

**PROJECT ON**

**Enhancing Search Engine Relevance for Video Subtitles**



**PROJECT TEAM**

MOHAMMED. SUMAYYA

**&**

IN1240602

RAJA SEKHAR RAPAKA

IN1241079

**Project Overview:**

In today's digital era, the demand for accurate and efficient search engines has never been higher. With the exponential growth of online video content, ensuring the accessibility and relevance of video subtitles has become a crucial aspect of user experience. This project delves into enhancing the search relevance for video subtitles, with the overarching goal of improving the accessibility and usability of video content for users worldwide.

**Background:**

The digital content landscape is constantly evolving, with a vast array of video content being created and consumed daily. Search engines like Google have set high standards for search relevance, aiming to provide users with seamless and accurate search experiences. This project's background lies in addressing the challenge of improving search relevance specifically for video subtitles, recognizing the importance of making video content more accessible and user-friendly.

**Objective:**

The primary objective of this project is to develop an advanced search engine algorithm tailored to efficiently retrieve video subtitles based on user queries. This algorithm places a strong emphasis on the content of subtitles, leveraging sophisticated natural language processing (NLP) and machine learning (ML) techniques. By enhancing the relevance and accuracy of search results, the project aims to significantly improve the overall search experience for users seeking video content with specific subtitle requirements.

**Project Scope:**

**Covered Aspects:**

1. Semantic Search for Video Subtitles: The project covers the development of an advanced search algorithm specifically tailored for video subtitles, focusing on semantic analysis to improve search relevance.
2. Natural Language Processing (NLP): Utilizing NLP techniques to analyze and understand the content of video subtitles, enabling accurate retrieval based on user queries.
3. Machine Learning (ML) Integration: Incorporating ML models to enhance the search engine's performance in understanding user intent and improving search result accuracy.
4. User Interface (UI) Design: Designing an intuitive and user-friendly interface for users to interact with the semantic search engine efficiently.

**Key Steps followed in Building and Implementing the Search Engine:**

* Reading the Data from the Database
* Data Decoding: Decoded the data using encoding techniques like Latin-1 Encoder
* Data Preprocessing/Data cleaning
* Data chunking
* Embedding Generation
* Stored the data in a ChromaDB database for efficient retrieval.
* Information Retrieval Process: This encompasses the steps of User Query Processing, Query Embedding, Similarity Calculation, and Retrieval of Relevant Documents.

**Technologies Used:**

In this project, we utilized the following tools and frameworks:

* Python for overall application development
* Jupyter notebook
* Streamlit UI: Developed an intuitive user interface using Streamlit.
* ChromaDB : Configured and utilized ChromaDB for efficient storage and retrieval of subtitle data.
* NLP (Natural Language Processing)

**Database Overview:**

1.Database Name : eng\_subtitles\_database.db

Total Subtitle Files : 82,498

Source: opensubtitles.org

2. Subtitle Details:

Subtitle Types: Mostly from movies and TV series

Release Range: Between 1990 and 2024

3.Table Information:

Table Name: zipfiles

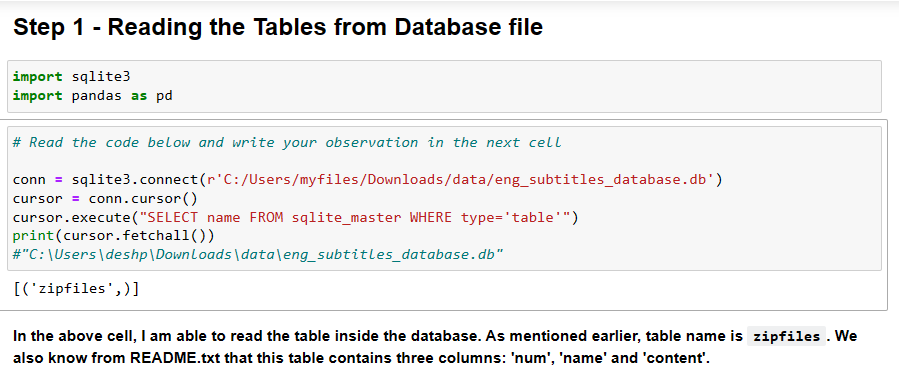
Columns:

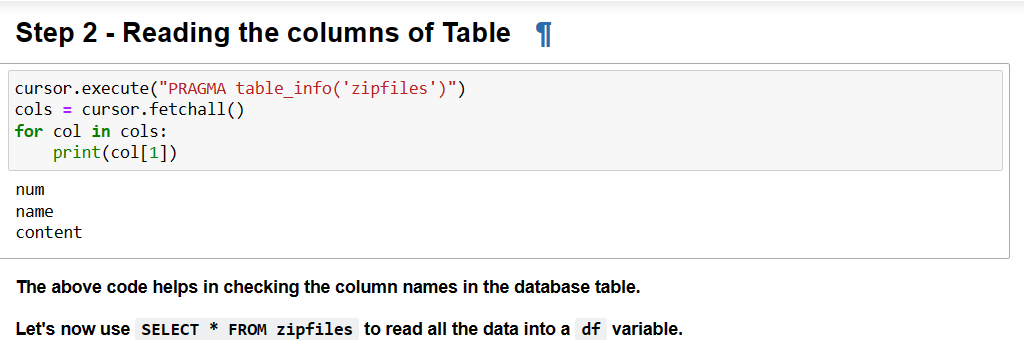
num : Unique Subtitle ID referencing www.opensubtitles.org

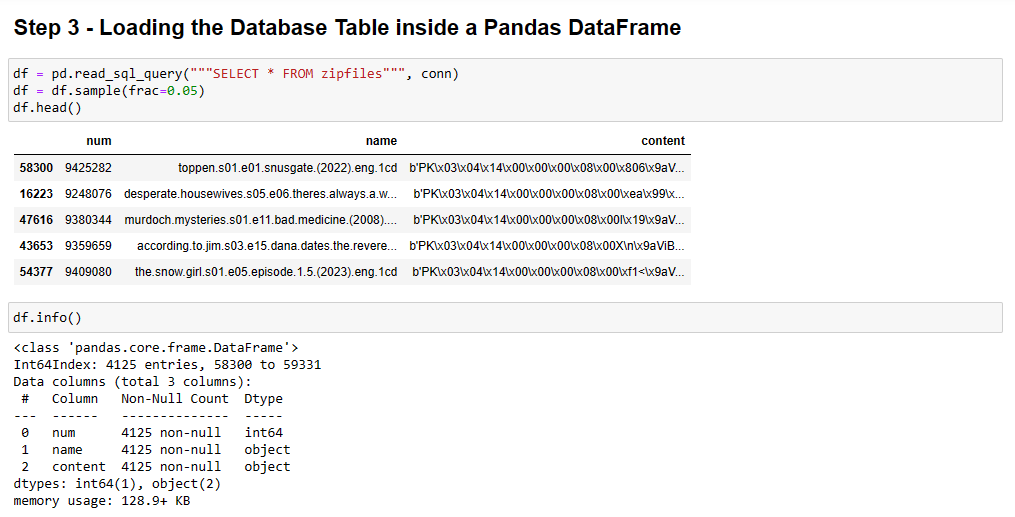
name : Subtitle File Name

content : Compressed Subtitle Files (binary data) encoded in 'latin-1'

* Established a connection to the SQLite database using the sqlite3 module in Python.
* Utilized the sqlite\_master table to retrieve information about the tables in the database.
* Identified the table named zipfiles as part of the database, which contains information about subtitles.
* Retrieved details about the columns in the zipfiles table, including the column names and their data types.



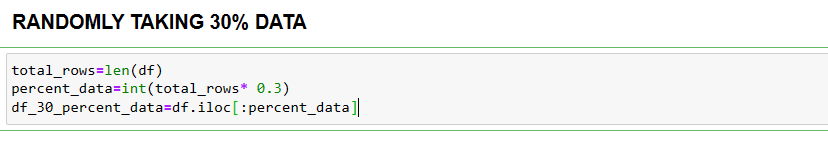




* This is the DataFrame obtained after reading from the database.
* As we can see that the “content” column contains data that is in encoded format so we have to decode the column using Latin-1 encoder.



