Sustainable Smart City Assistant Project Documentation

1. Introduction

• Project Title: Sustainable Smart City Assistant Using IBM Granite LLM

• Team Members:

1. HARSHITHA P R
2. RAMJAN FARHATH S
3. SWATHILAKSHMI G

2. Project Overview

• **Purpose:**

The Sustainable Smart City Assistant is an AI-driven platform that equips cities and their residents with eco-friendly tools and data-driven insights. Utilizing IBM Watsonx Granite LLM and advanced data pipelines, it enhances the management of energy, water, and waste resources while offering easy-to-understand policy summaries, avenues for citizen feedback, environmental advice, KPI forecasting, and anomaly detection. This platform connects technology, governance, and community involvement to promote greener, more inclusive, and resilient urban spaces.

• **Features:**

Conversational Chat Assistant

Key Point: Natural language interaction

Functionality: Allows citizens and officials to ask sustainability-related questions and receive AI-driven advice.

Policy Summarization

Key Point: Simplified comprehension

Functionality: Converts complex city policy documents into clear, actionable summaries.

Resource Forecasting

Key Point: Predictive analytics

Functionality: Projects future water, energy, and waste consumption using historical data.

Eco-Tip Generator

Key Point: Sustainable lifestyle guidance

Functionality: Suggests daily eco-friendly actions tailored to user input.

Citizen Feedback Reporting

Key Point: Instant issue reporting

Functionality: Enables residents to quickly report city problems for government attention.

KPI Forecasting & Anomaly Detection

Key Point: Strategic insights and early alerts

Functionality: Predicts key performance indicators and identifies irregularities in urban datasets.

Multimodal Input Support

Key Point: Versatile data processing

Functionality: Handles inputs in text, PDF, and CSV formats for summarization, forecasting, and anomaly identification.

Streamlit Dashboard

Key Point: Intuitive user interface

Functionality: Offers interactive dashboards for visualizing data, reports, and eco-related insights.

Use Case Scenarios

Policy Search & Summarization: A municipal planner uploads a complex city policy document, and the assistant generates a simplified summary.

Citizen Feedback Reporting: A resident submits an issue such as a burst pipe, and the assistant logs it with category tagging for officials.

KPI Forecasting: A city administrator submits water usage data from the previous year and receives AI-generated forecasts to aid in planning.

3. Architecture

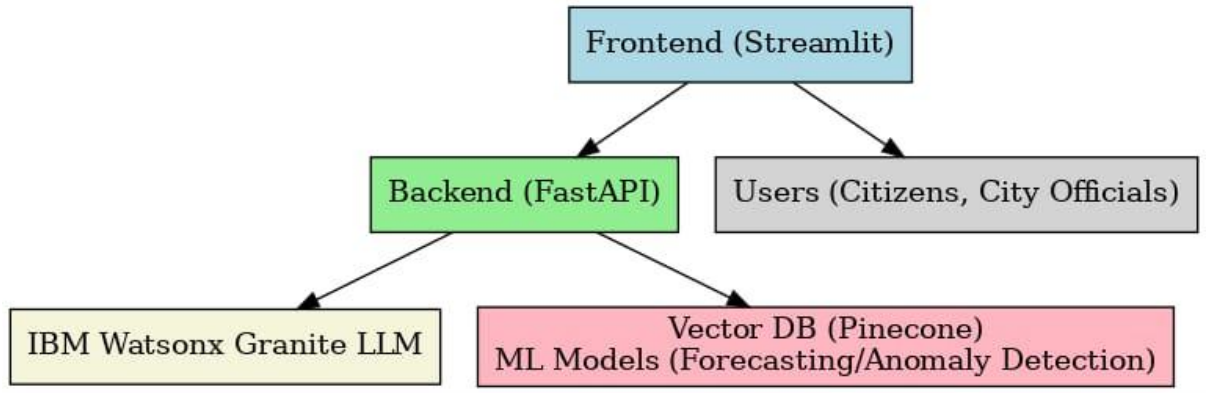
Frontend (Streamlit): A modular dashboard featuring chat, feedback submission, KPI visualization, policy search, eco tips, and anomaly detection.

Backend (FastAPI): RESTful APIs responsible for handling file uploads, machine learning forecasting, anomaly detection, and integrating the large language model.

