SUSTAINABLE SMART CITY ASSISTANT USING IBM GRANITE LLM

Project Documentation

1. Introduction :

Project title : SUSTAINABLE SMART CITY ASSISTANT USING IBM GRANITE LLM

- Team member : SRIRAM CJ
- Team member : SYED RIZWAN
- Team member : THARUN R
- Team member : VISHWA N

2. Project Overview:

- **Purpose:** Sustainable Smart City Assistant uses the Granite model from Hugging Face to help with city sustainability, governance, and citizen engagement. It includes quick tools for a City Health Dashboard, citizen feedback, document summaries and eco tips. This project will be deployed in Google Colab using Granite for easy setup and smooth performance.

- **Features:**

  - **City Health Dashboard – Provides quick insights into the overall health and sustainability metrics of the city.**
  - **Citizen Feedback System – Allows citizens to share opinions and concerns, improving engagement and governance.**
  - **Document Summarization – Helps in quickly summarizing long city-related documents and reports for easier understanding.**
  - **Eco-friendly Tips – Offers practical sustainability tips for citizens to adopt greener lifestyles.**
  - **AI-powered Insights – Uses IBM Granite LLM for natural language processing and decision support.**
  - **Easy Deployment – Runs on Google Colab with GPU support for smooth performance.**
  - **Interactive User Interface – Built using Gradio, making the assistant user-friendly and accessible through a simple web interface.**

- **Open-source Collaboration – The project is uploaded to GitHub for version control, sharing, and teamwork.**
- **Lightweight Model – Uses granite-3.2-2b-instruct, which is efficient and fast, suitable for real-time usage.**
- **Scalable & Adaptable – Can be expanded to support more smart city services in the future.**

3. Architecture:

- **Frontend (Streamlit):** The frontend is an interactive web UI with multiple pages for dashboards, file uploads, a chat interface, feedback forms, and report viewers. It uses the Streamlit-option-menu library for sidebar navigation, and each page is modularized for scalability.
- **Backend (FastAPI):** This serves as the REST framework for API endpoints that handle document processing, chat, eco-tip generation, and more. It is optimized for asynchronous performance and easy Swagger integration.
- **LLM Integration (IBM Watsonx Granite):** The project uses Granite LLM models from IBM Watsonx for natural language understanding and generation. Prompts are specifically designed to produce summaries, reports, and sustainability tips.
- **Vector Search (Pinecone):** Uploaded policy documents are converted into embeddings using Sentence Transformers and stored in Pinecone. Semantic search is enabled via cosine similarity, letting users search documents using natural language queries.
- **ML Modules (Forecasting and Anomaly Detection):** Lightweight ML models from Scikit-learn are used for forecasting and anomaly detection. Time-series data is parsed, modeled, and visualized using pandas and matplotlib.

4. Setup Instructions:

- **Prerequisites:**
  - Python 3.9 or later
  - pip and virtual environment tools
  - API keys for IBM Watsonx and Pinecone
  - Internet access for cloud services
- **Installation Process:**
  - Clone the repository.
  - Install dependencies from requirements.txt.
  - Create and configure a .env file with credentials.

- Run the backend server using FastAPI.
- Launch the frontend via Streamlit.
- Upload data and interact with the modules.

5. Folder Structure:

- app/ - Contains all FastAPI backend logic, including routers, models, and integration modules.
- app/api/ - Subdirectory for modular API routes like chat, feedback, and document vectorization.
- ui/ - Contains frontend components for Streamlit pages and form UIs.
- smart_dashboard.py - The entry script for the main Streamlit dashboard.
- granite_llm.py - Handles all communication with the IBM Watsonx Granite model.
- document_embedder.py - Converts documents to embeddings and stores them in Pinecone.
- kpi_file_forecaster.py - Forecasts future trends for energy/water using regression.
- anomaly_file_checker.py - Flags unusual values in uploaded KPI data.
- report_generator.py - Constructs AI-generated sustainability reports.

6. Running the Application:

- To start the project, launch the FastAPI server and then run the Streamlit dashboard.
- Navigate through the pages using the sidebar.
- Users can upload documents or CSVs, interact with the chat assistant, and view outputs like reports, summaries, and predictions.
- All interactions are real-time, with the frontend dynamically updating via backend APIs.

7. API Documentation:

- The backend APIs include:
  - POST /chat/ask - Accepts a user query and returns an AI-generated message.
  - POST /upload-doc - Uploads and embeds documents in Pinecone.
  - GET /search-docs - Returns semantically similar policies to a user query.
  - GET /get-eco-tips - Provides sustainability tips on selected topics.
  - POST /submit-feedback - Stores citizen feedback.
- Each endpoint is documented and tested in Swagger UI.