* The "content" column contains data in an encoded format, possibly binary or compressed data.
* The "file\_content" column appears to be the decoded output of the "content" column, as it contains readable text and subtitles.
* We have sampled 30% of the data from the entire database for our analysis and project.

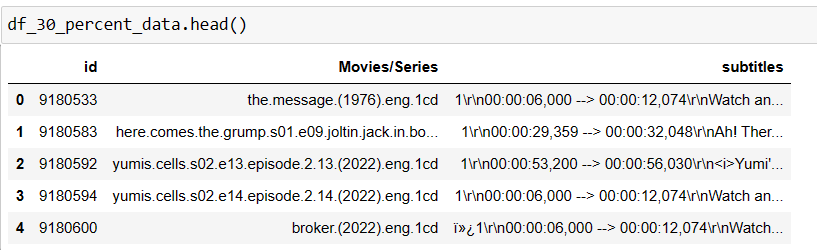


**Data Preprocessing:**

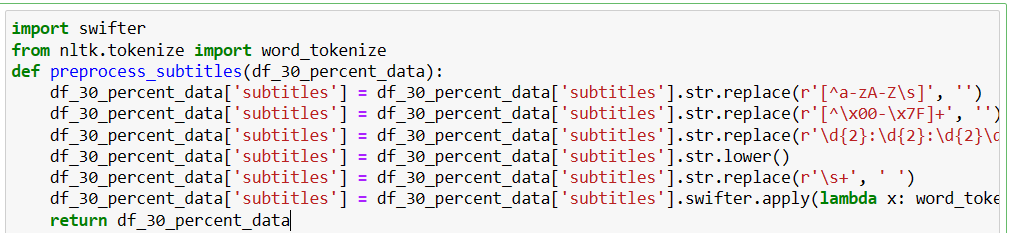
Preprocessing refers to a set of techniques and steps used to clean, transform, and prepare raw data for analysis or modeling.

In data preprocessing, we also perform specific tasks such as dropping unwanted columns and renaming columns to ensure our dataset is streamlined and structured for further analysis and modeling.

So, after dropping unwanted columns and renaming the column names, the resulting DataFrame is as follows:



We can see that the 'subtitles' column contains text data that needs to undergo preprocessing steps.

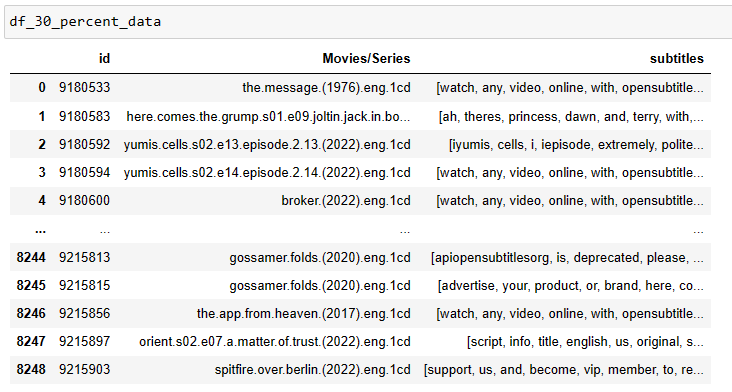


In the above figure, we can observe that the “preprocess\_text” function is designed to preprocess text data in a DataFrame column called 'file\_content'.

It performs the following steps:

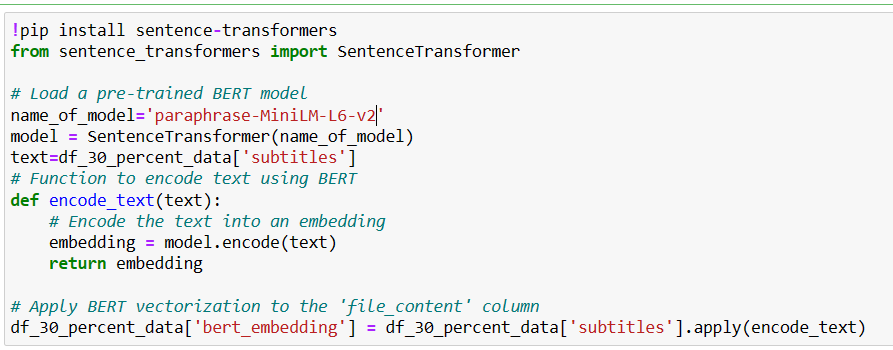
* Removes non-alphabetic characters using regular expressions.
* Removes non-ASCII characters using regular expressions.
* Removes timestamp patterns using regular expressions.
* Converts text to lowercase.
* Removes extra spaces.
* Tokenizes the text using the word\_tokenize function from the NLTK library.