LLM Integration (IBM Watsonx Granite): Powers summarization, eco-tip generation, sustainability reporting, and conversational AI capabilities.

Vector Search (Pinecone): Enables semantic search of policies through document embeddings.

ML Modules: Implements forecasting and anomaly detection using scikit-learn, pandas, and matplotlib.



4. Setup Instructions

Prerequisities

- Python 3.9 or higher
- FastAPI and Streamlit
- API keys for IBM Watsonx and Pinecone
- scikit-learn, pandas, and matplotlib
- Internet connectivity

Installation Process:

- **Clone the project repository**
- **Install required packages from requirements.txt**
- **Set up API credentials in the .env file**
- **Start the FastAPI backend server**
- **Launch the Streamlit frontend interface**
- **Upload documents or data to begin using the various modules**

5. Folder Structure

app/ – Contains the FastAPI backend logic

app/api/ – Houses routers for chat, feedback, eco tips, policies, and KPI endpoints

ui/ – Includes Streamlit frontend components

smart_dashboard.py – Main script to launch the dashboard

granite_llm.py – Functions for LLM services such as summaries, eco tips, and chat

document_embedder.py – Handles conversion of documents into embeddings

kpi_file_forecaster.py – Performs forecasting of urban KPIs

anomaly_file_checker.py – Detects anomalies within datasets

report_generator.py – Generates sustainability reports

6. Running the Application

- Launch the FastAPI backend
- Start the Streamlit dashboard
- Use the sidebar for navigation
- Upload policy documents or KPI datasets
- Engage with the chat assistant and eco-friendly tools
- Access forecasts, anomaly detections, and sustainability reports

7. API Documentation

POST /chat/ask – Generates AI-powered responses

POST /upload-doc – Uploads and creates embeddings for documents

GET /search-docs – Performs semantic search on policies

GET /get-eco-tips – Retrieves sustainability tips

POST /submit-feedback – Records citizen feedback

8. Authentication

- Token-based authentication using JWT or API keys
- OAuth2 integration with IBM Cloud
- Role-based access control for admins, citizens, and researchers
- Upcoming features: session management and history tracking

