

VIETNAM NATIONAL UNIVERSITY – HCM INTERNATIONAL UNIVERSITY

SCHOOL OF COMPUTER SCIENCE AND ENGINEERING



MACHINE LEARNING PLATFORMS

PROJECT REPORT

Semester 1, 2024–2025

Instructor: Ho Long Van

Topic: Movies Recommender System

Github: https://github.com/Khim3/Film_Recommender_System_using_VectorDB

TEAM MEMBERS

No.	Name	Student ID	Contribution
1	Nguyễn Huy Bảo	ITDSIU21076	33%
2	Nguyễn Nhật Khiêm	ITDSIU21091	33%
3	Trịnh Bình Nguyên	ITDSIU21103	33%

TABLE OF CONTENTS

TABLE OF CONTENTS	3
TABLE OF FIGURES	4
PROJECT TIMELINE	5
CHAPTER 1: INTRODUCTION	7
A. Overview	7
B. Objectives.....	7
C. About the dataset	7
D. Specifications	8
E. Project workflow	8
CHAPTER 2: DATA SELECTION AND PREPROCESSING	9
A. Data Selection	9
B. Preprocessing Steps	9
CHAPTER 3: VECTOR DATABASE SELECTION AND CREATION	11
A. Selection Criteria	11
B. Database Implementation	12
CHAPTER 4: UTILITIES IMPLEMENTATION	15
A. Embedding Generation	15
B. Smooth Description Generation	15
C. Search and Recommendation	16
D. Data Display	16
CHAPTER 5: USER INTERFACE DESIGN AND AI INTEGRATION	17
A. User Interface	17
B. AI Integration.....	19
CHAPTER 6: CONCLUSION	19
A. Accomplishment.....	19
B. Limitations.....	20
D. Reference	21

TABLE OF FIGURES

Figure 1 - Project 's Pipeline	8
Figure 2 - Overview of Dataset.....	9
Figure 3 - Comma-separated string conversion	10
Figure 4 - Drop unnecessary features	10
Figure 5 - Function to create embeddings.....	11
Figure 6 - Search index establishment	13
Figure 7 - Vector search pipeline.....	14
Figure 8 - Leverage LLM power to generate natural recommendations	16
Figure 9 - Display the recommendations to users.....	17
Figure 10 - Bried overview of the UI.....	18
Figure 11 - Chat area for user to input film plot	18
Figure 12 - LLM answer generation	19

PROJECT TIMELINE

Stage	Task	Member	Week No.
Planning	Individual's topic research	All	1 - 2
	Set up project timeline, design workflow	Khiem	
	Topic confirmation	All	
	Identify the objectives and purposes of the analysis	All	
	Agree on means of communication, workflow and tools	All	
	Decide scopes, aims, goals and requirements of the project	All	
	Create GitHub repository	Khiem	
Data Selection & Data Preprocessing	Data searching and confirmation	Khiem	3
	Detect and deal with missing, duplicated values	Nguyen	
	Detect and deal with outliers	Nguyen	
	Compare pre-process and post-process	Bao	
Set up vector database	Select the database platform and set up connection	Nguyen	3 - 7
	Create a vector database schema with embeddings		
	Establish index search on vector database		
	Set up the vector search function for choosing essential info for retrieval		
Implement utilities	Set up embedding creation for both data and user input	Bao	3 - 7
	Set up function to leverage embedding for natural answer		

	upon retrieval and practical use		
	Display search result with add more function		
Design User Interface	Set up Streamlit design for UI	Khiem	9 - 12
	Integrate embedding and language model for text generation into UI		
	Maintain and check the functionality of each component of the system		
Report & Presentation	Report form setting up	Khiem	13 - 15
	Data Overview and Preprocessing	Nguyen	
	Database selection and implementation	Nguyen	
	Utils functions details	Bao	
	UI implementation and AI integration	Khiem	
	Accomplishments		
	Restrictions		
	Future works		

CHAPTER 1: INTRODUCTION

A. Overview

The Film Recommender System is designed to suggest movies by comparing their stories and identifying those that are most similar. It stores information about movie plots in a special database that helps find connections between them. By focusing on the content of the stories rather than just basic details like genre or year, the system can provide recommendations that feel more relevant and meaningful. With its simple and easy-to-use interface, the system offers users a fun and engaging way to discover movies tailored to their interests.