Additionally, the function utilizes the “swifter” library for faster apply operations.



--Preprocessed DataFrame--

**Embedding Generation:**



After preprocessing the 'file\_content' column, we utilized the BERT-based

SentenceTransformers framework with the \*\*paraphrase-MiniLM-L6-v2\*\* model to generate embeddings. These embeddings encode semantic information, enabling us to capture the contextual meaning of the text for further analysis and natural language processing tasks.

* 🤔Do you ever wonder why we opted for using BERT to vectorize and obtain embeddings rather than relying on BOW/TFIDF❓
* The answer lies in our goal of building a semantic search engine.
* There are two types of search engines we can build: a keyword-based search engine and a semantic search engine. We are focusing on building a semantic search engine that leverages BERT-based SentenceTransformers to capture semantic relationships and contextual meaning in the text data.

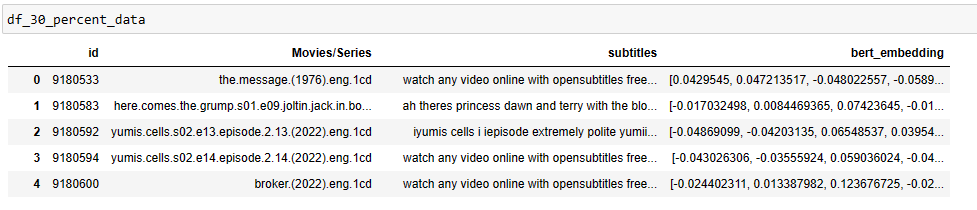
**Keyword based vs Semantic Search Engines:**

* Keyword Based Search Engine: These search engines rely heavily on exact keyword matches between the user query and the indexed documents.
* Semantic Search Engines: Semantic search engines go beyond simple keyword matching to understand the meaning and context of user queries and documents.

**Comparison:** While keyword-based search engines focus primarily on matching exact keywords in documents, semantic-based search engines aim to understand the deeper meaning and context of user queries to deliver more relevant and meaningful search results.

The main takeaway points regarding the use of BOW/TFIDF and BERT-based SentenceTransformers are:

1. **BOW / TFIDF** to generate sparse vector representations. Note that this will only help you to build a **Keyword Based Search Engine.**
2. **BERT** based “SentenceTransformers” to generate embeddings which encode semantic information. This can help us build a **Semantic Search Engine**.



--Embedded DataFrame--

* Choosing “paraphrase-MiniLM-L6-v2” model : Unveiling Its Advantages

1. The “paraphrase-MiniLM-L6-v2” model demonstrates superior performance in semantic understanding tasks, surpassing many other models in accuracy and efficiency.
2. It excels in capturing nuanced semantic relationships, providing embeddings that encode rich contextual meaning and semantic similarity.
3. This model is versatile and adaptable to various NLP tasks, making it a valuable choice for a wide range of applications beyond semantic search engines.
4. Despite its advanced capabilities, paraphrase-MiniLM-L6-v2 maintains efficient computation, ensuring scalability and practicality for real-world projects.

