Edu Tutor AI: Personalized Learning Generative AI with IBM

1. Introduction

- **Project Title:** EduTutor AI Assistant with IBM Watson
- Team member : JAYANTHASRI S
- Team member : KALPANA DEVI V
- Team member : KAMACHI NIVEDHA S
- Team member : KAMALI A

2. Project Overview

- **Purpose:** The purpose of the EduTutor AI Assistant is to address the lack of personalized and interactive learning experiences in education. The assistant is designed to serve as a smart learning companion for students and a valuable tool for educators. By leveraging AI, it provides tailored content, offers support, and generates insights to help improve the learning process.

- **Features:** The EduTutor AI Assistant includes the following key features:
    - **Interactive Q&A:** Users can ask questions from PDFs and other documents or pose their own queries.
    - **Topic Summarization:** The assistant can summarize complex topics to help with understanding and retention.

    - **Personalized Study Plans:** It generates customized study plans to guide students' learning journey.
    - **Quiz Generator:** The assistant can create quizzes to test a student's knowledge on a given topic.
    - **Teacher Analytics:** It provides insights for educators to help them understand and track student performance.

3. Architecture

The project is built using a modern, scalable architecture composed of a frontend, a backend, and external integrations.

- **Frontend (Streamlit):** The user interface is built with Streamlit, which provides an intuitive and interactive web application. It includes a conversational chat interface, file upload functionality, and a dashboard to display information.

- **Backend (FastAPI):** The backend is powered by FastAPI, a high-performance Python web framework. It handles all the logic for processing user requests, managing document uploads, and interacting with the LLM.

- **LLM Integration (IBM Watson Granite):** The core AI functionality is driven by the IBM Watsonx.ai platform, specifically utilizing the Granite family of models. These models are used for natural language understanding, question answering, summarization, and content generation.

- **Vector Search (Pinecone):** Uploaded documents are converted into vector embeddings using **Sentence-Transformers** and stored in Pinecone. This allows the assistant to perform semantic search, which is crucial for answering questions based on the content of the uploaded files.

- **ML Modules (LangChain):** The project uses the **LangChain** framework to orchestrate the interactions between the LLM, the vector database, and other components. This ensures a streamlined process from user query to AI-generated response.

4. Setup Instructions

To set up and run the EduTutor AI Assistant, follow these steps:

Prerequisites:

- Python 3.9 or later
- **pip** and a virtual environment tool (e.g., **venv**)
- API keys for IBM Watsonx.ai and Pinecone

Installation Process:

1. Clone the project repository.

2. Navigate to the project directory in your terminal.

3. Create and activate a virtual environment:

```
python -m venv venv

source venv/bin/activate    # On macOS/Linux
venv\Scripts\activate       # On Windows
```

4. Install the required dependencies by creating a **requirements.txt** file and running pip install -r requirements.txt. The required libraries include **streamlist, requests,** and **python-dotenv.**

5. Create a .env file in the root directory and configure your credentials. This file should contain your IBM Watsonx and Pinecone API keys and other necessary configurations.

5. Folder Structure

The project is organized into a clean and logical folder structure to separate the frontend from the backend logic.

This structure allows for easy management of both the frontend UI (**edututor_ai.py**) and the backend API **(/api).**

6. Running the Application

To run the EduTutor AI Assistant, you must first start the backend server and then launch the frontend application.

Start the Backend: Navigate to your backend directory and run the FastAPI server.
uvicorn api.main:app --reload

1. **Launch the Frontend:** In a separate terminal, navigate to the project's root directory and launch the Streamlit app.
   streamlit run edututor_ai.py

2. The application will automatically open in your web browser. You can then interact with the chat interface, upload documents, and test the different features.

7. API Documentation

The backend APIs are used to handle user interactions and communicate with the AI models. Based on the provided code, these are the likely API endpoints:

- **POST /chat/ask:** Accepts a user query and responds with an AI-generated message.
- **POST /upload-doc:** Uploads and processes a document (e.g., a PDF) for embedding.
- **POST /generate-quiz:** Generates a quiz based on a specific topic.
- **GET /get-study-plan**: Retrieves a personalized study plan for a user.

8. Authentication

This version of the project runs in an open environment for demonstration purposes. However, for a secure, production-ready deployment, you could implement token-based authentication (e.g., JWT) to secure the API endpoints.

9. User Interface

The user interface is built with Streamlit and is designed to be user-friendly. It features a clean, minimalist layout with a conversational chat window, a file upload button, and sections for viewing generated content like summaries and quizzes. The use of Streamlit allows for an interactive experience without complex front-end development.