B. Objectives

The goal of this project is to develop a user-friendly film recommender system that provides personalized movie suggestions by analyzing plot content and themes. It aims to enhance the movie discovery experience using advanced technologies, ensuring accurate and engaging recommendations tailored to users' preferences.

C. About the dataset

The chosen dataset contains detailed information about movies in the Western, Action, or Fantasy genres. Each record includes attributes such as title, release year, cast, plot summary, genres, runtime, ratings, directors, writers, production countries, languages, and awards. Additionally, the dataset features a `plot_embedding` field with embeddings generated using OpenAI's text-embedding-ada-002 model, enabling advanced search and analysis based on plot similarities.

For our project, we replaced these pre-generated embeddings with custom embeddings created using the Ollama Nomic text embedding model to better align with the system's specific requirements, limitations and improve recommendation accuracy.

The dataset contains detailed information about movies, including:

- **Plot:** A brief description of the movie storyline.
- **Runtime:** Duration of the movie.
- **Genres:** Categories such as Drama, Action, etc.
- **Directors, Writers, Cast:** Key contributors to the movie.
- **Countries, Languages:** Locations and languages associated with the movie.
- **Metacritic:** Critical rating of the movie (dropped during preprocessing).

D. Specifications

Tools	Purpose
Python 3	For interpreting and running Python script
Streamlit	For creating attractive and straightforward user interface
MongoDB Atlas Cloud	For embedding vector creation and storage
Ollama	For embedding model and LLM integration to the system

E. Project workflow

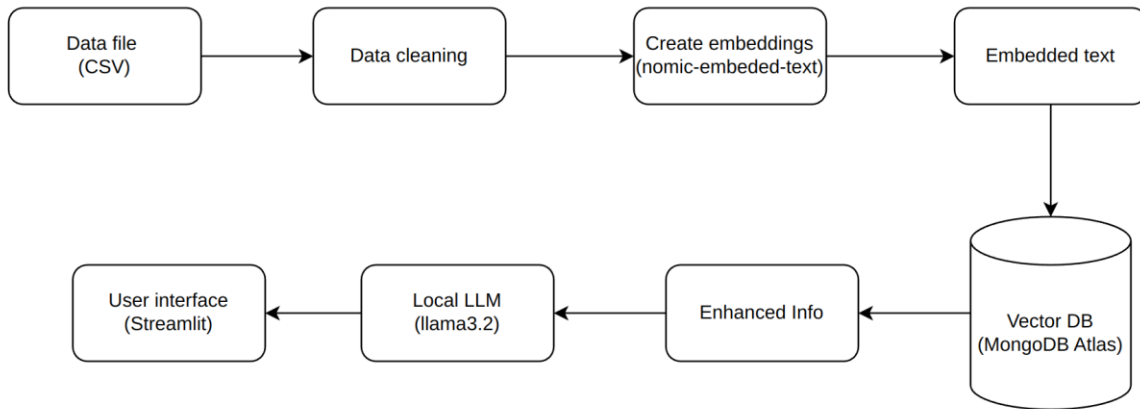


Figure 1 - Project 's Pipeline

1. **Data File (CSV):** The system begins with a dataset containing raw movie information, including plot details and metadata, stored in a CSV file.
2. **Data Cleaning:** The raw data undergoes a cleaning process to ensure consistency, remove noise, and prepare it for embedding generation.
3. **Create Embeddings:** Using the **Nomic-embedded-text** model, movie plots are converted into vector embeddings that capture their semantic meaning.
4. **Embedded Text:** These embeddings are then stored in a **Vector Database (MongoDB Atlas)**, enabling efficient similarity searches and data retrieval.
5. **Enhanced Information:** The stored embeddings and metadata are accessed to generate improved recommendations by leveraging additional context.