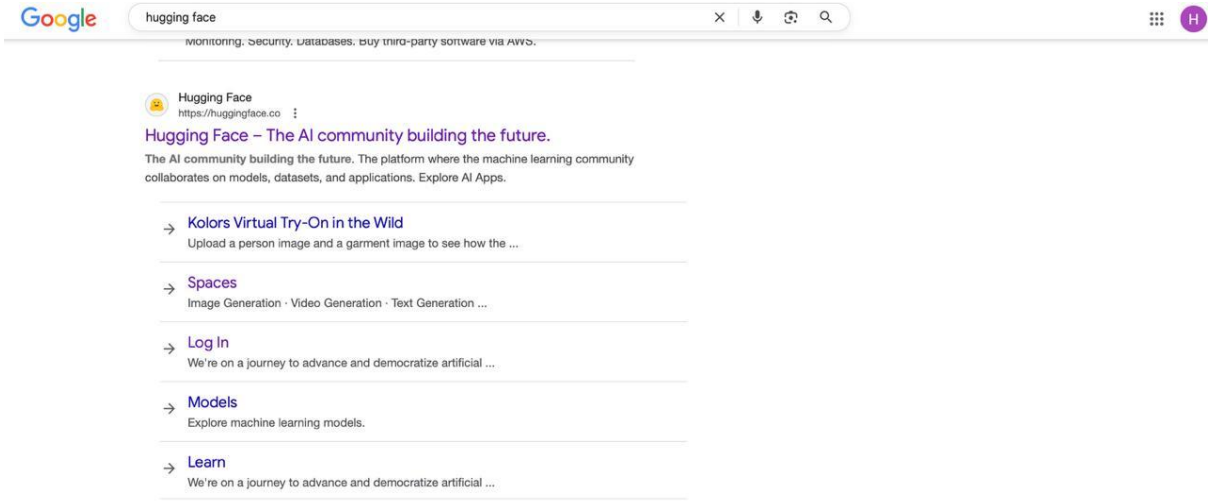
9. User Interface

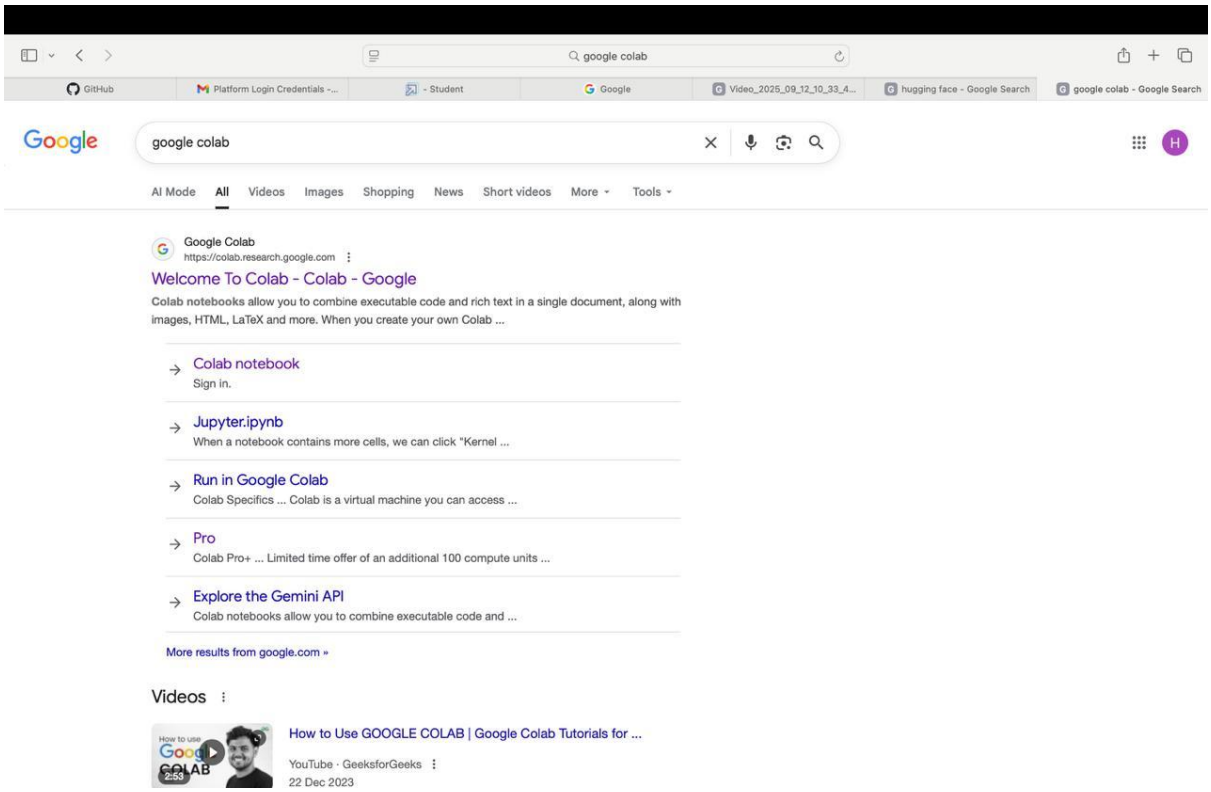
- Sidebar navigation with themed icons
- KPI visualizations accompanied by summary cards
- Chat assistant providing real-time AI responses
- Feedback forms with categorized issue reporting
- Display of policy summaries and eco tips
- Ability to generate and download reports