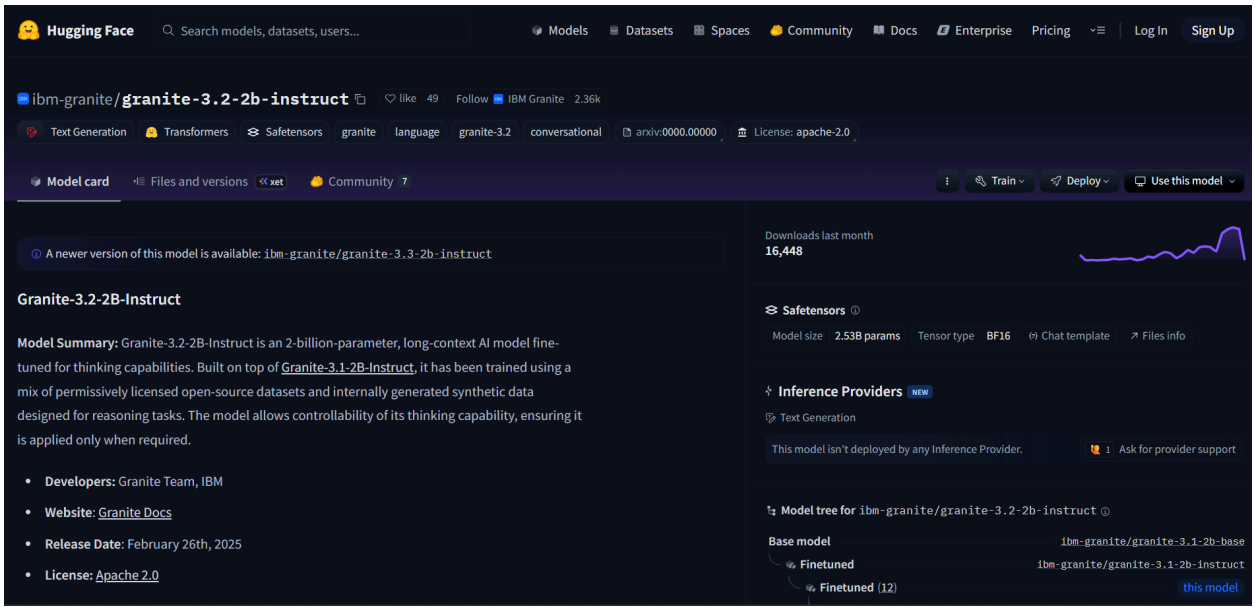
10. Testing

While the project did not include an automated testing suite, manual testing was performed to ensure core functionality. This included:

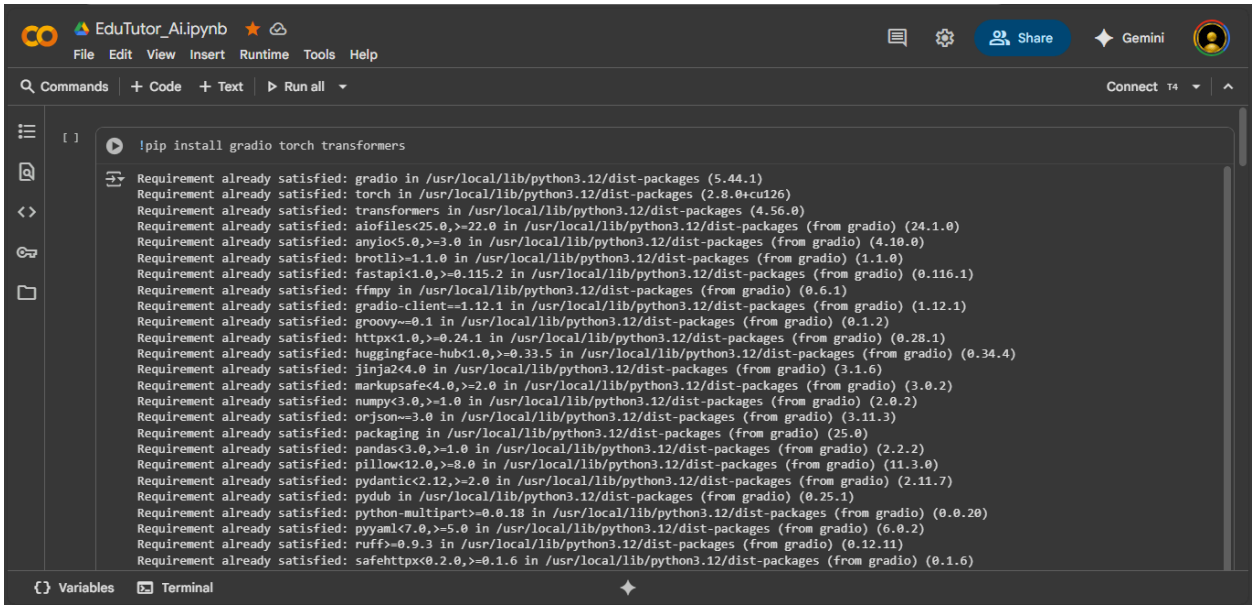
- **File Uploads:** Manually uploading various document types to ensure they are processed correctly.
- **Chat Responses:** Verifying that the AI assistant provides relevant and accurate answers to different types of queries.
- **Feature Validation:** Checking that the summary, quiz, and other features generate the expected output.
- **Edge Case Handling:** Testing with malformed inputs and large files to observe system behavior.

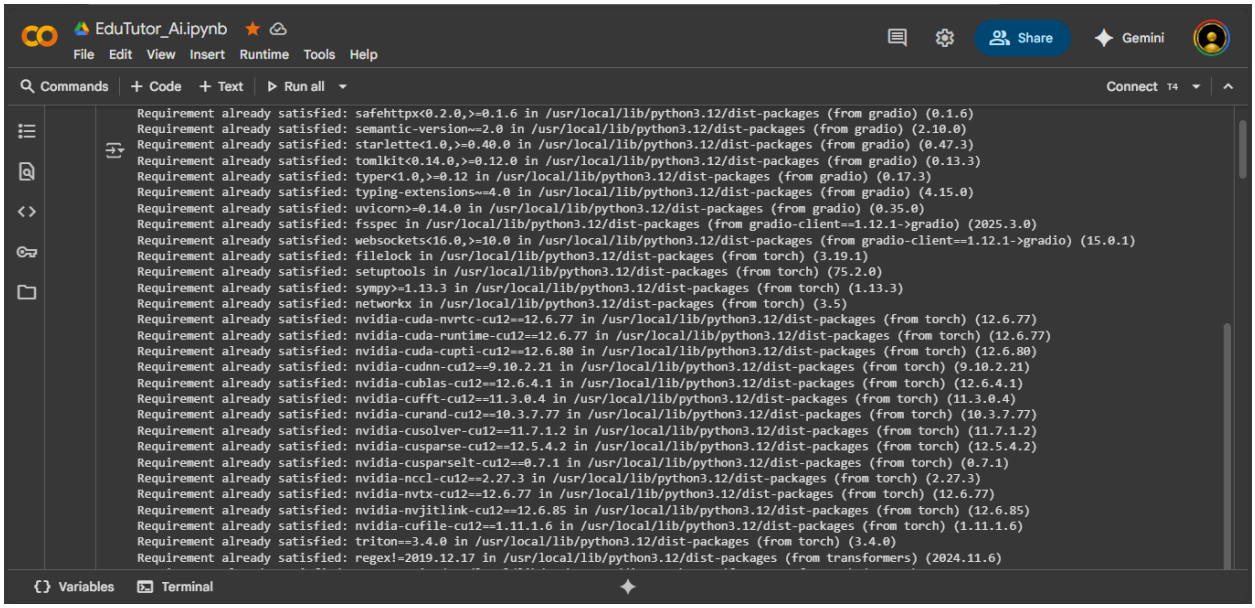
11.Screen Shots :