6. **Local LLM (Llama 3.2):** A local language model processes the retrieved data to refine and polish the output, ensuring the recommendations are user-friendly and contextually appropriate.
7. **User Interface (Streamlit):** Finally, the recommendations are displayed in a streamlined and interactive user interface built with Streamlit, allowing users to easily explore suggested movies.

CHAPTER 2: DATA SELECTION AND PREPROCESSING

A. Data Selection

The dataset was selected for its comprehensive coverage of movie metadata, allowing for diverse and meaningful recommendations. Key columns included as illustrated below:

- Plot descriptions for semantic embedding.
- Genre and runtime for metadata-based filtering.
- Contributors and filming locations for added context.

	plot	runtime	genres	fullplot	directors	writers	countries	poster	languages	cast	title
0	Young Pauline is left a lot of money when her ...	199.0	['Action']	Young Pauline is left a lot of money when her ...	['Louis J. Gasnier', 'Donald MacKenzie']	['Charles W. Goddard (screenplay)', 'Basil Dic...	['USA']	amazon.com/images/M/MV5BMzgxD...	['English']	['Pearl White', 'Crane Wilbur', 'Paul Panzer' ,...	The Perils of Pauline
1	A penniless young man tries to save an heiress...	22.0	['Comedy', 'Short', 'Action']	As a penniless man worries about how he will m...	['Alfred J. Goulding', 'Hal Roach']	['H.M. Walker (titles)']	['USA']	amazon.com/images/M/MV5BNzE1OW...	['English']	['Harold Lloyd', 'Mildred Davis', '"Snub' Poll...	From Hand to Mouth
2	Michael "Beau" Geste leaves England in disgrac...	101.0	['Action', 'Adventure', 'Drama']	Michael "Beau" Geste leaves England in disgrac...	['Herbert Brenon']	['Herbert Brenon (adaptation)', 'John Russell ...	['USA']	NaN	['English']	['Ronald Colman', 'Neil Hamilton', 'Ralph Forb...	Beau Geste

Figure 2 - Overview of Dataset

B. Preprocessing Steps

1. Column Cleaning:

- Removed unnecessary columns like metacritic.
- Converted string representations of lists (e.g., directors, cast) into comma-separated strings.

```

1 import ast
2 column_list = ['directors', 'writers', 'cast', 'countries', 'languages']
3
4 # Drop the 'metacritics' column
5 df.drop(columns='metacritic', inplace=True)
6
7 # Convert string representations of lists to comma-separated strings
8 for col in column_list:
9     df[col] = df[col].apply(lambda x: ', '.join(ast.literal_eval(x)) if isinstance(x, str) else x)

```

Figure 3 - Comma-separated string conversion

2. Handling Missing Data:

- Imputed missing values where necessary.
- Dropped rows with insufficient data in critical fields such as plot, poster, rated, metacritics.

```

1 df.drop(['rated', 'type', 'num_mflix_comments'], axis=1, inplace=True)
2 df.head()

```

Figure 4 - Drop unnecessary features

3. Embedding Generation:

- Transformed plot into vector embeddings using the **nomic-embedded-text** model through **Ollama API**.
- Added a new column plot_embedding to store the generated vectors (as described below)

```

1  def create_embedding(df, chosen_column):
2      embedding_column = chosen_column + '_embedding'
3      # Initialize an empty list to store embeddings
4      embeddings = []
5
6      for index, row in df.iterrows():
7          text = row[chosen_column]
8          if isinstance(text, str) and text.strip():
9              try:
10                 # Generate the embedding for the text
11                 response = ollama.embeddings(model="nomic-embed-text", prompt=text)
12                 embedding = response.get("embedding", [])
13             except Exception as e:
14                 embedding = []
15                 print(f"Error embedding row {index}: {e}")
16             else:
17                 embedding = []
18
19             embeddings.append(embedding)
20             df[embedding_column] = embeddings
21         return df

```

Figure 5 - Function to create embeddings

CHAPTER 3: VECTOR DATABASE SELECTION AND CREATION

A. Selection Criteria

The selection of the Vector Database (VectorDB) for this project was guided by the following essential criteria:

1. Scalability:

- The database needed to efficiently handle large datasets, such as those containing thousands of movie embeddings. This ensures that the system remains responsive and effective as the dataset grows in size over time.
- Scalability is crucial for real-world applications, where increasing user interactions or additional data sources can lead to exponential growth in stored information.