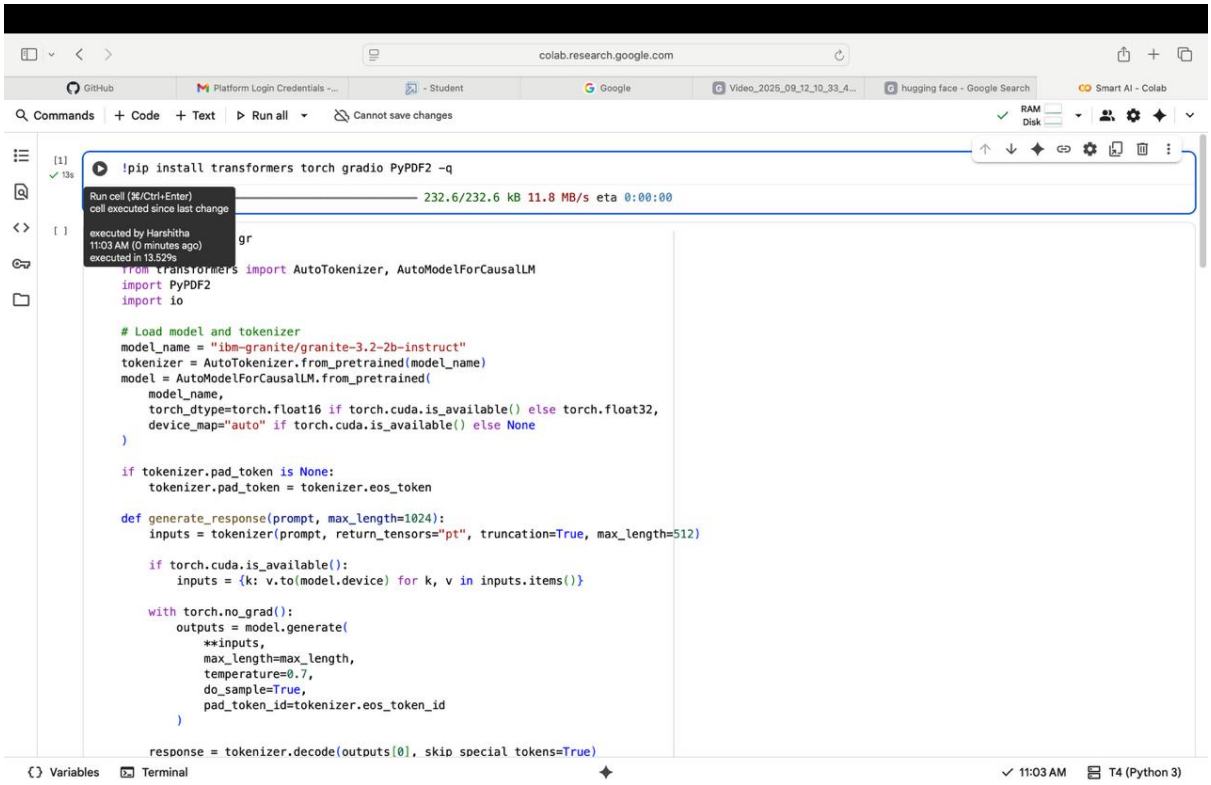
10. Testing

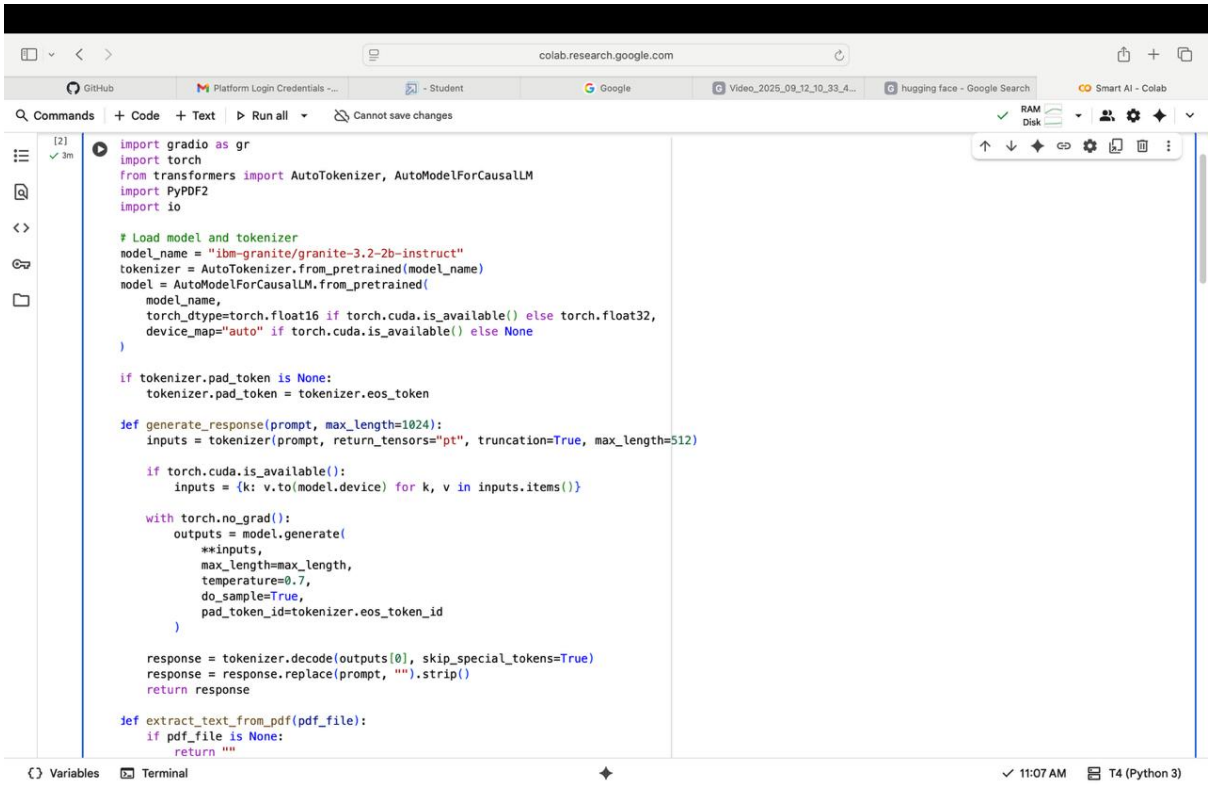
- Unit tests for backend services and machine learning modules
- API testing using Swagger UI and Postman
- Manual testing of chat, policy search, and forecasting features
- Handling edge cases like large file uploads, malformed inputs, and invalid API keys