**Data Chunking:**

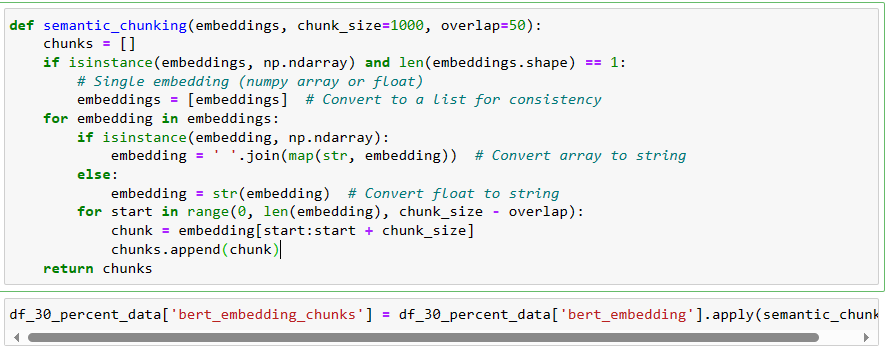
Data chunking involves breaking large datasets into smaller, manageable parts to improve processing efficiency and enable tasks like storage, retrieval, and analysis.

**Why Data Chunking??**

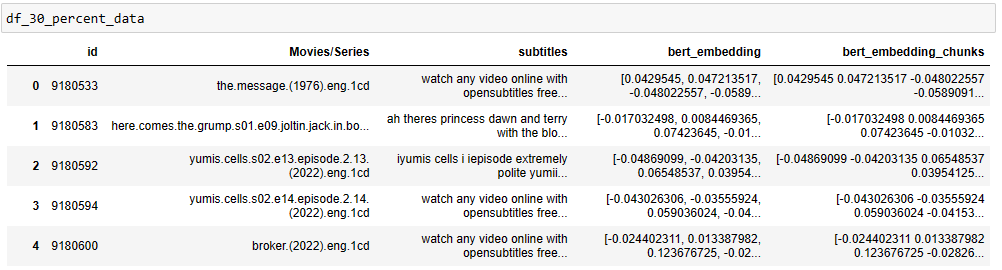
Data chunking is essential when dealing with large documents to prevent information loss during embedding processes.

Embedding an entire document as a single vector is impractical due to the risk of losing crucial contextual details. By dividing large documents into smaller, more manageable chunks, data chunking enables efficient embedding processes while preserving important information.

This strategy ensures that the resulting embeddings maintain the context and coherence of the original document, enhancing the quality and accuracy of downstream tasks such as semantic search engines.



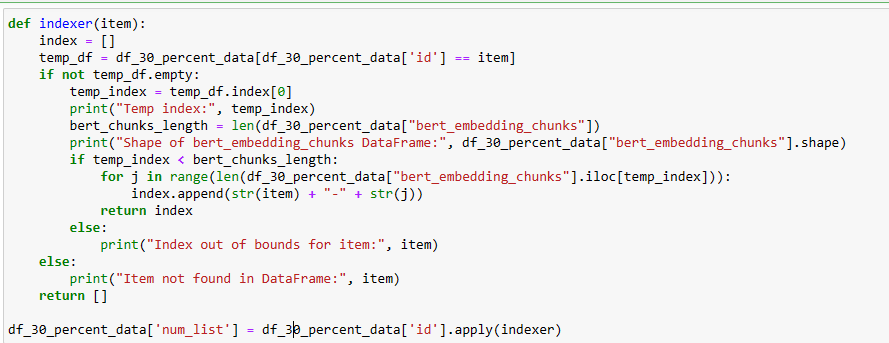
* The above code defines a function “semantic\_chunking” that takes embeddings (which could be a single embedding or a list of embeddings) along with optional parameters for chunk size and overlap.
* It converts the embeddings into chunks of specified size, with an overlap to ensure continuity.
* The function handles various input types, such as numpy arrays or floats, by converting them into strings for chunking.
* The resulting list chunks contains the segmented embeddings, suitable for semantic analysis or processing large datasets in manageable portions.