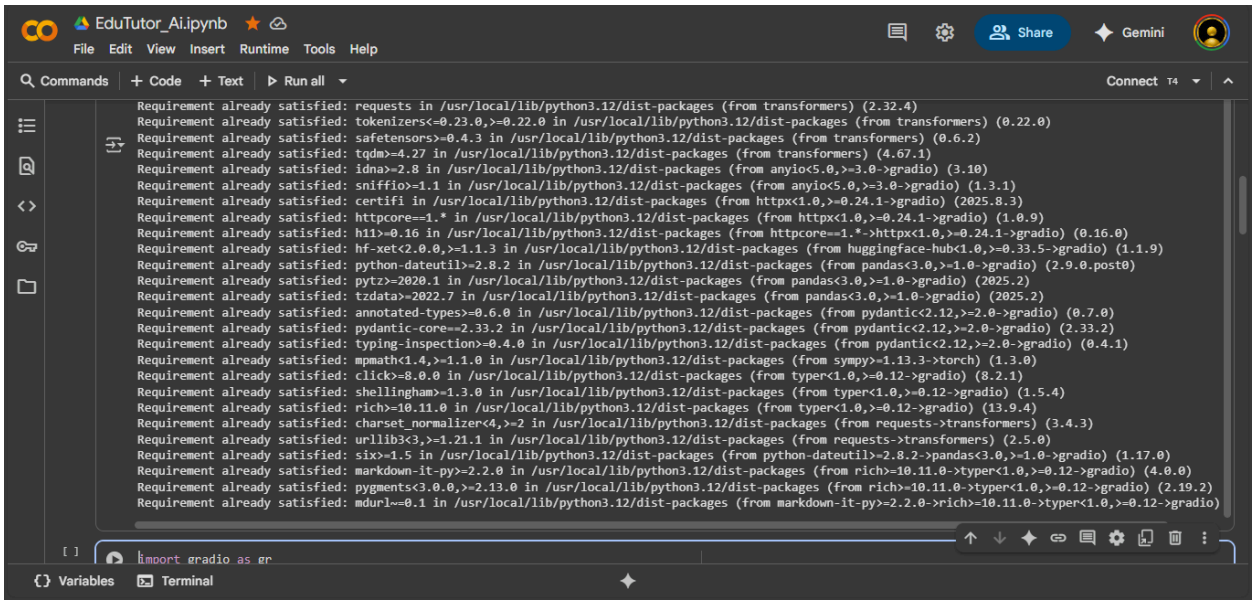
Hugging Face

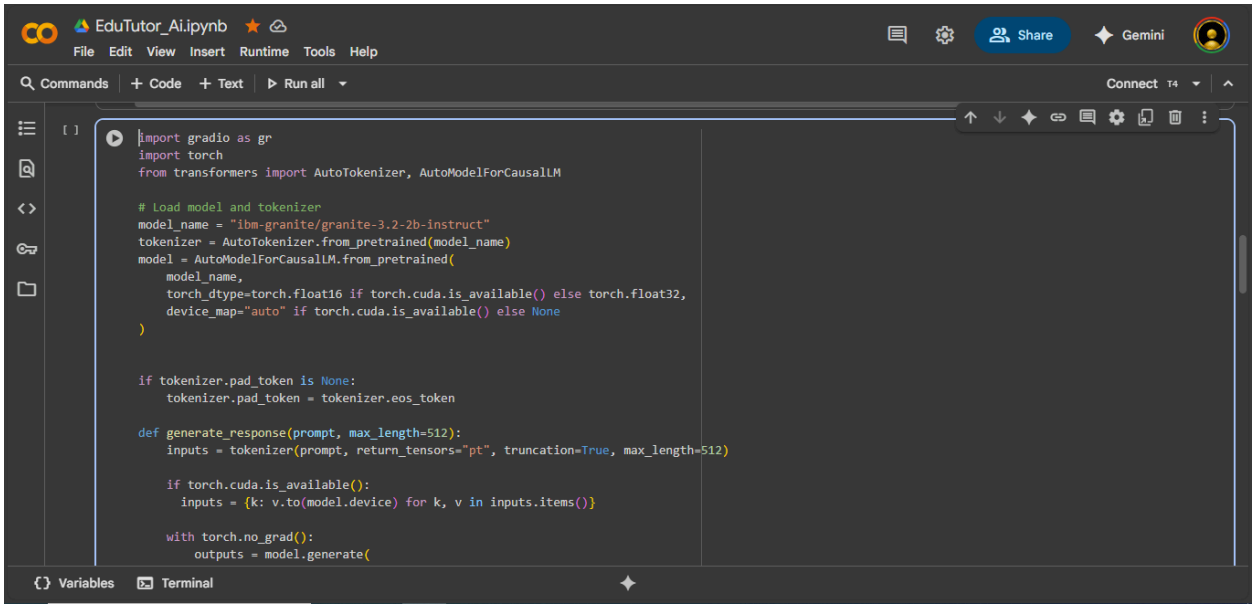


Google Colab:

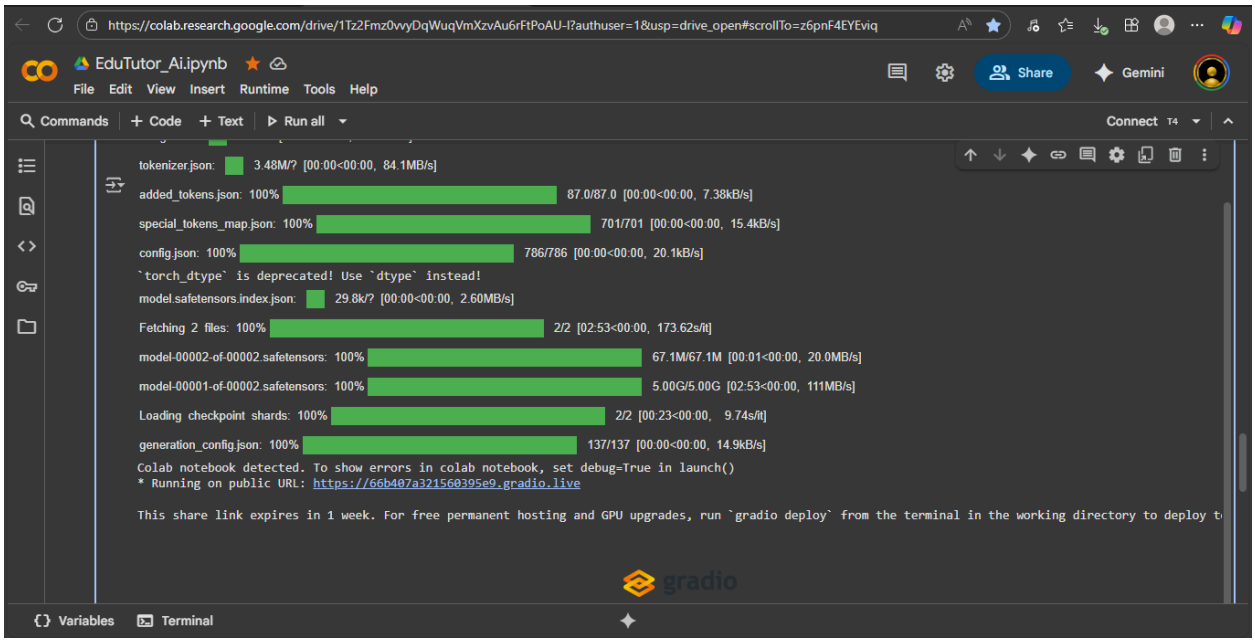


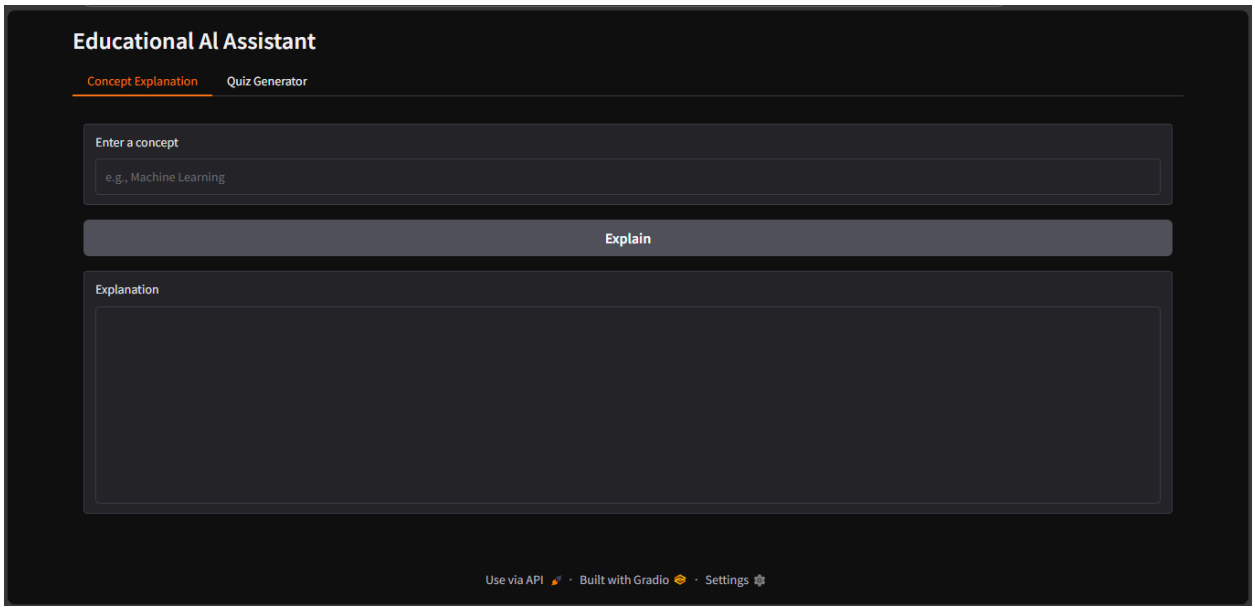

EduTutor_Ai.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands
+ Code
+ Text
Run all
Connect T4
Requirement already satisfied: safehttpx<0.2.0,>=0.1.6 in /usr/local/lib/python3.12/dist-packages (from gradio) (0.1.6)
Requirement already satisfied: semantic-version~=2.0 in /usr/local/lib/python3.12/dist-packages (from gradio) (2.10.0)
Requirement already satisfied: starlette<1.0,>=0.40.0 in /usr/local/lib/python3.12/dist-packages (from gradio) (0.47.3)
Requirement already satisfied: tomlkit<0.14.0,>=0.12.0 in /usr/local/lib/python3.12/dist-packages (from gradio) (0.13.3)
Requirement already satisfied: typer<1.0,>=0.12 in /usr/local/lib/python3.12/dist-packages (from gradio) (0.17.3)
Requirement already satisfied: typing-extensions~=4.0 in /usr/local/lib/python3.12/dist-packages (from gradio) (4.15.0)
Requirement already satisfied: uvicorn>=0.14.0 in /usr/local/lib/python3.12/dist-packages (from gradio) (0.35.0)
Requirement already satisfied: fsspec in /usr/local/lib/python3.12/dist-packages (from gradio-client==1.12.1->gradio) (2025.3.0)
Requirement already satisfied: websockets<16.0,>=10.0 in /usr/local/lib/python3.12/dist-packages (from gradio-client==1.12.1->gradio) (15.0.1)
Requirement already satisfied: filelock in /usr/local/lib/python3.12/dist-packages (from torch) (3.19.1)
Requirement already satisfied: setuptools in /usr/local/lib/python3.12/dist-packages (from torch) (75.2.0)
Requirement already satisfied: sympy>=1.13.3 in /usr/local/lib/python3.12/dist-packages (from torch) (1.13.3)
Requirement already satisfied: networkx in /usr/local/lib/python3.12/dist-packages (from torch) (3.5)
Requirement already satisfied: nvidia-cuda-nvrtc-cu12==12.6.77 in /usr/local/lib/python3.12/dist-packages (from torch) (12.6.77)