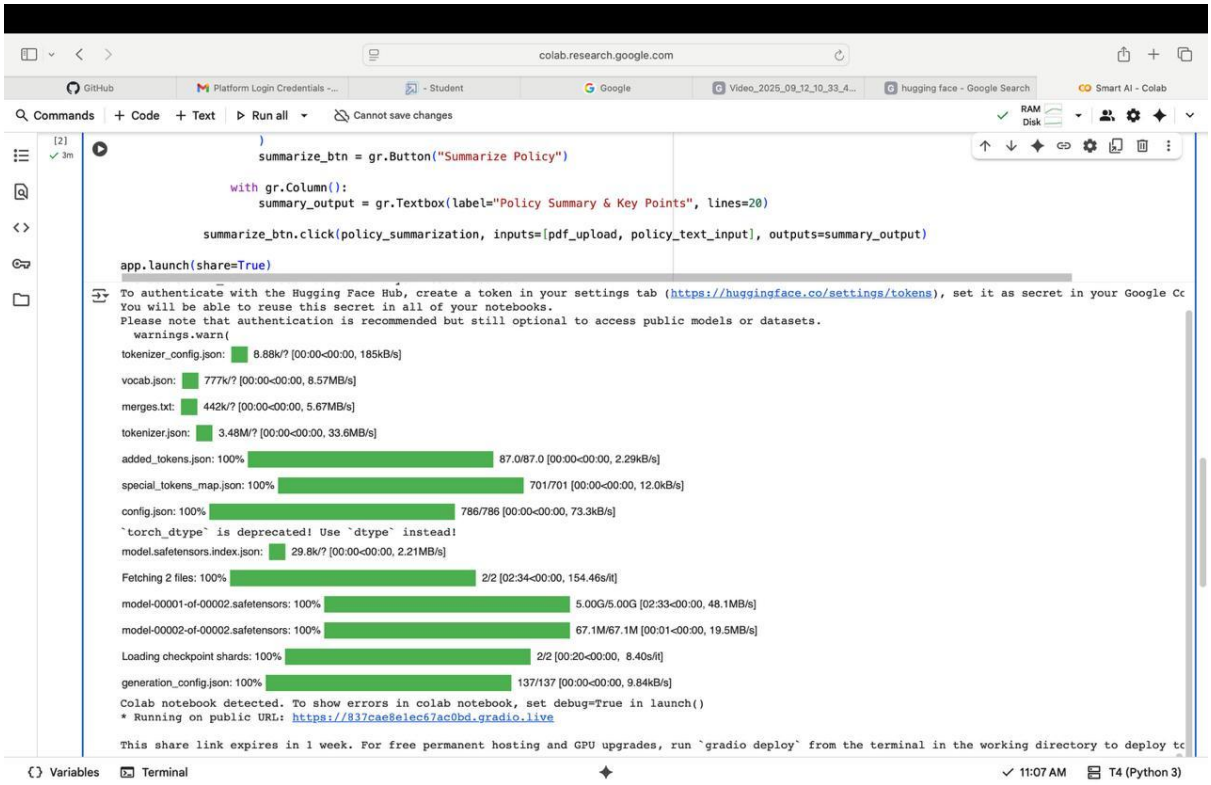
11. Screenshots

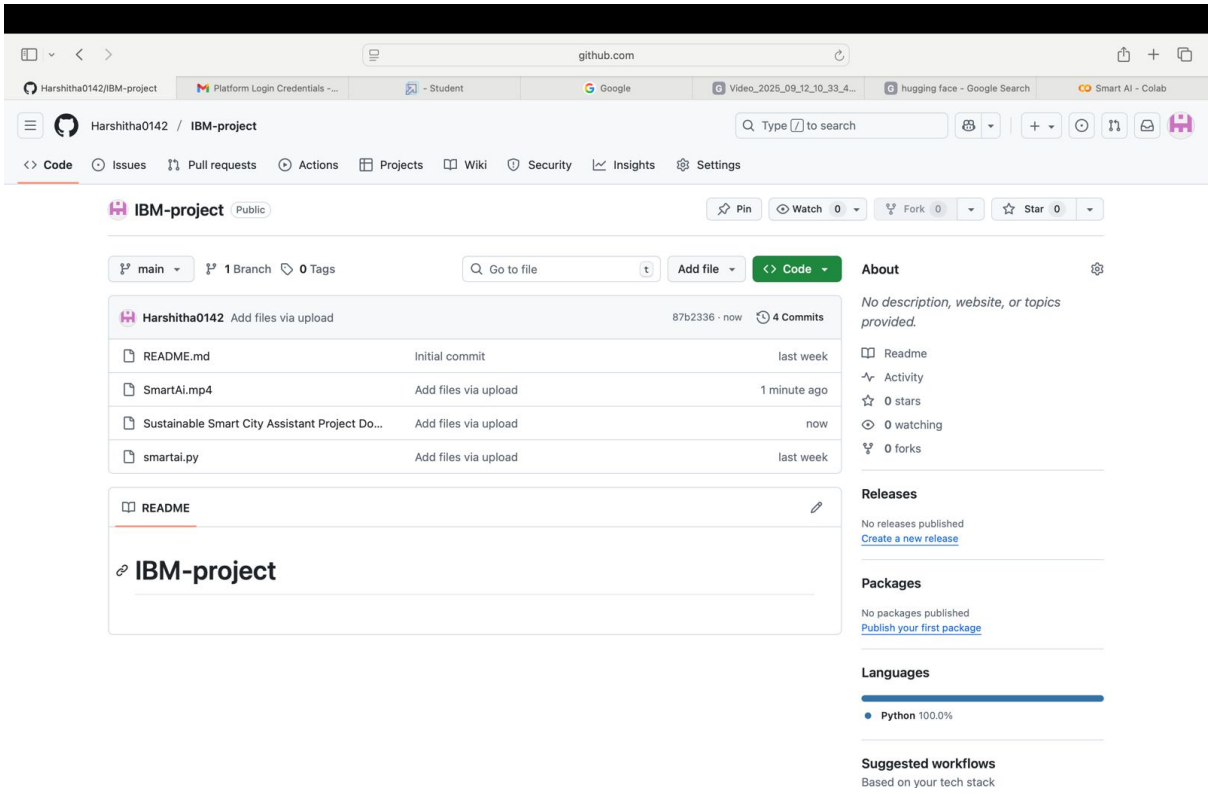





colab.research.google.com
Commands
+ Code
+ Text
Run all
Cannot save changes
RAM
Disk
[1]
13s
!pip install transformers torch gradio PyPDF2 -q
Run cell (⌘/Ctrl+Enter)
cell executed since last change
executed by Harshitha
11:03 AM (0 minutes ago)
executed in 13.529s
232.6/232.6 kB 11.8 MB/s eta 0:00:00
[ ]
gr
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)
Variables
Terminal
11:03 AM
T4 (Python 3)


colab.research.google.com
Commands
+ Code
+ Text
Run all
Cannot save changes
RAM
Disk
[2]
3m
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 ""
Variables
Terminal
11:07 AM
T4 (Python 3)





12. Known Issues

- Support for limited languages
- Requires reliable cloud connectivity
- Subject to API quota restrictions

13. Future Enhancements

- Support for multiple languages
- Integration with IoT devices and city sensors
- Advanced anomaly detection with deep learning
- Mobile-optimized dashboard interface
- Official/doctoral verification of eco policies