

SUSTAINABLE SMART CITY ASSISTANT

USING IBM GRANITE LLM

INTRODUCTION :

A sustainable smart city is an urban area that uses modern digital technologies, Artificial Intelligence (AI), Internet of Things (IoT), renewable energy, and eco-friendly infrastructure to improve the quality of life while reducing environmental impact. It focuses on sustainable development through energy efficiency, waste management, smart transportation, clean water supply, and green spaces.

The main goal of a sustainable smart city is to balance economic growth, social well-being, and environmental protection. Technologies like cloud computing, AI-based decision-making, and IoT sensors play a vital role in real-time monitoring and management of resources.

This project explores the design, implementation, and testing of a Smart City model using AI platforms such as Hugging Face, Google Colab, and VS Code. It also analyzes feasibility, problem definition, and the requirements needed for developing a digital framework that promotes sustainable urban living.

PROJECT DESCRIPTION :

Our project “Sustainable Smart City” is designed and developed by our team under the Naan Mudhalvan program.

Team ID: NM2025TMID06256

Team Leader - SREEJA S

Member 1 - SUMATHI S

Member 2 - SWETHA A

Our sustainable smart city model integrates AI, IoT, and cloud-based solutions for efficient energy, waste, water, and transport management.

FEASIBILITY STUDY :

1. Technical Feasibility:

- ❖ Use of Hugging Face for AI model deployment.
- ❖ Cloud execution on Google Colab for scalability.
- ❖ Local development with VS Code.
- ❖ Hardware sensors (IoT devices) for data collection.

2. Economic Feasibility:

- ❖ Open-source platforms reduce cost.
- ❖ Cloud solutions eliminate need for expensive servers.
- ❖ Affordable IoT devices and renewable energy adoption

3. Operational Feasibility:

- ❖ User-friendly dashboards for city administrators.
- ❖ AI-based predictions for traffic, pollution, and waste management.
- ❖ Real-time monitoring of electricity and water usage.

4. Social Feasibility:

- ❖ Enhances citizens' lifestyle with clean, safe, and efficient infrastructure.
- ❖ Promotes sustainable living.
- ❖ Encourages public participation in digital governance.

PROBLEM DEFINITION :

Challenges in Modern Cities:

1. Traffic Congestion – Increasing vehicles cause pollution and time wastage
2. Waste Management – Lack of segregation leads to land pollution.
3. Energy Consumption – Dependence on non-renewable energy sources.
4. Water Scarcity – Poor distribution and wastage.
5. Pollution – Air, water, and noise pollution affecting public health.
6. Unplanned Growth – Uncontrolled construction leads to imbalance.
7. Limited Use of Technology – Lack of integration of AI/IoT for decision-making.

Our Proposed Solution:

- ❖ AI-driven smart traffic management.
- ❖ IoT-based waste segregation and recycling.
- ❖ Renewable energy grid monitoring.
- ❖ Smart water meters with leak detection.
- ❖ AI-based air quality monitoring.
- ❖ Cloud-based dashboard for city administration.

REQUIREMENTS :

The system requires both hardware and software components.

1. Hardware Requirements :

- ❖ IoT sensors (temperature, pollution, motion, smart meters).
- ❖ Microcontrollers (Arduino / Raspberry Pi).
- ❖ Solar panels and renewable energy equipment.

- ❖ Wireless modules (Wi-Fi, LoRa).
- ❖ Edge devices for local computation.

2. Software Requirements :

- ❖ Programming Language: Python.
- ❖ Development Tools: VS Code, Google Colab.
- ❖ AI Libraries: Hugging Face Transformers, TensorFlow, PyTorch.
- ❖ Database: Firebase / MySQL.
- ❖ Cloud: Google Drive / Colab storage.

DESIGN ARCHITECTURE & DATA FLOW DIAGRAM :

1. System Architecture:

Sensors collect real-time data → Data sent to cloud (Google Colab) → AI model (Hugging Face) analyzes data → Dashboard displays insights → City admin takes action.

2. Data Flow Diagram (DFD):

Level 0: Citizen → System → Smart City Dashboard.

Level 1: IoT Devices → Cloud Processing → AI Decision → User Interface.

IMPLEMENTATION :

1. Using Hugging Face for AI Models :

Used Hugging Face Transformers for text classification and prediction (e.g., analyzing traffic/pollution reports).