Requirement already satisfied: nvidia-cuda-runtime-cu12==12.6.77 in /usr/local/lib/python3.12/dist-packages (from torch) (12.6.77)
Requirement already satisfied: nvidia-cuda-cupti-cu12==12.6.80 in /usr/local/lib/python3.12/dist-packages (from torch) (12.6.80)
Requirement already satisfied: nvidia-cudnn-cu12==9.10.2.21 in /usr/local/lib/python3.12/dist-packages (from torch) (9.10.2.21)
Requirement already satisfied: nvidia-cublas-cu12==12.6.4.1 in /usr/local/lib/python3.12/dist-packages (from torch) (12.6.4.1)
Requirement already satisfied: nvidia-cufft-cu12==11.3.0.4 in /usr/local/lib/python3.12/dist-packages (from torch) (11.3.0.4)
Requirement already satisfied: nvidia-curand-cu12==10.3.7.77 in /usr/local/lib/python3.12/dist-packages (from torch) (10.3.7.77)
Requirement already satisfied: nvidia-cusolver-cu12==11.7.1.2 in /usr/local/lib/python3.12/dist-packages (from torch) (11.7.1.2)
Requirement already satisfied: nvidia-cusparse-cu12==12.5.4.2 in /usr/local/lib/python3.12/dist-packages (from torch) (12.5.4.2)
Requirement already satisfied: nvidia-cusparselt-cu12==0.7.1 in /usr/local/lib/python3.12/dist-packages (from torch) (0.7.1)
Requirement already satisfied: nvidia-nccl-cu12==2.27.3 in /usr/local/lib/python3.12/dist-packages (from torch) (2.27.3)
Requirement already satisfied: nvidia-nvtx-cu12==12.6.77 in /usr/local/lib/python3.12/dist-packages (from torch) (12.6.77)
Requirement already satisfied: nvidia-nvjitlink-cu12==12.6.85 in /usr/local/lib/python3.12/dist-packages (from torch) (12.6.85)
Requirement already satisfied: nvidia-cufile-cu12==1.11.1.6 in /usr/local/lib/python3.12/dist-packages (from torch) (1.11.1.6)
Requirement already satisfied: triton==3.4.0 in /usr/local/lib/python3.12/dist-packages (from torch) (3.4.0)
Requirement already satisfied: regex!=2019.12.17 in /usr/local/lib/python3.12/dist-packages (from transformers) (2024.11.6)
Variables
Terminal

