INNOMATICS®
RESEARCH LABS

INNOVATION. AUTOMATION. ANALYTICS

PROJECT ON

Enhancing Search Engine Relevance for Video Subtitles

By: NAGESH CHATURE

Table of Content

Abstract:

In the rapidly evolving landscape of digital content consumption, accessibility to video content is increasingly important. This project addresses the need for enhancing video accessibility by improving the search relevance of subtitles, particularly focusing on the content within subtitles. The objective is to develop an advanced search engine algorithm that efficiently retrieves subtitles based on user queries, emphasizing natural language processing and machine learning techniques to enhance relevance and accuracy.

This project uses between semantic search engines over traditional keyword-based search engines. As keyword-based engines rely on exact matches between user queries and indexed documents, semantic search engines delve deeper into understanding the meaning and context of queries and documents.

The core logic involves three key steps: data preprocessing, vectorization of subtitle documents and user queries, and cosine similarity calculation. Data preprocessing includes cleaning steps such as removing timestamps to ensure accurate analysis. Vectorization converts textual data into numerical representations, facilitating comparison. Cosine similarity calculation measures the relevance of documents to the user's query based on vector representations.

By employing semantic search techniques, this project aims to provide more meaningful and relevant search results for video subtitles, enhancing the accessibility of subtitle content for all users.

1. Read the given data:

Database contains a sample of 82498 subtitle files from opensubtitles.org. Most of the subtitles are of movies and tv-series which were released after 1990 and before 2024.

Database File Name: eng_subtitles_database.db

Database contains a table called 'zipfiles' with three columns.

1. num: Unique Subtitle ID reference for www.opensubtitles.org

2. name: Subtitle File Name

3. content: Subtitle file were compressed and stored as a binary using 'latin-1' encoding.

You can use 'num' to get more details about each subtitle by going to the following link: https://www.opensubtitles.org/en/subtitles/{num} **Replace {num} with Unique Subtitle ID.

`

In [6]:
```
import sqlite3
import pandas as pd
import zipfile
import io
import re

database_path = r"D:\INNOMATICS INTERN\final_project\files_folder\eng_subtitles_database.db"
conn = sqlite3.connect(database_path)
cursor = conn.cursor()

try:
    cursor.execute("SELECT num, name, content FROM zipfiles")
    rows = cursor.fetchall()
    df = pd.DataFrame(rows, columns=['subtitle_id', 'subtitle_name','subtitle_content'])

    print("Subtitle files fetched, decoded, cleaned, and stored in DataFrame successfully.")


except sqlite3.Error as e:
    print(f"SQLite error: {e}")
except Exception as e:
    print(f"Error: {e}")
finally:
    cursor.close()
    conn.close()
```

```
Subtitle files fetched, decoded, cleaned, and stored in DataFrame successfully.
```

This Python script demonstrates the process of fetching subtitle data from a SQLite database, decoding it, cleaning it, and storing it in a Pandas DataFrame.

- The script establishes a connection to the SQLite database containing subtitle data using the sqlite3.connect() function. The database path is specified as database_path.
- A SQL query is executed to select data from the "zipfiles" table within the SQLite database. The query retrieves columns named "num", "name", and "content", representing subtitle ID, name, and content respectively.
- The fetched rows from the query result are stored in variable named rows.
- Using the Pandas library, a DataFrame named df is created from the fetched rows. The DataFrame consists of three columns: "subtitle_id", "subtitle_name", and "subtitle_content", corresponding to the subtitle data fetched from the database.
- Upon successful retrieval, decoding, cleaning, and storage of subtitle data in the DataFrame, a success message is printed to indicate the completion of the process.
- Exception handling is implemented to catch any potential errors during the execution of the script. This includes handling SQLite errors (sqlite3.Error) and other general exceptions (Exception).
- Finally, the SQLite cursor and connection are closed using the cursor.close() and conn.close() functions respectively, ensuring proper cleanup and release of database resources.

In [7]:
```
df.shape
```

Out[7]:
```
(82498, 3)
```

In [8]:
```
df.head()
```

Out[8]:

	subtitle_id	subtitle_name	subtitle_content
0	9180533	the.message.(1976).eng.1cd	b'PK\x03\x04\x14\x00\x00\x00\x08\x00\x1c\xa9\x...
1	9180583	here.comes.the.grump.s01.e09.joltin.jack.in.bo...	b'PK\x03\x04\x14\x00\x00\x00\x08\x00\x17\xb9\x...
2	9180592	yumis.cells.s02.e13.episode.2.13.(2022).eng.1cd	b'PK\x03\x04\x14\x00\x00\x00\x08\x00L\xb9\x99V...
3	9180594	yumis.cells.s02.e14.episode.2.14.(2022).eng.1cd	b'PK\x03\x04\x14\x00\x00\x00\x08\x00U\xa9\x99V...
4	9180600	broker.(2022).eng.1cd	b'PK\x03\x04\x14\x00\x00\x00\x08\x001\xa9\x99V...