2. Similarity Search:

- High-speed nearest neighbor search was a key requirement for identifying embeddings that are most similar to a given query. This is particularly

important for use cases like movie recommendations, where quick and accurate retrieval of relevant data directly impacts the user experience.

- The ability to compare vector embeddings with minimal latency was a non-negotiable feature, given the need for real-time query resolution.

3. **Integration:**

- The chosen VectorDB had to seamlessly integrate with machine learning workflows, allowing for direct embedding storage and retrieval. This compatibility eliminates the need for additional data conversion layers and simplifies the overall architecture.
- Support for Python-based APIs and libraries, such as pymongo, was prioritized to align with the existing tech stack.

4. **Accessibility:**

- The database should be easy to configure and accessible through standard Uniform Resource Identifiers (URIs). This simplifies the setup process and allows for rapid deployment across various environments, including local, cloud, and hybrid setups.
- MongoDB's flexibility in providing managed services and cross-platform support enhanced its appeal as the database of choice.

B. Database Implementation

1. Database Choice

- MongoDB was selected for its ability to handle unstructured and semi-structured data efficiently. It offers features such as flexible schema design, high availability, and horizontal scaling, making it an ideal choice for embedding-based applications.
- MongoDB's built-in vector search capabilities further solidified its position as a robust solution for this project. These capabilities allow for the direct handling of high-dimensional vector data without requiring additional plugins or tools.

2. Index Creation

- To enable efficient similarity search, a Hierarchical Navigable Small World (HNSW) index was created on the plot_embedding field. This index supports the cosine similarity metric, which is particularly effective for comparing embeddings derived from machine learning models.
- The HNSW algorithm ensures low-latency retrieval by organizing vectors into a navigable graph structure, allowing for quick nearest neighbor searches.
- The index was configured to accommodate embeddings with 768 dimensions, reflecting the output size of the machine learning model used for feature extraction.

- This ensures that the database is optimized for high-performance queries, even as the dataset grows.

```

1 def create_search_index(client, db_name, collection_name, field_name, num_dimensions=768):
2     try:
3         database = client[db_name]
4         collection = database[collection_name]
5         search_index_model = SearchIndexModel(
6             definition={
7                 "fields": [
8                     {
9                         "type": "vector",
10                        "numDimensions": num_dimensions,
11                        "path": field_name,
12                        "similarity": "cosine"
13                    }
14                ]
15            },
16            name="vector_search_index",
17            type="vectorSearch"
18        )
19        result = collection.create_search_index(model=search_index_model)
20        # st.sidebar.success("Search index created successfully!")
21    except Exception:
22        st.sidebar.info("Search index already exists!")

```

Figure 6 - Search index establishment

3. Data Insertion

- The insertion process involved preprocessing movie data, including title, genres, director, and plot embeddings generated by a pre-trained language model. These embeddings, stored as 768-dimensional vectors, represent the semantic meaning of the movie descriptions.
- Each data entry was structured as a document in **MongoDB**, with fields such as title, director, genres, plot_embedding, and other relevant metadata.

4. Vector Search Implementation

- MongoDB's **\$vectorSearch** stage was used for querying the vector data. The query involved providing a 768-dimensional vector as input and retrieving the top matching documents based on cosine similarity.
- The pipeline for vector search was designed to include stages for similarity scoring, data projection, and optional field removal for better performance. (as shown below)

```

1 def vector_search(user_query, collection: Collection, field):
2     query_embedding = create_embedding_query(user_query)
3     if not query_embedding:
4         st.error("Invalid query or empty embedding")
5         return 'Invalid query'
6
7     vector_search_stage = {
8         "$vectorSearch": {
9             "index": "vector_search_index",
10            "queryVector": query_embedding,
11            "path": field,
12            "numCandidates": 25,
13            'limit': 20
14        }
15    }
16    unset_stage = {
17        '$unset': field
18    }
19
20    project_stage = {
21        '$project': {
22            '_id': 0,
23            'fullplot': 1,
24            'title': 1,
25            'director': 1,
26            'countries': 1,
27            'genres': 1,
28            'poster': 1,
29            'score': {
30                '$meta': 'vectorSearchScore'
31            }
32        }
33    }
34
35    pipeline = [vector_search_stage, unset_stage, project_stage]
36
37    try:
38        results = collection.aggregate(pipeline)
39        return list(results)
40    except Exception:
41        st.error("Error performing vector search!")
42        return []