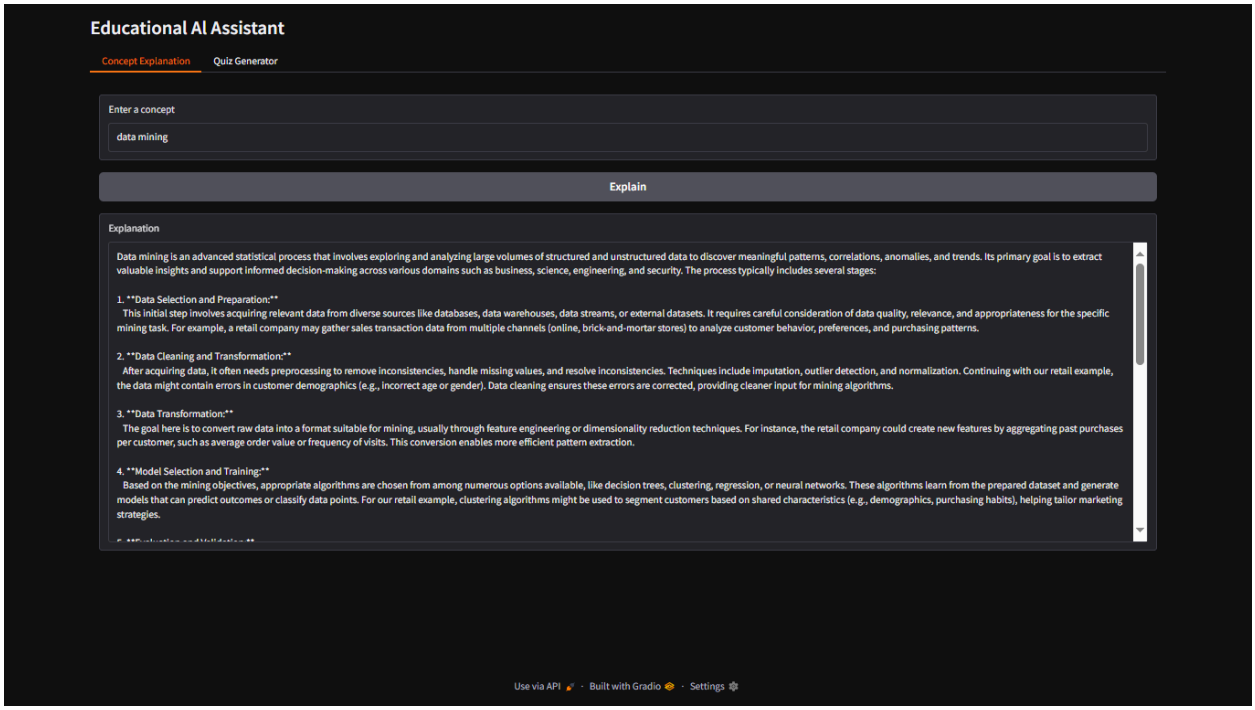

EduTutor_Ai.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands
+ Code
+ Text
Run all
Connect T4
Requirement already satisfied: requests in /usr/local/lib/python3.12/dist-packages (from transformers) (2.32.4)
Requirement already satisfied: tokenizers<=0.23.0,>=0.22.0 in /usr/local/lib/python3.12/dist-packages (from transformers) (0.22.0)
Requirement already satisfied: safetensors>=0.4.3 in /usr/local/lib/python3.12/dist-packages (from transformers) (0.6.2)
Requirement already satisfied: tqdm>=4.27 in /usr/local/lib/python3.12/dist-packages (from transformers) (4.67.1)
Requirement already satisfied: idna>=2.8 in /usr/local/lib/python3.12/dist-packages (from anyio<5.0,>=3.0->gradio) (3.10)
Requirement already satisfied: sniffio>=1.1 in /usr/local/lib/python3.12/dist-packages (from anyio<5.0,>=3.0->gradio) (1.3.1)
Requirement already satisfied: certifi in /usr/local/lib/python3.12/dist-packages (from httpx<1.0,>=0.24.1->gradio) (2025.8.3)
Requirement already satisfied: httpcore==1.* in /usr/local/lib/python3.12/dist-packages (from httpx<1.0,>=0.24.1->gradio) (1.0.9)
Requirement already satisfied: h11>=0.16 in /usr/local/lib/python3.12/dist-packages (from httpcore==1.*->httpx<1.0,>=0.24.1->gradio) (0.16.0)
Requirement already satisfied: hf-xet<2.0.0,>=1.1.3 in /usr/local/lib/python3.12/dist-packages (from huggingface-hub<1.0,>=0.33.5->gradio) (1.1.9)
Requirement already satisfied: python-dateutil>=2.8.2 in /usr/local/lib/python3.12/dist-packages (from pandas<3.0,>=1.0->gradio) (2.9.0.post0)
Requirement already satisfied: pytz>=2020.1 in /usr/local/lib/python3.12/dist-packages (from pandas<3.0,>=1.0->gradio) (2025.2)
Requirement already satisfied: tzdata>=2022.7 in /usr/local/lib/python3.12/dist-packages (from pandas<3.0,>=1.0->gradio) (2025.2)
Requirement already satisfied: annotated-types>=0.6.0 in /usr/local/lib/python3.12/dist-packages (from pydantic<2.12,>=2.0->gradio) (0.7.0)
Requirement already satisfied: pydantic-core==2.33.2 in /usr/local/lib/python3.12/dist-packages (from pydantic<2.12,>=2.0->gradio) (2.33.2)
Requirement already satisfied: typing-inspection>=0.4.0 in /usr/local/lib/python3.12/dist-packages (from pydantic<2.12,>=2.0->gradio) (0.4.1)
Requirement already satisfied: mpmath<1.4,>=1.1.0 in /usr/local/lib/python3.12/dist-packages (from sympy>=1.13.3->torch) (1.3.0)
Requirement already satisfied: click>=8.0.0 in /usr/local/lib/python3.12/dist-packages (from typer<1.0,>=0.12->gradio) (8.2.1)
Requirement already satisfied: shellingham>=1.3.0 in /usr/local/lib/python3.12/dist-packages (from typer<1.0,>=0.12->gradio) (1.5.4)
Requirement already satisfied: rich>=10.11.0 in /usr/local/lib/python3.12/dist-packages (from typer<1.0,>=0.12->gradio) (13.9.4)
Requirement already satisfied: charset_normalizer<4,>=2 in /usr/local/lib/python3.12/dist-packages (from requests->transformers) (3.4.3)
Requirement already satisfied: urllib3<3,>=1.21.1 in /usr/local/lib/python3.12/dist-packages (from requests->transformers) (2.5.0)
Requirement already satisfied: six>=1.5 in /usr/local/lib/python3.12/dist-packages (from python-dateutil>=2.8.2->pandas<3.0,>=1.0->gradio) (1.17.0)
Requirement already satisfied: markdown-it-py>=2.2.0 in /usr/local/lib/python3.12/dist-packages (from rich>=10.11.0->typer<1.0,>=0.12->gradio) (4.0.0)
Requirement already satisfied: pygments<3.0.0,>=2.13.0 in /usr/local/lib/python3.12/dist-packages (from rich>=10.11.0->typer<1.0,>=0.12->gradio) (2.19.2)
Requirement already satisfied: mdurl~=0.1 in /usr/local/lib/python3.12/dist-packages (from markdown-it-py>=2.2.0->rich>=10.11.0->typer<1.0,>=0.12->gradio)
[ ]
import gradio as gr
Variables
Terminal