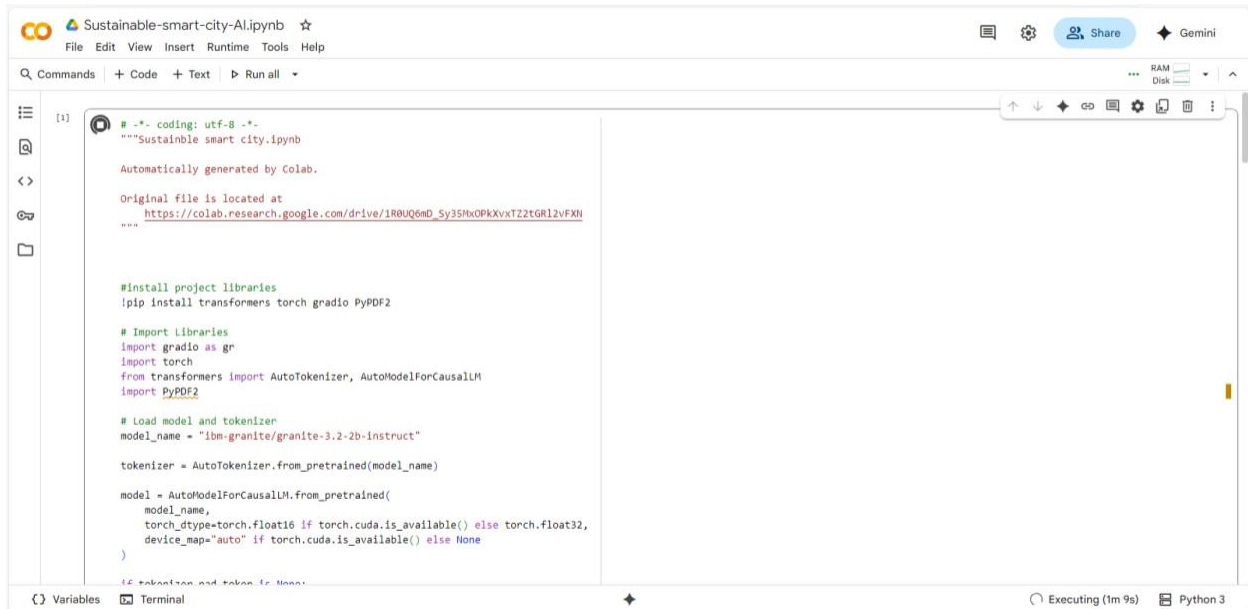
Integrated pre-trained models for Natural Language Processing and data analytics.

2. Using Google Colab for Cloud-Based Execution :

Executed Python scripts in cloud environment.

Trained and tested AI models without requiring local GPU.

Stored datasets in Google Drive for easy access.



```
[1] # -*- coding: utf-8 -*-
"""Sustainable smart city.ipynb

Automatically generated by Colab.

Original file is located at
https://colab.research.google.com/drive/1R0UQ6mD_Sy35MxOPkXvXTZ2tGR12vFXNl
"""

# Install project libraries
!pip install transformers torch gradio PyPDF2

# Import Libraries
import gradio as gr
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM
import PyPDF2

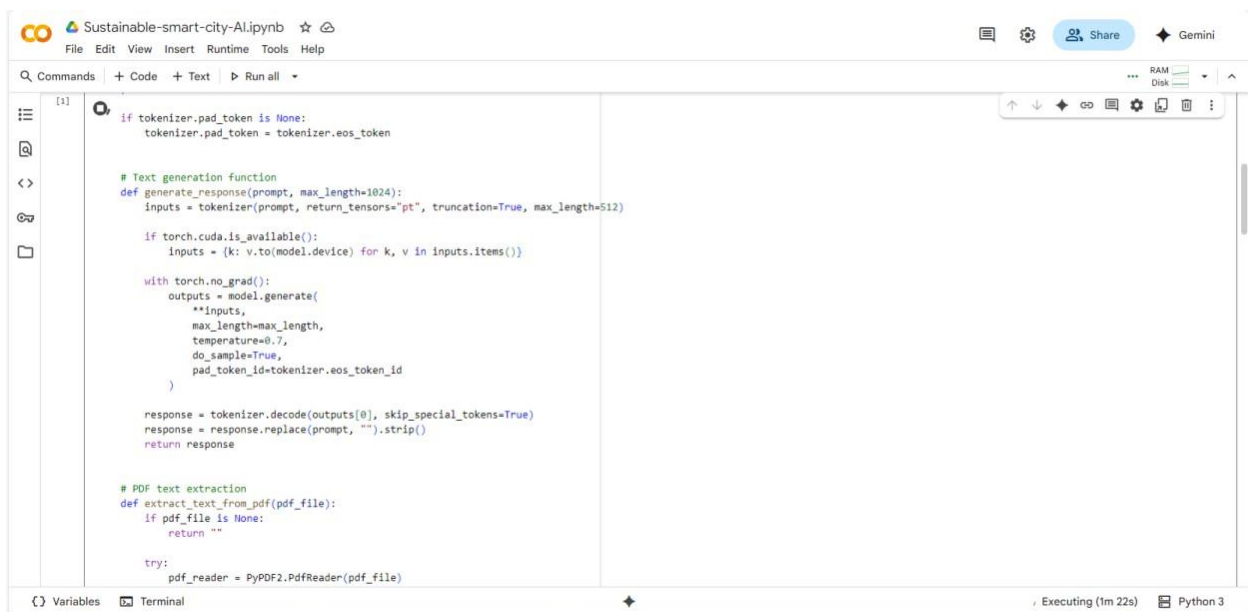
# Load model and tokenizer
model_name = "ibm-granite/granite-3.2-2b-instruct"

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 and token is None:
```