```

Figure 7 - Vector search pipeline

5. Advantages of HNSW and Cosine Similarity

- The HNSW index offers logarithmic complexity for search operations, ensuring fast response times.

- Cosine similarity is particularly well-suited for text embeddings as it focuses on the angle between vectors rather than their magnitude, ensuring robust comparisons even when vector scales vary.

6. Error Handling and Robustness

- Comprehensive error handling mechanisms were implemented to manage potential issues during data insertion, index creation, and query execution. This ensures system stability and a better user experience.

CHAPTER 4: UTILITIES IMPLEMENTATION

A. Embedding Generation

1. **Model Used:** The **nomic-embed-text** model from Ollama was employed for generating 768-dimensional embeddings.
2. **Embedding Process:**
 - Each movie plot is passed through the embedding model to generate high-dimensional vector representations.
 - The embeddings are stored in MongoDB's **plot_embedding** field for efficient similarity matching.
3. **Query Embedding:**
 - User input is also embedded into a vector, which is then compared against stored embeddings to retrieve similar movies.

B. Smooth Description Generation

1. **Model Used:** The **llama 3.2** local language model was integrated for description generation.
2. **Functionality:**
 - Structured metadata such as title, genres, and plot are processed by the language model.
 - The model transforms the information into a conversational and humanized format.
3. **Usage:**
 - This utility is invoked after retrieving search results to refine and polish the movie details presented to users.



Figure 8 - Leverage LLM power to generate natural recommendations

C. Search and Recommendation

1. Vector Search:

- MongoDB's **\$vectorSearch** stage is utilized to perform similarity searches based on cosine similarity.
- User queries are matched against the stored embeddings, retrieving the top-ranked movies.

2. Pipeline:

- Query embedding creation.
- Vector search execution with similarity scoring.
- Projection of relevant fields like title, plot, genres, and scores.

3. Dynamic Retrieval:

- The system supports iterative queries, allowing users to retrieve additional results incrementally.

D. Data Display

1. Interactive Features:

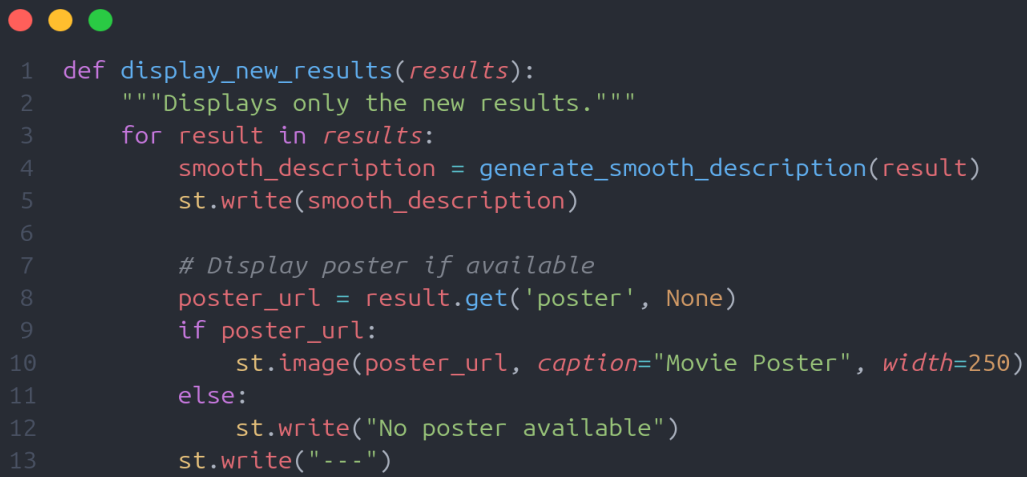
- Display of movie titles, smooth descriptions, and metadata.
- Inclusion of movie posters (if available) for visual appeal.