--Chunked DataFrame—

**Preparing Indexes:**

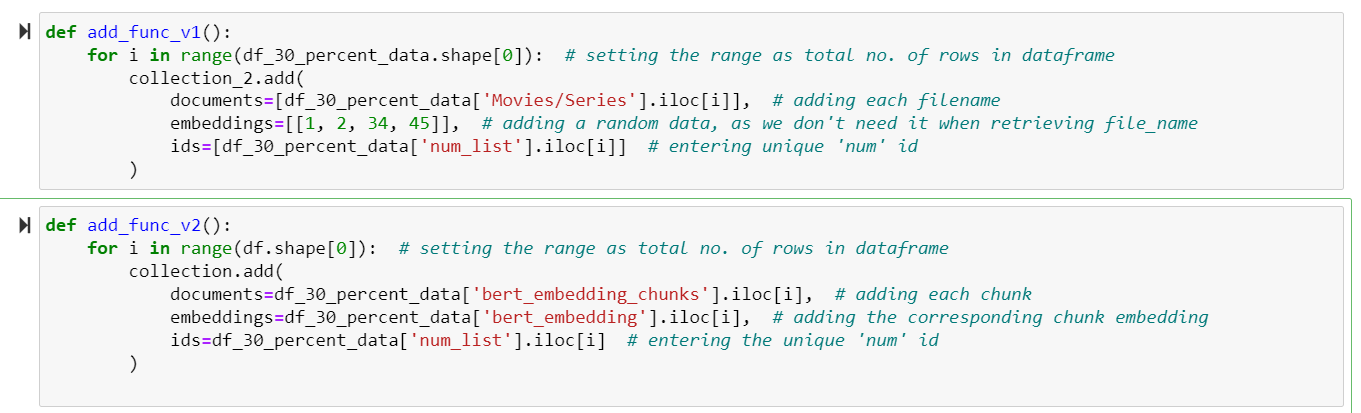
To handle multiple chunks per subtitle file and ensure accurate retrieval of data based on user queries, we developed an "indexer" function. This function uses the 'num' column as a reference to find the corresponding row in the DataFrame. It then generates unique identifiers for elements within a chunk by combining the item with a sequential index separated by a hyphen. This approach allows us to store and retrieve data effectively, even when dealing with multiple chunks per subtitle file.



**Storing the data in a ChromaDB database:**



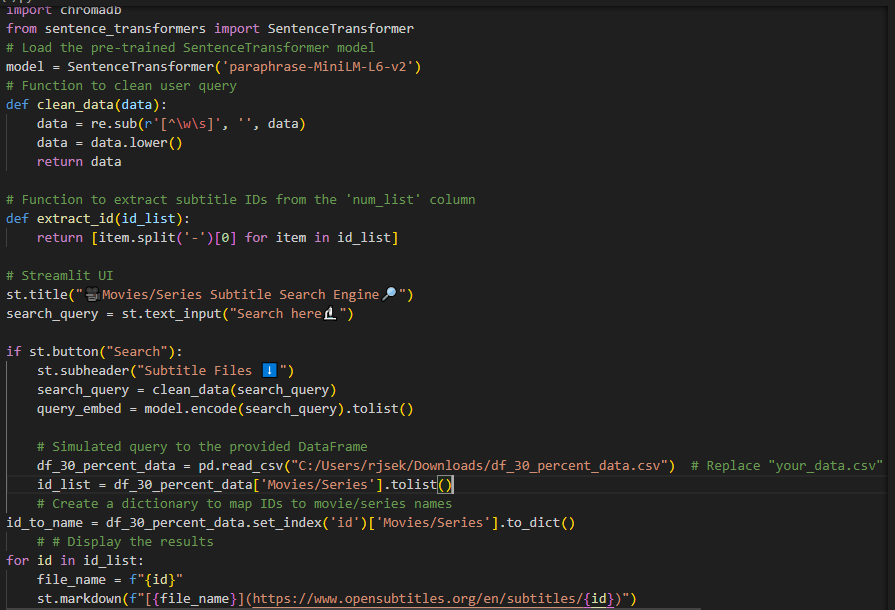
The above code snippet imports the “chromadb” library and initializes a persistent client for accessing a ChromaDB database located at the specified path. It then creates two collections named "subtitles" and "titles" within the database, each configured with metadata specifying the space for similarity search (in this case, cosine similarity) using the HNSW indexing method (defaulting to L2 space if not specified). These collections are intended to store and index data for efficient similarity search operations based on the specified similarity metric.