8. Authentication:

- For demonstration purposes, this version of the project runs in an open environment.
- Secure deployments can include:
  - Token-based authentication (JWT or API keys).
  - OAuth2 with IBM Cloud credentials.
  - Role-based access for different user types (admin, citizen, researcher).
- Future enhancements will include user sessions and history tracking.

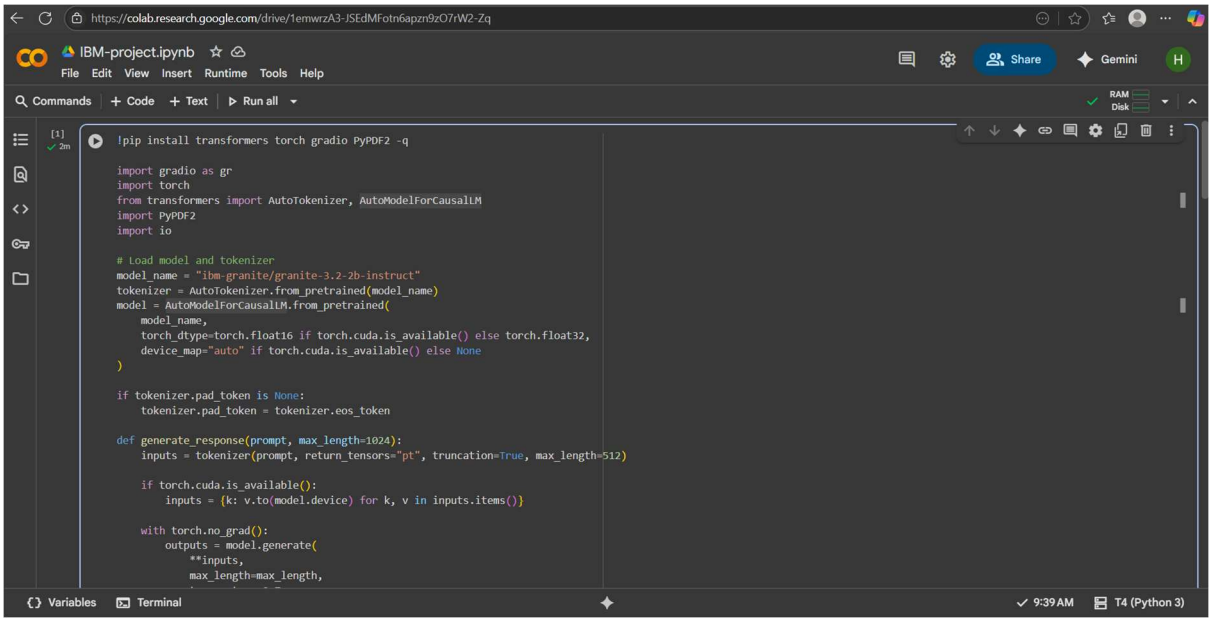
9. User Interface:

- The interface is minimalist and designed for accessibility for non-technical users.
- Key elements include:
  - A sidebar for navigation.
  - KPI visualizations with summary cards.
  - Tabbed layouts for chat, eco tips, and forecasting.
  - Real-time form handling.
  - PDF report download capability.