Variables Terminal Executing (1m 9s) Python 3



```
if tokenizer.pad_token is None:
    tokenizer.pad_token = tokenizer.eos_token

# Text generation function
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

# PDF text extraction
def extract_text_from_pdf(pdf_file):
    if pdf_file is None:
        return ""

    try:
        pdf_reader = PyPDF2.PdfReader(pdf_file)
```

Variables Terminal Executing (1m 22s) Python 3

Sustainable-smart-city-AI.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

[1]

```
try:
    pdf_reader = PyPDF2.PdfReader(pdf_file)
    text = ""

    for page in pdf_reader.pages:
        page_text = page.extract_text()
        if page_text:
            text += page_text + "\n"

    return text

except Exception as e:
    return f"Error Reading PDF: {str(e)}"

# Eco tips generator
def eco_tips_generate(problem_keywords):
    prompt = (
        f"Generate practical and actionable eco-friendly tips for sustainable living "
        f"related to: {problem_keywords}. Provide specific solutions and suggestions."
    )
    return generate_response(prompt, max_length=1000)

# Policy summarization
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, key provisions, and implications:\n\n{content}"
        )
    else:
        summary_prompt = f
```

RAM
Disk

Variables Terminal

Executing (1m 29s) Python 3

Sustainable-smart-city-AI.ipynb

File Edit View Insert Runtime Tools Help

Commands + Code + Text Run all

[1]

```
        summary_prompt = (
            f"Summarize the following policy document and extract the most important "
            f"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.Tab("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_generate, inputs=keywords_input, outputs=tips_output)

    with gr.Tab("Policy Summarization"):
        with gr.Row():
            pdf_upload = gr.File(
```

RAM
Disk

Variables Terminal

Executing (1m 33s) Python 3



OUTPUT:

- Now you can see our model is being Downloaded and application is running
- Click on the URL to open the Gradio Application click on the link
[“https://colab.research.google.com/drive/1a3J5VP9cI35cOI4Hg_YsIWxs0fsFx3N”](https://colab.research.google.com/drive/1a3J5VP9cI35cOI4Hg_YsIWxs0fsFx3N).

3. Using VS Code for Local Development :

- ❖ Developed Python scripts and IoT data simulation.
- ❖ Debugging and testing at local level.
- ❖ Screenshots of code editor with working output included.

TESTING :

- ❖ Unit Testing: Verified each module (IoT, AI, cloud, dashboard).
- ❖ Integration Testing: Checked combined performance of sensors + AI + cloud.
- ❖ System Testing: Validated entire smart city workflow.
- ❖ Performance Testing: Measured response time and accuracy.

CONCLUSION :

This project demonstrates how AI, IoT, and cloud computing can transform cities into sustainable smart ecosystems. By integrating Hugging Face models, Google Colab cloud execution, and VS Code local development, we created a cost-effective, scalable, and practical solution.

Our smart city model provides better energy efficiency, waste management, and transportation systems while reducing pollution and improving citizens' quality of life.

Future improvements can include blockchain for data security, 5G integration, and larger-scale real-time deployment.

.....