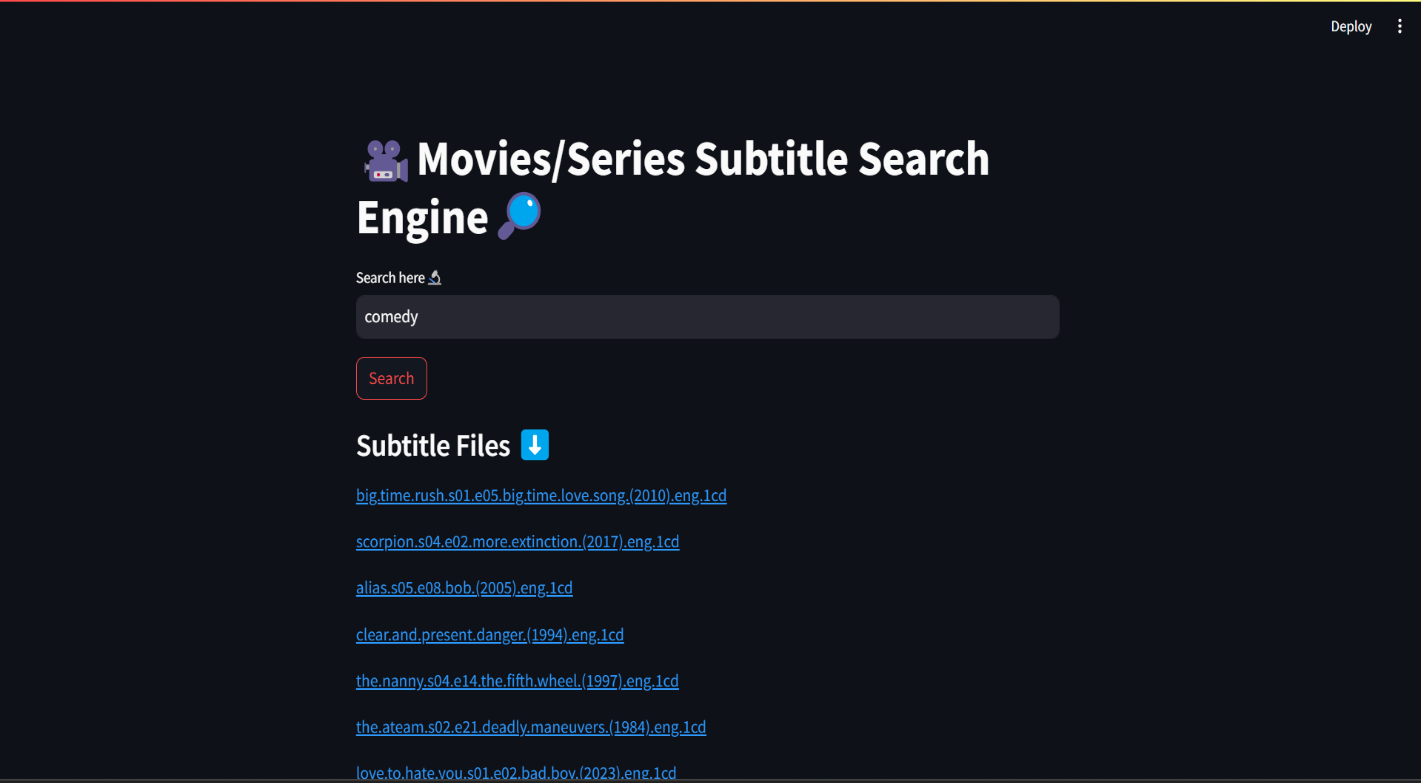
The add\_func\_v1 function iterates through each row of the DataFrame df\_30\_percent\_data, adding filenames as documents, random data as embeddings (not used during retrieval), and unique IDs from the 'num\_list' column into the "titles" collection in ChromaDB.

On the other hand, add\_func\_v2 iterates through another DataFrame df, adding chunks of BERT embeddings as documents, corresponding BERT embeddings, and unique IDs from the 'num\_list' column into the "subtitles" collection in ChromaDB. These functions prepare data for efficient similarity search and retrieval based on embeddings and unique identifiers.