`

After fetching the data from the SQLite database and storing it in a Pandas DataFrame. The DataFrame df has 82,498 rows and 3 columns.subtitle_id: Represents the unique identifier for each subtitle. subtitle_name: Contains the name or title of each subtitle file. subtitle_content: Holds the content or text of each subtitle. The subtitle_content column contains the binary content of the subtitle files, as indicated by the prefix 'b'. This content likely needs further processing or decoding to extract meaningful text information.

```
In [11]: import zipfile
         import io

         def extract_content(content):
             with io.BytesIO(content) as bio:
                 with zipfile.ZipFile(bio, "r") as zipf:
                     file_list = zipf.namelist()
                     for file_name in file_list:
                         with zipf.open(file_name) as file:
                             content = file.read().decode("latin-1")
                             return content

         df['subtitle_content'] = df['subtitle_content'].apply(extract_content)
```

1(a). Unzipping the content and decoding using latin-1

The extract_content function is defined with one parameter, content, representing the binary content of a zip archive containing subtitle files. The function utilizes the io.BytesIO class to create an in-memory binary stream (bio) from the provided content. Within the context of a zipfile.ZipFile object (zipf), created from the binary stream, the function iterates through the list of file names (file_list) contained in the zip archive. For each file in the archive, the function opens it (with zipf.open(file_name) as file) and reads its content (content = file.read().decode("latin-1")). The content is decoded using the "latin-1" encoding, assuming it represents text data. The extracted content of the first file encountered in the zip archive is returned. The extract_content function is applied to the subtitle_content column of the DataFrame df using the apply method. This enables the function to be executed on each row of the Data Frame, extracting the content of zip archives stored in the subtitle_content column.

```
In [7]: df.shape
Out[7]: (82498, 3)
```

Working with a Data Frame of shape (824925, 3) where one of the row contains more than 100,000 tokens can consume a significant amount of computer resources, especially in terms of memory and processing power. Because of limited compute resources, only random 30% of the data is used.

`

```
In [12]: new_df = df.sample(frac = 0.3, random_state = 42)

In [13]: new_df.shape
Out[13]: (24749, 3)

In [14]: new_df.head()
```

Out[14]:

	subtitle_id	subtitle_name	subtitle_content
17262	9251120	maybe.this.time.(2014).eng.1cd	ï»¿1\r\n00:00:06,000 --> 00:00:12,074\r\nWatch...
7294	9211589	down.the.shore.s01.e10.and.justice.for.all.(19...	1\r\n00:00:09,275 --> 00:00:11,876\r\n¶ Oh, I ...
47707	9380845	uncontrollably.fond.s01.e07.heartache.(2016).e...	1\r\n00:00:07,140 --> 00:00:14,220\r\n<i>Timin...
29914	9301436	screen.two.s13.e04.the.precious.blood.(1996).e...	1\r\n00:00:06,133 --> 00:00:08,900\r\n[etherea...
54266	9408707	battlebots.(2015).eng.1cd	ï»¿1\r\n00:00:01,480 --> 00:00:03,570\r\n[Chri...

```
In [16]: new_df['subtitle_content'][0]
```

```
Out[16]: 'ï»¿1\r\n00:00:06,000 --> 00:00:12,074\r\nWatch any video online with Open-SUBTITLES\r\nFree Browser extension: osdb.link/ext
\r\n\r\n2\r\n00:00:37,328 --> 00:00:39,706\r\n<i>It could\'ve been\r\njust another summer.</i>\r\n\r\n3\r\n00:00:40,790 --> 0
0:00:43,042\r\n<i>But as I set foot on the sand,</i>\r\n\r\n4\r\n00:00:43,209 --> 00:00:46,212\r\n<i>that summer\r\nsuddenly
felt different.</i>\r\n\r\n5\r\n00:00:55,221 --> 00:00:56,973\r\n<i>Like it was going to be the summer</i>\r\n\r\n6\r\n00:00:
57,098 --> 00:00:59,142\r\n<i>that would change my life.</i>\r\n\r\n7\r\n00:00:59,350 --> 00:01:01,770\r\n<i>The summer of fr
eedom.</i>\r\n\r\n8\r\n00:01:02,562 --> 00:01:05,607\r\n<i>The summer of\r\nendless possibilities.</i>\r\n\r\n9\r\n00:01:06,2
74 --> 00:01:09,402\r\n<i>The summer of 2007.</i>\r\n\r\n10\r\n00:01:16,493 --> 00:01:18,036\r\nOoh, aah!\r\n\r\n11\r\n00:01:
24,459 --> 00:01:26,169\r\nOoh, oh!\r\n\r\n12\r\n00:01:26,377 --> 00:01:28,254\r\n<i>â\x99ª Oh, oh, ooh â\x99ª</i>\r\n\r\n13
\r\n00:01:37,514 --> 00:01:40,683\r\n<i>That was the summer of you and me.</i>\r\n\r\n14\r\n00:01:40,809 --> 00:01:43,520\r\n
You\'re quite the dancer.\r\nWhy did you stop?\r\n\r\n15\r\n00:01:44,103 --> 00:01:45,396\r\nCome on! Keep dancing!\r\n\r\n16
\r\n00:01:53,863 --> 00:01:55,365\r\nWhatever!\r\n\r\n17\r\n00:01:56,366 --> 00:01:57,826\r\nI\'m kidding. Don\'t get mad.\r
\n\r\n18\r\n00:02:10,713 --> 00:02:11,713\r\nHuh?\r\n\r\n19\r\n00:02:13,424 --> 00:02:14,424\r\nWhat...\r\n\r\n20\r\n00:02:1
7,887 --> 00:02:18,887\r\nHey!\r\n\r\n21\r\n00:02:24,394 --> 00:02:26,187\r\nI\'m just going to get my towel.\r\n\r\n22\r\n0
0:02:26,312 --> 00:02:27,312\r\nWhat?\r\n\r\n23\r\n00:02:28,398 --> 00:02:30,107\r\nStop that!\r\n\r\n24\r\n00:02:33,027 -->
00:02:34,444\r\nYou thought I was gonna kiss you.\r\n\r\n25\r\n00:02:34,445 --> 00:02:36,573\r\nNo! Excuse me!\r\n\r\n26\r\n0
0:02:37,949 --> 00:02:39,868\r\nI wanna kiss you but not just yet.\r\n\r\n27\r\n00:02:41,161 --> 00:02:42,619\r\nWhat do you
mean "not yet"?\r\n\r\n28\r\n00:02:42,620 --> 00:02:43,955\r\nOnly when you\'re my girl.\r\n\r\n29\r\n00:02:44,914 --> 00:02:
47,875\r\n- What do you mean your girl?\r\n- My girlfriend, Miss.\r\n\r\n30\r\n00:02:47,876 --> 00:02:50,043\r\nAs if! You wi
sh!\r\n\r\n31\r\n00:02:50,044 --> 00:02:52,838\r\nAnd don\'t call me miss.\r\n\r\n32\r\n00:02:52,839 --> 00:02:54,923\r\nDon
```

```
In [17]: def clean_text(text):
             text = re.sub(r'\d{2}:\d{2}:\d{2},\d{3} --> \d{2}:\d{2}:\d{2},\d{3}\r\n', '', text)
```