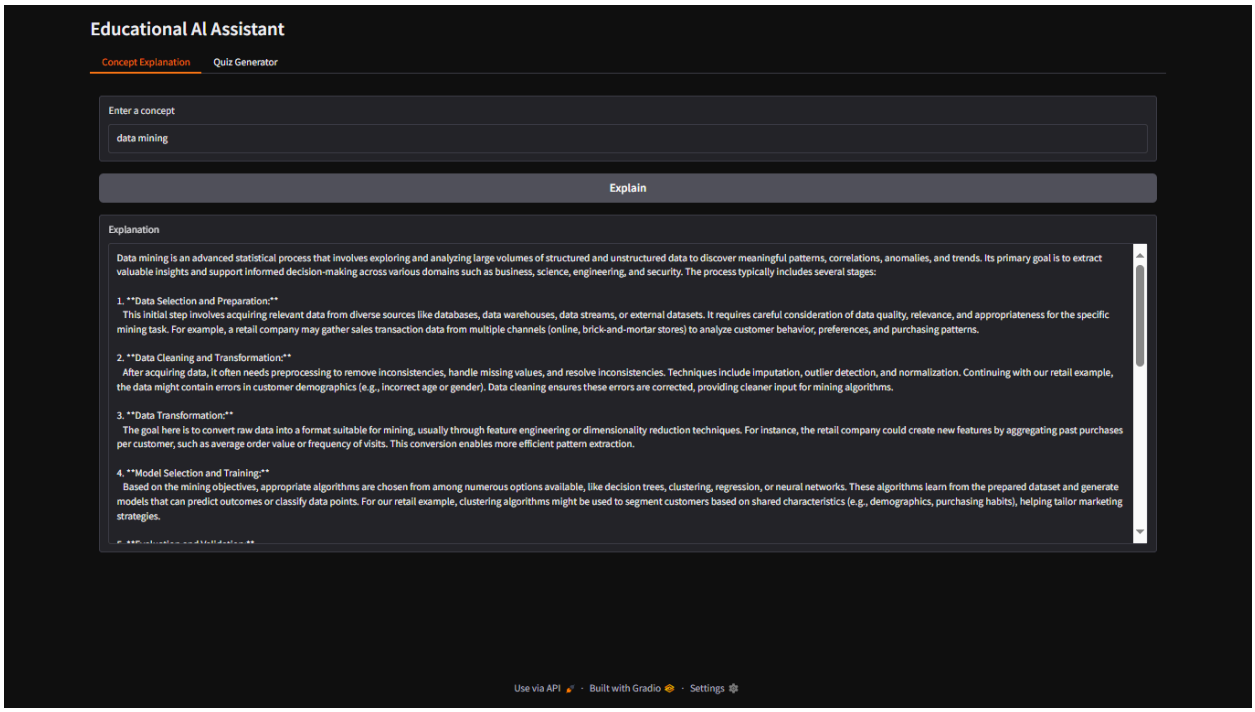

EduTutor_Ai.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands
+ Code
+ Text
Run all
Connect
[ ]
import gradio as gr
import torch
from transformers import AutoTokenizer, AutoModelForCausalLM
# Load model and tokenizer
model_name = "ibm-granite/granite-3.2-2b-instruct"
tokenizer = AutoTokenizer.from_pretrained(model_name)
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=512):
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(
Variables
Terminal