**Building the Web Application Frontend Using Streamlit Framework:**



This Streamlit app code defines a user interface for a Movies/Series Subtitle Search Engine. It loads a pre-trained SentenceTransformer model ('paraphrase-MiniLM-L6-v2') for semantic embeddings. Users can input search queries, which are then cleaned and converted into embeddings using the model. The app simulates a query against a provided DataFrame (df\_30\_percent\_data.csv), retrieves relevant IDs from the DataFrame, and displays clickable links to subtitle files on OpenSubtitles.org based on the search results.

* Here are the visuals showcasing our project's final output—the Movies/Series Subtitle Search Engine in action. 



At last, this project has successfully developed a robust Movies/Series Subtitle Search Engine, leveraging advanced NLP techniques and a user-friendly interface. It not only enhances the accessibility of subtitles but also empowers users to explore and enjoy cinematic content with ease. The project's journey reflects a commitment to innovation and practical solutions in the realm of media and entertainment.

∙∙∙

**Conclusion:**

In conclusion, the development journey of the Movie Subtitle Search Engine has been both enlightening and challenging, its core functionality holds promise for providing a valuable service to users seeking subtitles. This project aimed to create a user-friendly platform where users could search for subtitles using natural language queries. Throughout the development process, several key insights and outcomes emerged.

**Problem Statement and Motivation:**

The project stemmed from the recognition of the growing need for efficient subtitle search tools. With the increasing popularity of movies and TV shows across diverse demographics and languages, finding accurate subtitles has become essential for a seamless viewing experience

**Challenges Faced:**

**Database Configuration:** Initial challenges were encountered in setting up and configuring the ChromaDB database, leading to errors related to file creation and schema mismatch.

**Library Compatibility:** Compatibility issues arose between the ChromaDB library and the expected database schema, highlighting the importance of version compatibility and documentation clarity.

**Future Directions:**

**Addressing Technical Issues:** Resolving database configuration errors and ensuring compatibility with library requirements is crucial for the project's success.

**Enhancing User Experience:** Implementing additional features such as filtering options, multi-language support, and user feedback mechanisms can enhance the overall user experience.

**Scaling and Optimization:** Optimizing database queries and scaling the application infrastructure can improve performance and accommodate a larger user base.

**References**

The following articles, videos and documentations were instrumental during this project:

Pradip Nichite. (2023). Semantic Search with Open-Source Vector DB: Chroma DB |

Pinecone Alternative | Code. Www.youtube.com.

https://www.youtube.com/watch?v=eCCHDxMaFIk

Bakharia, A. (2020, January 19). Quick Semantic Search using Siamese-BERT Networks.

Medium.

https://towardsdatascience.com/quick-semantic-search-using-siamese-bert-networks

-1052e7b4df1

Bricken, A. (2021a). Does BERT Need Clean Data? Part 1: Data Cleaning. Medium.

https://towardsdatascience.com/part-1-data-cleaning-does-bert-need-clean-data-6a50

c9c6e9fd

Chroma. the AI-native open-source embedding database. Www.trychroma.com.

https://www.trychroma.com/

Bricken, A. (2021b). Does BERT Need Clean Data? Part 2: Classification. Medium.

https://towardsdatascience.com/does-bert-need-clean-data-part-2-classification-d29a

df9f745a

Hoffman, H. (2023). Embeddings and Vector Databases With ChromaDB – Real Python.

Realpython.com. https://realpython.com/chromadb-vector-database/

Kamradt, G. ChunkViz. Chunkviz.up.railway.app. Retrieved 2024, from

https://chunkviz.up.railway.app/#explanation

NeuralNine. (2023). Parallelize Python Tasks with Joblib. Www.youtube.com.

https://www.youtube.com/watch?v=Dm4up8\_zJdo

SBERT.net. (2024). Pretrained Models — Sentence-Transformers documentation.

Www.sbert.net. https://www.sbert.net/docs/pretrained\_models.html

Todeschini, S. (2023). How to Chunk Text Data — A Comparative Analysis. Medium.

https://towardsdatascience.com/how-to-chunk-text-data-a-comparative-analysis-385

8c4a0997a