2. Cleaning:

Text data is a subtitle contect which has time stamps, HTML tags and lots of noises. Cleaning has to be done on the text data, as cleaning text data is an essential preprocessing step in natural language processing (NLP) tasks, especially before vectorization. Cleaning the text aims to remove noise, irrelevant information, and inconsistencies from the text, making it more suitable for downstream processing.

```
In [17]: def clean_text(text):
             text = re.sub(r'\d{2}:\d{2}:\d{2},\d{3} --> \d{2}:\d{2}:\d{2},\d{3}\r\n', '', text)
             text = re.sub(r'\r\n', ' ', text)
             text = re.sub(r'<[^>]+>', '', text)
             text = re.sub(r'[^a-zA-Z\s]', '', text)
             text = re.sub(r'\s+', ' ', text)
             text = text.strip()
             return text
         new_df['subtitle_content'] = new_df['subtitle_content'].apply(clean_text)

In [18]: new_df['subtitle_content'][0]
```

```
Out[18]: 'Watch any video online with OpenSUBTITLES Free Browser extension osdblinkext It couldve been just another summer But as I se
t foot on the sand that summer suddenly felt different Like it was going to be the summer that would change my life The summe
r of freedom The summer of endless possibilities The summer of Ooh aah Ooh oh Oh oh ooh That was the summer of you and me You
re quite the dancer Why did you stop Come on Keep dancing Whatever Im kidding Dont get mad Huh What Hey Im just going to get
my towel What Stop that You thought I was gonna kiss you No Excuse me I wanna kiss you but not just yet What do you mean not
yet Only when youre my girl What do you mean your girl My girlfriend Miss As if You wish And dont call me miss Dont pretend t
o be a gentleman when youre clearly not So what should I call you Rude Snob Bitch And you Douche Handsome Conceited Just like
you Huh Jerk Exactly your type Leave me alone Steph Aha Steph Ill just call you Tep Remove the S and the F By the way Im Toni
o Will you still be here tomorrow Dont leave yet Im going to court you I chose to walk away from you But fate had a different
plan Councilor Were teaching basic English literacy Well be teaching the children how to read and write in English Is that so
Yes How long will this program run If its okay the entire summer I thought I could escape you But you somehow found me again
Tep What are you doing here You couldnt resist me huh So youre courting me instead Excuse me Im not here to court you Oh So y
oure here to be courted Yeah No Uncle Tonio shes the one who will teach us this summer Dont believe that story Thats just her
excuse to get me to date her Gramps Uncle Erning Aunt Elma this is Tep My suitor Approved Approved You passed Mmm I dont appr
ove Excuse us Well be going now All right my dear Thank you sir Well go ahead Excuse me please Oh no Tep Tep are you okay I s
aid Excuse me Were not yet a couple and were already fighting He has a point Oh Tonio Seventy Snacks Seven Seven Silvery Silv
```

`

1. The code defines a function clean_text(text) that applies several cleaning steps to the text data:

2. re.sub(r'\d{2}:\d{2}:\d{2},\d{3} --> \d{2}:\d{2}:\d{2},\d{3}\r\n', '', text) This regular expression removes timestamps in the format "hh:mm:ss,mmm --> hh:mm:ss,mmm" along with the carriage return and newline characters (\r\n).