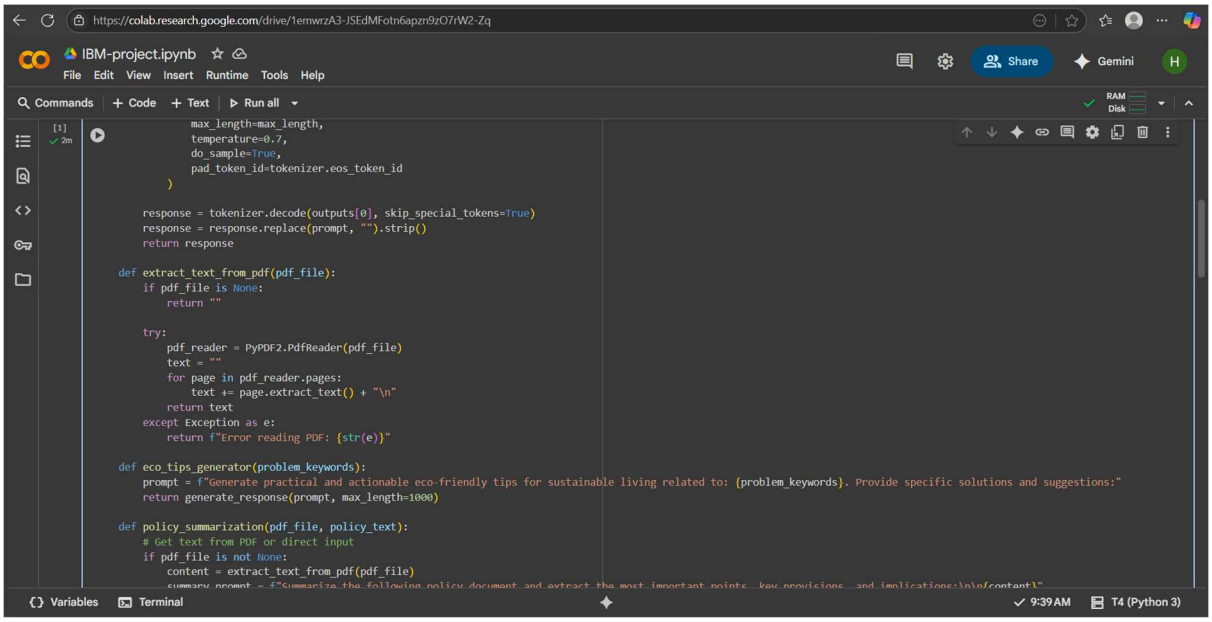
10. Testing:

- Testing was conducted in several phases:
  - **Unit Testing:** For prompt engineering functions and utility scripts.
  - **API Testing:** Done via Swagger UI, Postman, and test scripts.
  - **Manual Testing:** To validate file uploads, chat responses, and output consistency.
  - **Edge Case Handling:** To address malformed inputs, large files, and invalid API keys.
- Each function was validated to ensure reliability in both offline and API-connected modes.

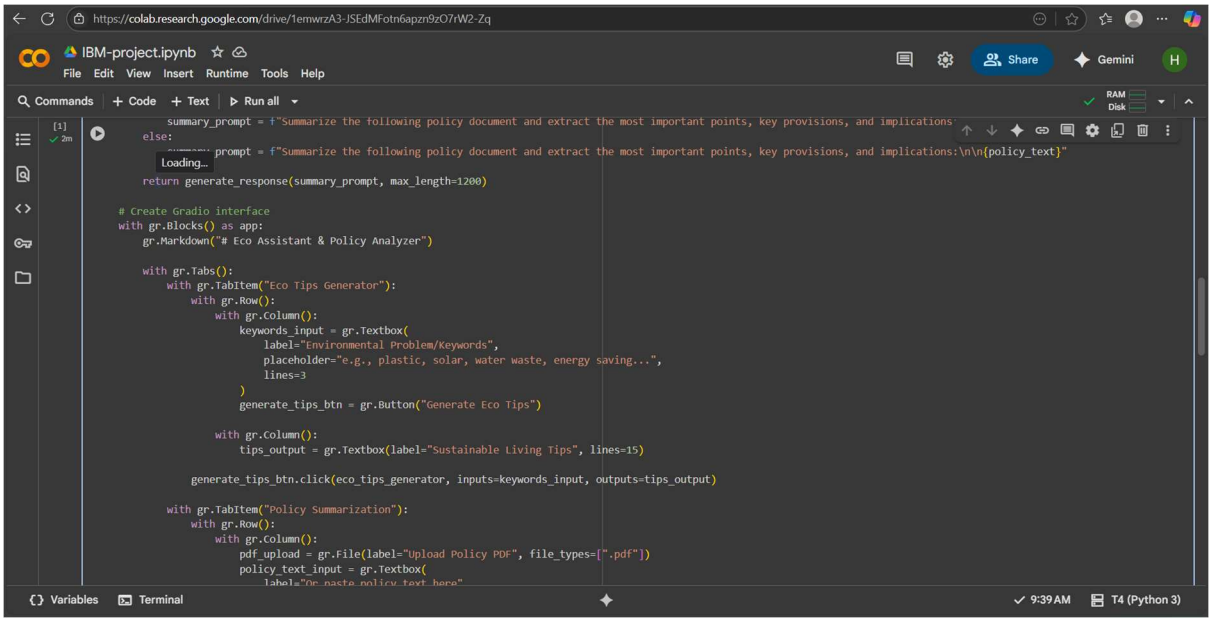
11.Source Code Screenshots:

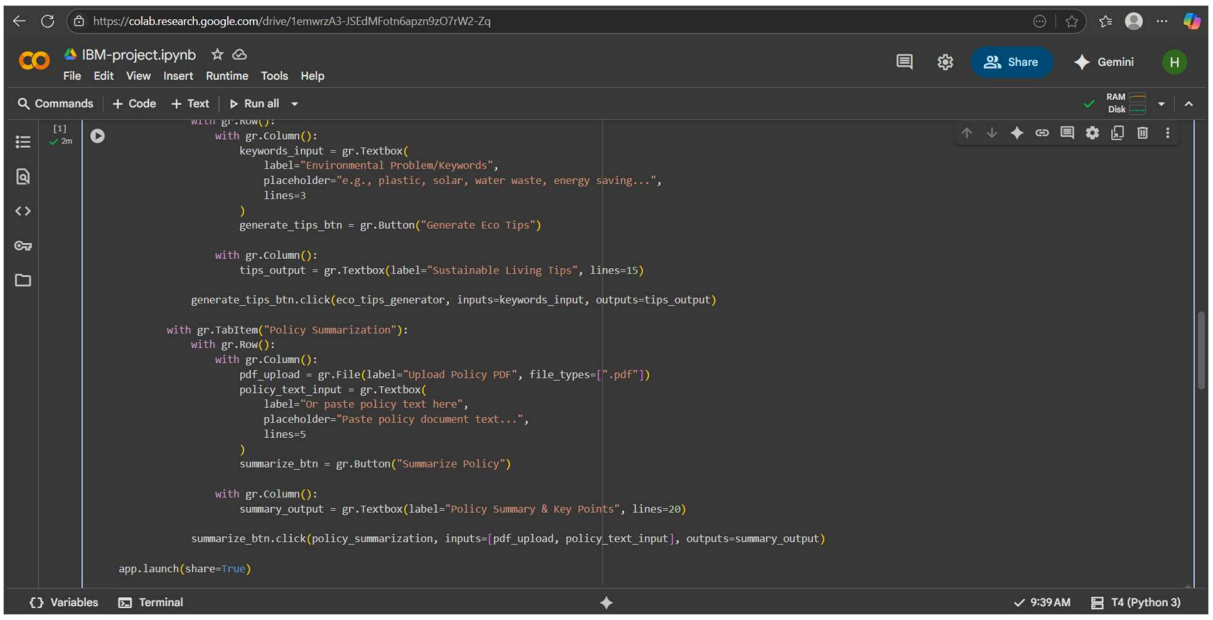


```
!pip install transformers torch gradio PyPDF2 -q

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,
```