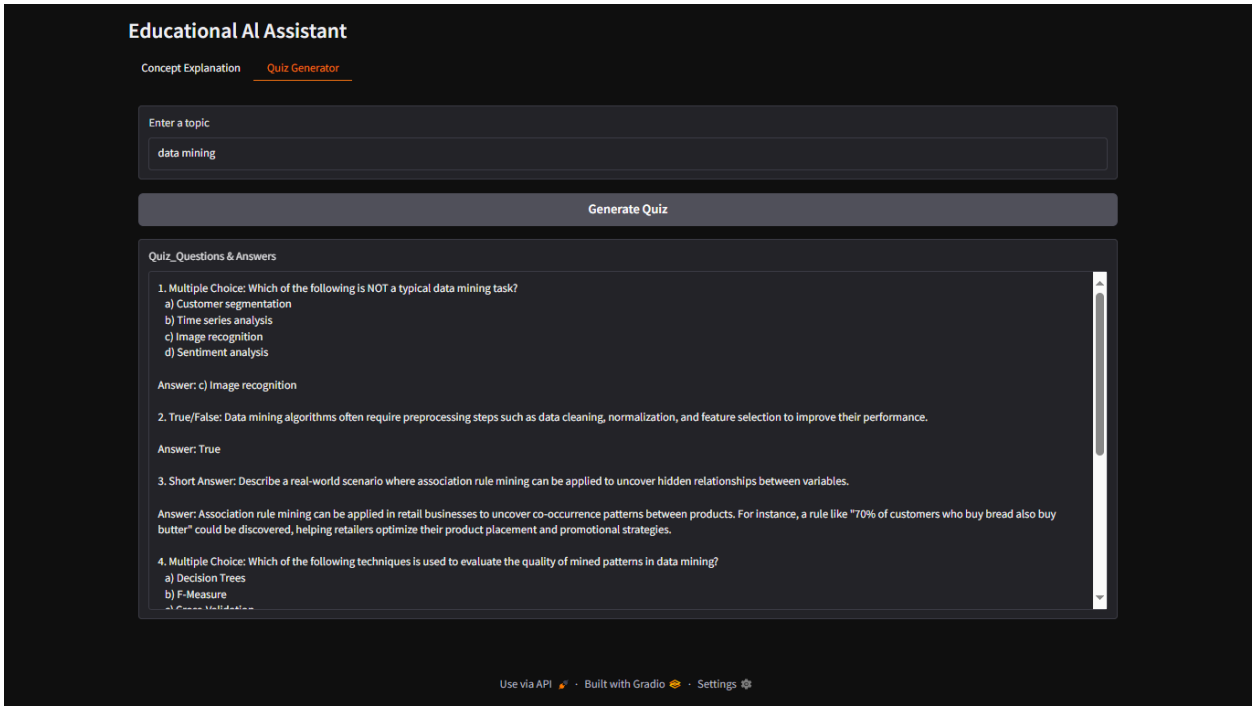
Output


https://colab.research.google.com/drive/1Tz2Fmz0vvyDqWuqVmXzvAu6rFtPoAU-l?authuser=1&usp=drive_open#scrollTo=z6pnF4EYEviq
EduTutor_Ai.ipynb
File Edit View Insert Runtime Tools Help
Share
Gemini
Commands + Code + Text Run all
Connect T4
tokenizer.json: 3.48M/? [00:00<00:00, 84.1MB/s]
added_tokens.json: 100% 87.0/87.0 [00:00<00:00, 7.38kB/s]
special_tokens_map.json: 100% 701/701 [00:00<00:00, 15.4kB/s]
config.json: 100% 786/786 [00:00<00:00, 20.1kB/s]
`torch_dtype` is deprecated! Use `dtype` instead!
model.safetensors.index.json: 29.8k/? [00:00<00:00, 2.60MB/s]
Fetching 2 files: 100% 2/2 [02:53<00:00, 173.62s/it]
model-00002-of-00002.safetensors: 100% 67.1M/67.1M [00:01<00:00, 20.0MB/s]
model-00001-of-00002.safetensors: 100% 5.00G/5.00G [02:53<00:00, 111MB/s]
Loading checkpoint shards: 100% 2/2 [00:23<00:00, 9.74s/it]
generation_config.json: 100% 137/137 [00:00<00:00, 14.9kB/s]
Colab notebook detected. To show errors in colab notebook, set debug=True in launch()
* Running on public URL: https://66b407a321560395e9.gradio.live
This share link expires in 1 week. For free permanent hosting and GPU upgrades, run `gradio deploy` from the terminal in the working directory to deploy t
gradio
Variables
Terminal


Educational Al Assistant
Concept Explanation
Quiz Generator
Enter a concept
e.g., Machine Learning
Explain
Explanation
Use via API · Built with Gradio · Settings


Educational AI Assistant
Concept Explanation
Quiz Generator
Enter a concept
data mining
Explain
Explanation
Data mining is an advanced statistical process that involves exploring and analyzing large volumes of structured and unstructured data to discover meaningful patterns, correlations, anomalies, and trends. Its primary goal is to extract valuable insights and support informed decision-making across various domains such as business, science, engineering, and security. The process typically includes several stages:
1. **Data Selection and Preparation:**
This initial step involves acquiring relevant data from diverse sources like databases, data warehouses, data streams, or external datasets. It requires careful consideration of data quality, relevance, and appropriateness for the specific mining task. For example, a retail company may gather sales transaction data from multiple channels (online, brick-and-mortar stores) to analyze customer behavior, preferences, and purchasing patterns.
2. **Data Cleaning and Transformation:**
After acquiring data, it often needs preprocessing to remove inconsistencies, handle missing values, and resolve inconsistencies. Techniques include imputation, outlier detection, and normalization. Continuing with our retail example, the data might contain errors in customer demographics (e.g., incorrect age or gender). Data cleaning ensures these errors are corrected, providing cleaner input for mining algorithms.
3. **Data Transformation:**
The goal here is to convert raw data into a format suitable for mining, usually through feature engineering or dimensionality reduction techniques. For instance, the retail company could create new features by aggregating past purchases per customer, such as average order value or frequency of visits. This conversion enables more efficient pattern extraction.
4. **Model Selection and Training:**
Based on the mining objectives, appropriate algorithms are chosen from among numerous options available, like decision trees, clustering, regression, or neural networks. These algorithms learn from the prepared dataset and generate models that can predict outcomes or classify data points. For our retail example, clustering algorithms might be used to segment customers based on shared characteristics (e.g., demographics, purchasing habits), helping tailor marketing strategies.
Use via API · Built with Gradio · Settings


Educational AI Assistant
Concept Explanation
Quiz Generator
Enter a concept
data mining
Explain
Explanation
Data mining is an advanced statistical process that involves exploring and analyzing large volumes of structured and unstructured data to discover meaningful patterns, correlations, anomalies, and trends. Its primary goal is to extract valuable insights and support informed decision-making across various domains such as business, science, engineering, and security. The process typically includes several stages:
1. **Data Selection and Preparation:**
This initial step involves acquiring relevant data from diverse sources like databases, data warehouses, data streams, or external datasets. It requires careful consideration of data quality, relevance, and appropriateness for the specific mining task. For example, a retail company may gather sales transaction data from multiple channels (online, brick-and-mortar stores) to analyze customer behavior, preferences, and purchasing patterns.
2. **Data Cleaning and Transformation:**
After acquiring data, it often needs preprocessing to remove inconsistencies, handle missing values, and resolve inconsistencies. Techniques include imputation, outlier detection, and normalization. Continuing with our retail example, the data might contain errors in customer demographics (e.g., incorrect age or gender). Data cleaning ensures these errors are corrected, providing cleaner input for mining algorithms.
3. **Data Transformation:**
The goal here is to convert raw data into a format suitable for mining, usually through feature engineering or dimensionality reduction techniques. For instance, the retail company could create new features by aggregating past purchases per customer, such as average order value or frequency of visits. This conversion enables more efficient pattern extraction.
4. **Model Selection and Training:**
Based on the mining objectives, appropriate algorithms are chosen from among numerous options available, like decision trees, clustering, regression, or neural networks. These algorithms learn from the prepared dataset and generate models that can predict outcomes or classify data points. For our retail example, clustering algorithms might be used to segment customers based on shared characteristics (e.g., demographics, purchasing habits), helping tailor marketing strategies.
Use via API · Built with Gradio · Settings



12. Known Issues:

- The system's performance may be slow when processing very large documents (e.g., PDFs over 50MB) due to the time required for embedding and search.
- The AI may occasionally provide irrelevant answers to highly specific or nuanced queries, which can be improved with more detailed prompt engineering.
- The current interface does not support real-time collaboration or multi-user sessions, limiting its use in a classroom setting.

13. Future Enhancements

- User Authentication and Profiles: Implement a robust user authentication system to allow for personalized user profiles, saving of study plans, and tracking of progress.
- Expanded File Format Support: Add support for additional document types such as DOCX, CSV, and plain text files to make the assistant more versatile.
- Advanced Analytics Dashboard: Develop a more comprehensive dashboard for educators, including detailed analytics on student engagement, most-asked questions, and knowledge gaps.