3. re.sub(r'\r\n', ' ', text) This line replaces carriage return and newline characters (\r\n) with a space. This step helps ensure that the text remains coherent and well-structured after removing the timestamps.

4. re.sub(r'<[^>]+>', '', text) This regular expression removes HTML tags from the text. HTML tags are often present in text data obtained from web sources and may not carry meaningful information for NLP tasks.

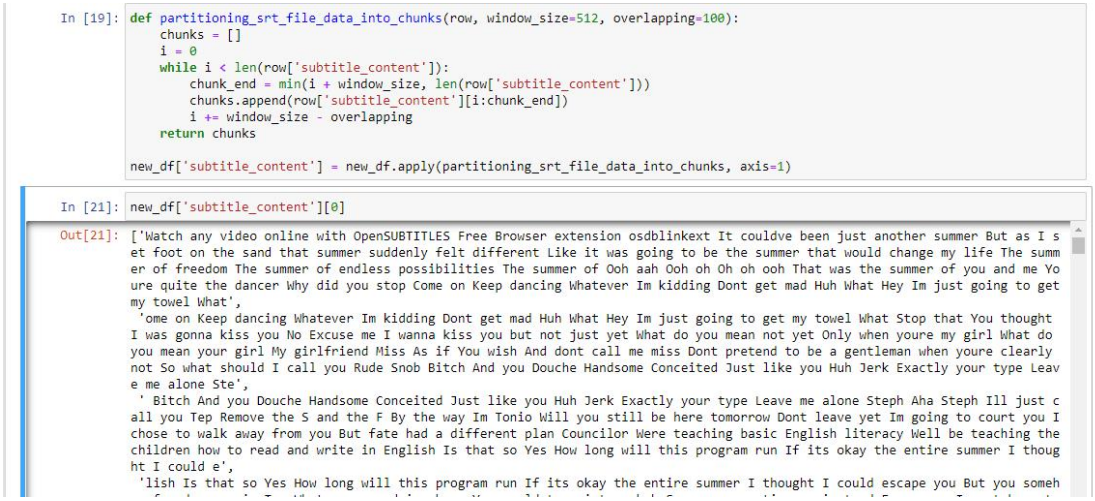
5. re.sub(r'[^a-zA-Z\s]', '', text) This line removes any characters that are not alphabetic or whitespace. It effectively eliminates punctuation and special characters from the text, retaining only letters and spaces.

6. re.sub (r'\s+', ' ', text) This regular expression replaces multiple consecutive whitespace characters with a single space. It helps normalize the text and ensures consistent spacing between words.

7. text.strip() Finally, this line removes leading and trailing whitespace from the text.

3. Document chunking:

As the text documents are very large, Consider the challenge of embedding large documents there will be Information Loss. The document chunking technique is used as a crucial step in handling large documents effectively, particularly in scenarios where embedding entire documents as single vectors is impractical due to the risk of information loss or computational constraints.



```
In [19]: def partitioning_srt_file_data_into_chunks(row, window_size=512, overlapping=100):
             chunks = []
             i = 0
             while i < len(row['subtitle_content']):
                 chunk_end = min(i + window_size, len(row['subtitle_content']))
                 chunks.append(row['subtitle_content'][i:chunk_end])
                 i += window_size - overlapping
             return chunks

         new_df['subtitle_content'] = new_df.apply(partitioning_srt_file_data_into_chunks, axis=1)

In [21]: new_df['subtitle_content'][0]

Out[21]: ['Watch any video online with OpenSUBTITLES Free Browser extension osdblinkext It couldve been just another summer But as I s
         et foot on the sand that summer suddenly felt different Like it was going to be the summer that would change my life The summ
         er of freedom The summer of endless possibilities The summer of Ooh aah Ooh oh Oh oh ooh That was the summer of you and me Yo
         ure quite the dancer Why did you stop Come on Keep dancing Whatever Im kidding Dont get mad Huh What Hey Im just going to get
         my towel What',
          'ome on Keep dancing Whatever Im kidding Dont get mad Huh What Hey Im just going to get my towel What Stop that You thought
         I was gonna kiss you No Excuse me I wanna kiss you but not just yet What do you mean not yet Only when youre my girl What do
         you mean your girl My girlfriend Miss As if You wish And dont call me miss Dont pretend to be a gentleman when youre clearly
         not So what should I call you Rude Snob Bitch And you Douche Handsome Conceited Just like you Huh Jerk Exactly your type Leav
         e me alone Ste',
          ' Bitch And you Douche Handsome Conceited Just like you Huh Jerk Exactly your type Leave me alone Steph Aha Steph Ill just c
         all you Tep Remove the S and the F By the way Im Tonio Will you still be here tomorrow Dont leave yet Im going to court you I
         chose to walk away from you But fate had a different plan Councilor Were teaching basic English literacy Well be teaching the
         children how to read and write in English Is that so Yes How long will this program run If its okay the entire summer I thoug
         ht I could e',
          'lish Is that so Yes How long will this program run If its okay the entire summer I thought I could escape you But you someh
```

`

- Embedding entire long documents as single vectors can lead to information loss due to their diverse topics, sentiments, and themes, necessitating significant computational resources.
- Addresses this by dividing large documents into smaller, manageable chunks for individual processing and embedding, improving efficiency.
- Overlapping windows are used to prevent splitting important text between chunks. Chunks are based on a token window size, ensuring each contains just below a specified number of tokens, and overlap to retain context between adjacent chunks.