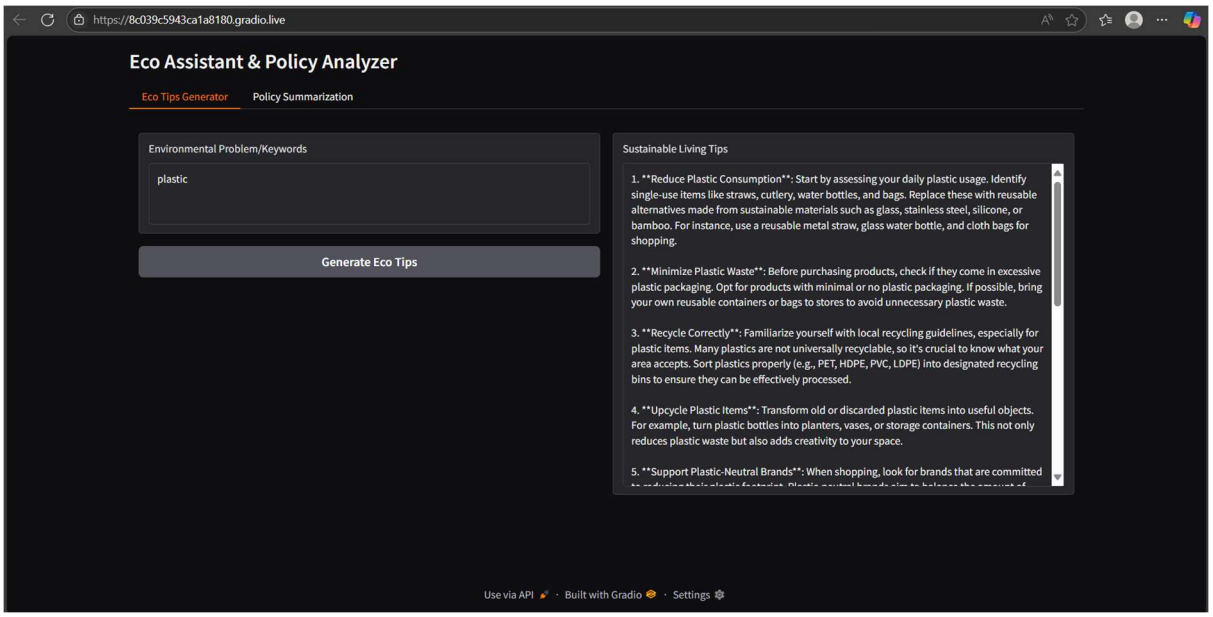


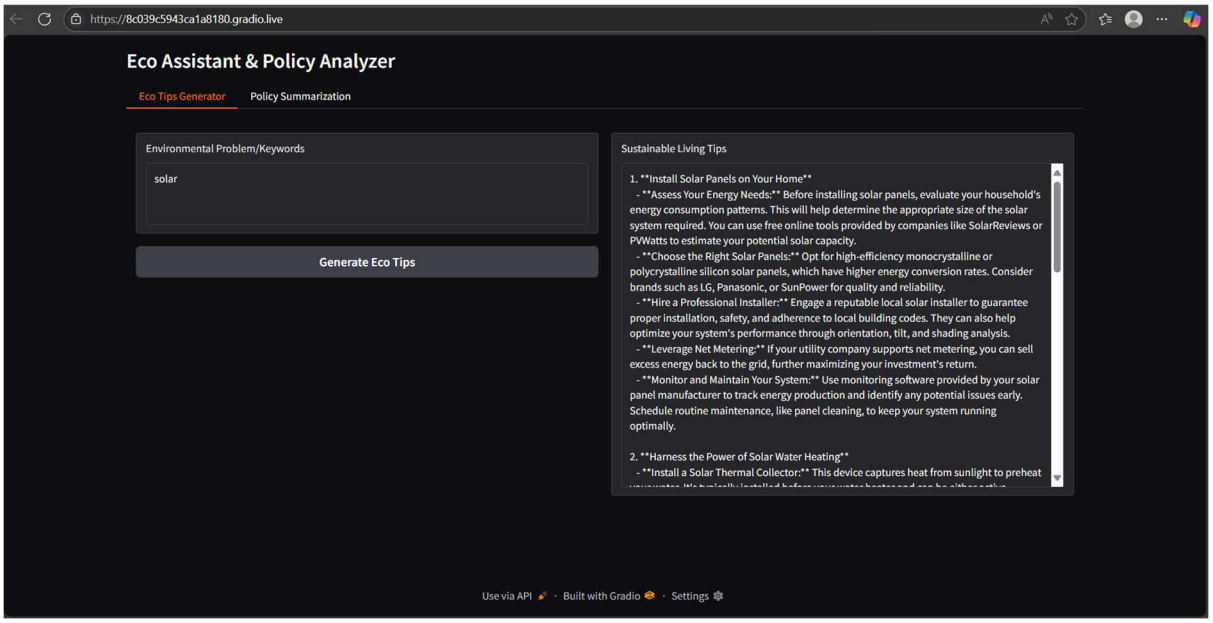
```
            max_length=max_length,
            temperature=0.7,
            do_sample=True,
            pad_token_id=tokenizer.eos_token_id
        )

    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)
```


https://colab.research.google.com/drive/1emwrzA3-JSEdMFotn6apzn9zO7rW2-Zq
IBM-project.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands
+ Code
+ Text
Run all
RAM
Disk
[1]
2m
summary_prompt = f"Summarize the following policy document and extract the most important points, key provisions, and implications
else:
summary_prompt = f"Summarize the following policy document and extract the most important points, key provisions, and implications:\n\n{policy_text}"
Loading...
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"
Variables
Terminal
9:39 AM
T4 (Python 3)


https://colab.research.google.com/drive/1emwrzA3-JSEdMFotn6apzn9zO7rW2-Zq
IBM-project.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands
+ Code
+ Text
Run all
RAM
Disk
[1]
2m
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)
Variables
Terminal
9:39 AM
T4 (Python 3)

12.Source Output:





13. Future Enhancements:

- **User Sessions and History Tracking:** The project plans to add the ability to track user sessions and interaction history. This will allow for a more personalized experience.
- **Security:** For secure deployments, the project can integrate token-based authentication (JWT or API keys), OAuth2 with IBM Cloud credentials, and role-based access for different users (e.g., admin, citizen, researcher).