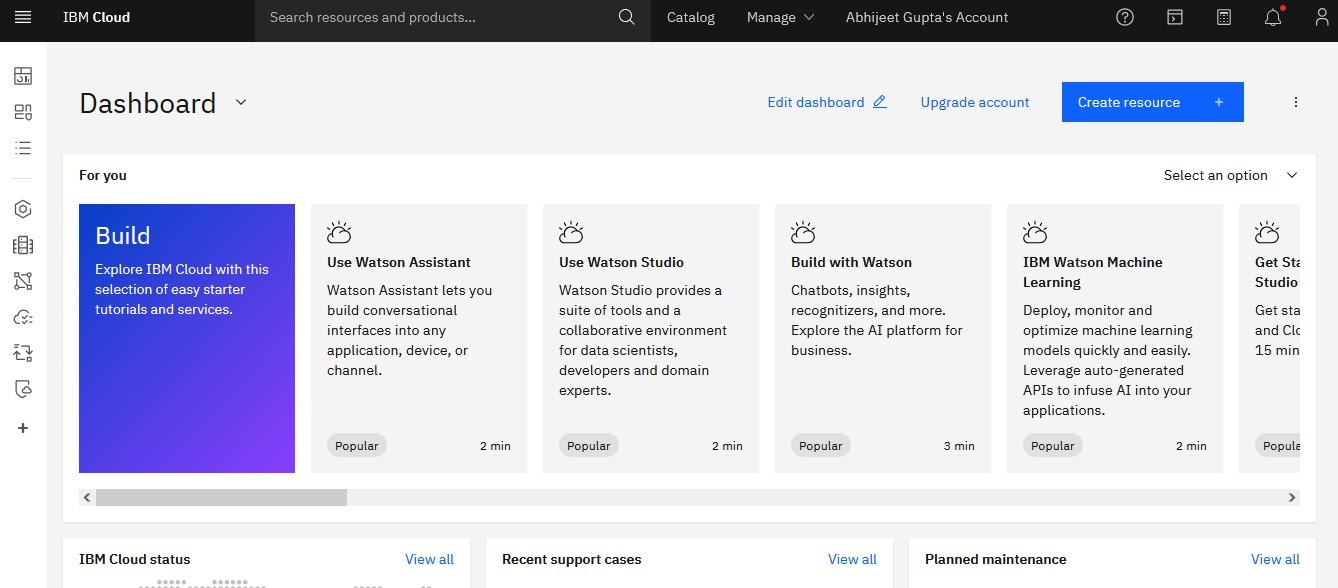
policy\_text\_input], outputs=summary\_output)

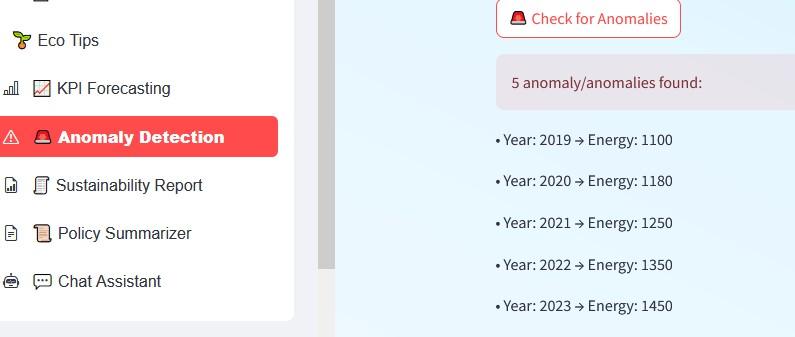
app.launch(share=True)

Output screenshot











CONCLUSION

The integration of IBM Granite Large Language Model (LLM) into a Sustainable Smart City Assistant demonstrates a powerful solution for enhancing urban management, promoting sustainability, and improving citizen engagement. By leveraging Granite LLM’s advanced natural language understanding and generative capabilities, the assistant efficiently processes large volumes of diverse urban data—ranging from energy consumption, traffic patterns, waste management, to public service requests—and delivers actionable insights in real-time.