4. Vectorize the given Subtitle Documents:

Bag-of-Words (BOW) and Term Frequency-Inverse Document Frequency (TF-IDF) are traditional methods for generating sparse vector representations of text data. While they are widely used and can be effective for certain tasks, they also have some disadvantages:

4 (a). Disadvantages of BOW / TF-IDF:

No Semantic Information: BOW and TF-IDF representations do not capture semantic relationships between words or documents. They treat each word independently and ignore the context in which words appear. This can lead to limitations in capturing the true meaning or intent of the text.

High Dimensionality: BOW and TF-IDF representations result in high-dimensional sparse vectors, especially for large vocabularies or datasets with many unique tokens. This high dimensionality can lead to computational inefficiencies and increased memory requirements.

Given these disadvantages, using BOW or TF-IDF for text vectorization may not be ideal for tasks that require capturing semantic information or understanding the context of the text. Instead, for tasks like building a Semantic Search Engine, it's more beneficial to use methods that encode semantic information and capture contextual relationships between words and documents.

Usage of BERT-based Sentence Transformers: BERT-based Sentence Transformers, such as those provided by the Sentence Transformers library, offer a solution to the limitations of BOW and TF-IDF. These models leverage pre-trained language representations, such as BERT, to generate dense vector embeddings that encode semantic information and capture contextual relationships between words and sentences. By using deep contextualized embeddings, Sentence Transformers can produce representations that better reflect the meaning and intent of the text.

`

4 (b). Advantages:

- Semantic Information: BERT-based Sentence Transformers capture semantic relationships between words and sentences, allowing for more nuanced understanding and interpretation of the text.
- Contextual Embeddings: Unlike BOW and TF-IDF, which treat each word independently, Sentence Transformers generate dense embeddings that take into account the context in which words appear. This helps preserve contextual information and improve the quality of representations.
- Lower Dimensionality: Dense embeddings produced by Sentence Transformers typically have lower dimensionality compared to sparse representations, reducing computational overhead and memory requirements.
- Pre-trained Models: Sentence Transformers leverage pre-trained models like BERT, which have been trained on large text corpora, capturing a wide range of linguistic patterns and semantic relationships.

```
In [39]: client = chromadb.PersistentClient(path="/path/to/save/to")

In [40]: client.heartbeat() # returns a nanosecond heartbeat. Useful for making sure the client remains connected.
Out[40]: 1713442327579887700

In [ ]: #client.reset() # Empties and completely resets the database. ⚠ This is destructive and not reversible.

In [41]: #client.delete_collection(name="subtitle_sem")

In [42]: from chromadb.utils import embedding_functions

In [44]: from sentence_transformers import SentenceTransformer,util

In [46]: sentence_transformer_ef = embedding_functions.SentenceTransformerEmbeddingFunction(model_name="all-MiniLM-L6-v2")

In [47]: collection = client.create_collection(
             name="subtitle_sem",
             metadata={"hnsw:space": "cosine"},
             embedding_function=sentence_transformer_ef
         )

In [48]: content = df_1['subtitle_content'].tolist()
         metadatas = [{'subtitle_name': name, 'subtitle_id': id} for name, id in zip(df_1['subtitle_name'], df_1['subtitle_id'])]
         ids = [str(i) for i in range(len(df_1))]

In [49]: batch_size = 5000
         num_batches = (len(content) + batch_size - 1) // batch_size

         for i in range(num_batches):
             start_idx = i * batch_size
             end_idx = min((i + 1) * batch_size, len(content))

             batch_content = content[start_idx:end_idx]
             batch_metadatas = metadatas[start_idx:end_idx]
             batch_ids = ids[start_idx:end_idx]

             collection.add(
                 documents=batch_content,
                 metadatas=batch_metadatas,
                 ids=batch_ids
             )
```

`

5. Store embeddings in a ChromaDB database:

ChromaDB is used for storing text embeddings because of its specialized features and capabilities that make it well-suited for managing and querying high-dimensional embeddings efficiently. The process of creating and populating a collection in the ChromaDB database with embeddings generated using a Sentence Transformer model. Here's a brief explanation of each step:

- Import necessary libraries including chromadb for interacting with the ChromaDB database, SentenceTransformer for generating sentence embeddings, and embedding_functions from chromadb.utils for defining the embedding function.
- It initializes a persistent client for interacting with the ChromaDB database, specifying the path where the database will be saved.
- An embedding function named ‘sentence_transformer_ef’ is defined using the SentenceTransformerEmbeddingFunction class from embedding_functions. This function utilizes a pre-trained SentenceTransformer model named "all-MiniLM-L6-v2" to generate embeddings for input sentences. "all-MiniLM-L6-v2" refers to a specific variant of the MiniLM model, which is optimized for speed, performance, and efficiency.
- A collection named "subtitle_sem" is created within the ChromaDB database. The collection is configured with metadata specifying the space as "cosine" to indicate cosine similarity for nearest neighbor search, and the defined embedding function sentence_transformer_ef is associated with the collection.
- The text content of subtitles (df_1['subtitle_content']) and corresponding metadata (df_1['subtitle_name'] and df_1['subtitle_id']) are retrieved from DataFrame df_1. The content and metadata are then split into batches for efficient insertion into the collection.
- The data is inserted into the collection in batches to avoid memory constraints and optimize performance. Each batch consists of a subset of the subtitle content, metadata, and unique IDs.The insertion process is repeated for each batch until all subtitle content, metadata, and IDs are processed.

6. Retrieving Documents

In [52]:
```
query_text = 'what happened last night when we were fully drunk'
```

In [53]:
```
result = collection.query(
    query_texts = query_text,
    include=["metadatas", 'distances'],
    n_results=10
)
```

In [54]:
```
ids = result['ids'][0]
distances = result['distances'][0]
metadatas = result['metadatas'][0]
zipped_data = zip(ids, distances, metadatas)
sorted_data = sorted(zipped_data, key=lambda x: x[1], reverse=True)
for _, distance, metadata in sorted_data:
    subtitle_name = metadata['subtitle_name']
    print(f"Subtitle Name: {subtitle_name.upper()}")
```

```
Subtitle Name: THE NOVELIST S01 E03 NIGHT OF DESIRE AND IMPULSE (2018) ENG 1CD
Subtitle Name: GHOSTS S02 E08 THE LIQUOR LICENSE (2022) ENG 1CD
Subtitle Name: NCIS S02 E15 CAUGHT ON TAPE (2005) ENG 1CD
Subtitle Name: CSI CRIME SCENE INVESTIGATION S06 E03 BITE ME (2005) ENG 1CD
Subtitle Name: THE REAL BLING RING HOLLYWOOD HEIST S01 E02 FAME FORTUNE AND FELONY (2022) ENG 1CD
Subtitle Name: 13 REASONS WHY S01 E09 TAPE 5 SIDE A (2017) ENG 1CD
Subtitle Name: CONVERSATIONS WITH A KILLER THE JEFFREY DAHMER TAPES S01 E01 SYMPATHY FOR THE DEVIL (2022) ENG 1CD
Subtitle Name: ENDEAVOUR S03 E03 PREY (2016) ENG 1CD
Subtitle Name: MISTER8 S01 E05 KOMEDIAPERJANTAI (2021) ENG 1CD
Subtitle Name: I WAS LORENA BOBBITT (2020) ENG 1CD
```

The process of retrieving documents based on a user's search query involves several key steps process of performing a semantic search using embeddings stored in a ChromaDB collection.

The user's search query is taken as input. In this case, the query text is "what happened last night when we were fully drunk".

While performing a query in the ChromaDB collection, query text is provided directly to ChromaDB, already a defined embedding function sentence_transformer_ef is associated with the collection , ChromaDB internally uses the embedding function to generate embeddings for the query text.

ChromaDB then calculates the similarity scores between the query embedding and the embeddings of documents stored in the collection using a cosine distance as similarity measure.

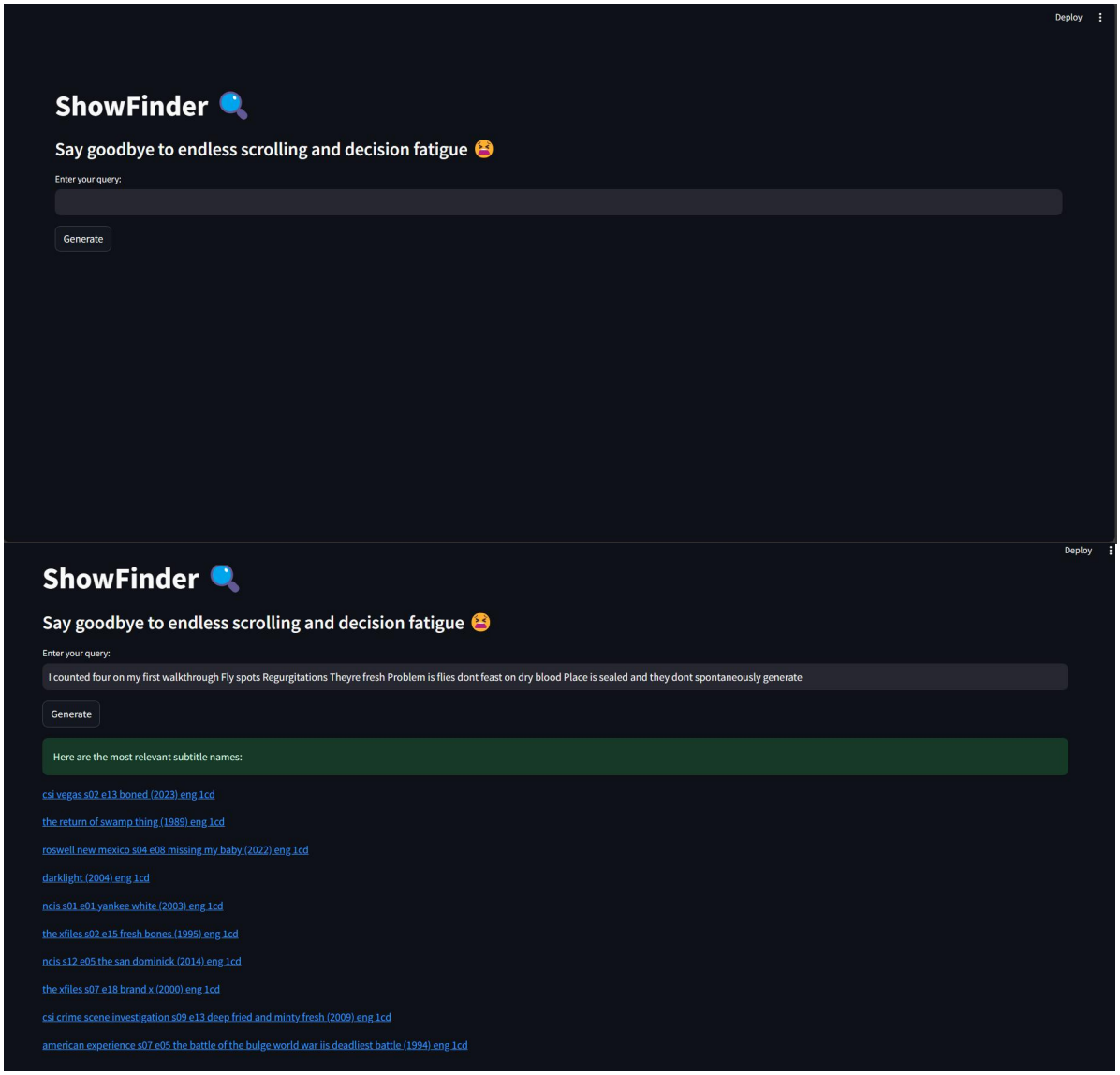
The cosine similarity scores are used to rank the candidate documents based on their relevance to the user's search query. The code retrieves the top 10 most relevant documents from the collection, along with their corresponding distances (cosine similarity scores) and metadata.

The sorted list of candidate documents, along with their metadata (e.g., subtitle name), is printed to the console. The documents are sorted based on their cosine similarity scores in descending order, with the most relevant documents appearing first.

This approach allows you to leverage the pre-trained embedding model directly within ChromaDB, simplifies the process of performing similarity searches without the need to generate embeddings externally.

`

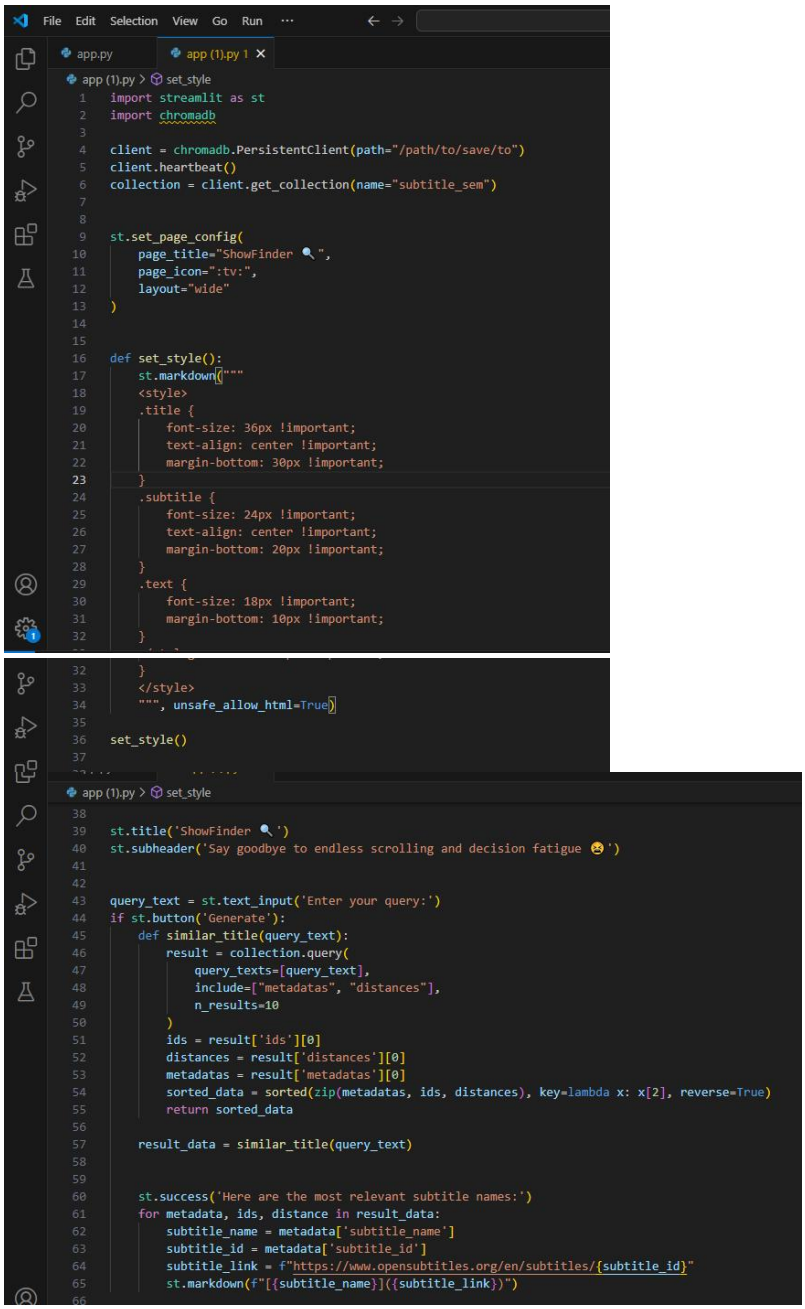
7. Retrieving Documents using user input Application:



This Streamlit app, titled "ShowFinder," is designed to help users find relevant subtitles based on a search query.

`

Backend application code:



```python
import streamlit as st
import chromadb

client = chromadb.PersistentClient(path="/path/to/save/to")
client.heartbeat()
collection = client.get_collection(name="subtitle_sem")


st.set_page_config(
    page_title="ShowFinder 🔍",
    page_icon=":tv:",
    layout="wide"
)


def set_style():
    st.markdown("""
    <style>
    .title {
        font-size: 36px !important;
        text-align: center !important;
        margin-bottom: 30px !important;
    }
    .subtitle {
        font-size: 24px !important;
        text-align: center !important;
        margin-bottom: 20px !important;
    }
    .text {
        font-size: 18px !important;
        margin-bottom: 10px !important;
    }
    }
    </style>
    """, unsafe_allow_html=True)

set_style()

st.title('ShowFinder 🔍')
st.subheader('Say goodbye to endless scrolling and decision fatigue 😫')


query_text = st.text_input('Enter your query:')
if st.button('Generate'):
    def similar_title(query_text):
        result = collection.query(
            query_texts=[query_text],
            include=["metadatas", "distances"],
            n_results=10
        )
        ids = result['ids'][0]
        distances = result['distances'][0]
        metadatas = result['metadatas'][0]
        sorted_data = sorted(zip(metadatas, ids, distances), key=lambda x: x[2], reverse=True)
        return sorted_data

    result_data = similar_title(query_text)


    st.success('Here are the most relevant subtitle names:')
    for metadata, ids, distance in result_data:
        subtitle_name = metadata['subtitle_name']
        subtitle_id = metadata['subtitle_id']
        subtitle_link = f"https://www.opensubtitles.org/en/subtitles/{subtitle_id}"
        st.markdown(f"[{subtitle_name}]({subtitle_link})")
```

- The app imports necessary libraries including streamlit for creating web applications and chromadb for interacting with the ChromaDB database.
- A persistent client for interacting with ChromaDB is initialized, specifying the path where the database will be saved.
- Streamlit page configuration settings are set, including page title, icon, and layout.

`

- Custom CSS styling is applied to the app using Streamlit's set_page_config and markdown functions to adjust font sizes and alignment for titles, subtitles, and text.
- The UI elements are defined using Streamlit's st functions. Users can input their search query using a text input field, and a button allows them to trigger the search.
- When the user clicks the "Generate" button, the search query is passed to the similar_title function, which queries the ChromaDB collection for similar subtitles based on the query text. The results include metadata such as subtitle name and ID, along with the cosine similarity distances.
- The most relevant subtitle names are displayed using Streamlit's st.success and st.markdown functions. Each subtitle name is presented as a clickable link that redirects to the corresponding subtitle page on OpenSubtitles.org.
- Overall, this Streamlit app provides an intuitive interface for users to search for subtitles based on their queries, leveraging embeddings stored in the ChromaDB collection for efficient and accurate search results.

8. Conclusion:

In conclusion, the documentation encompasses essential aspects of working with subtitle data, NLP models, and search engine development. It outlines techniques for retrieving and preprocessing subtitle data from a SQLite database, as well as extracting content from zip archives containing subtitle files. The choice of the "all-MiniLM-L6-v2" model for embedding generation is based on considerations of accuracy, speed, performance, model size, and compatibility. Additionally, the integration of NLP techniques into a search engine framework facilitates efficient information retrieval based on user queries. Overall, the documentation provides a comprehensive guide for developing NLP-based search systems tailored to subtitle data, enabling effective extraction of insights from text data in the context of video content.