2. Dynamic Loading:

- Users can load more results with a "**Get More Results**" button.
- Results are presented in batches, ensuring smooth scrolling and usability.

3. Error Handling:

- Messages are displayed for invalid queries or missing results.



```
1 def display_new_results(results):
2     """Displays only the new results."""
3     for result in results:
4         smooth_description = generate_smooth_description(result)
5         st.write(smooth_description)
6
7         # Display poster if available
8         poster_url = result.get('poster', None)
9         if poster_url:
10             st.image(poster_url, caption="Movie Poster", width=250)
11         else:
12             st.write("No poster available")
13         st.write("---")
```

Figure 9 - Display the recommendations to users

CHAPTER 5: USER INTERFACE DESIGN AND AI INTEGRATION

A. User Interface

1. Developed using **Streamlit** for an intuitive and interactive experience.

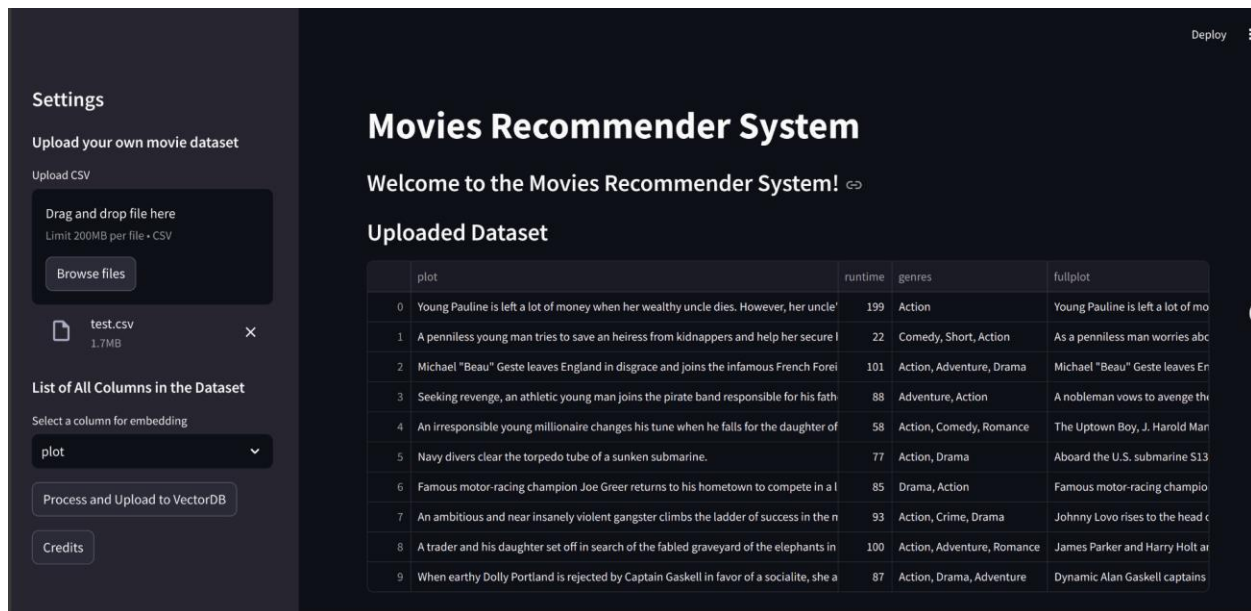


Figure 10 - Bried overview of the UI

- First once the data is loaded to the system, it would show the first 10 lines from the dataset which give users a basic intuition about data.
- Interact to initialize the data preparation and vector database
 - After the data is fully loaded, users need to press Process and Upload to VectorDB in order that the system could be set up and ready for use.
 - Recommended movies with posters and descriptions
 - Next, users could enter the plot of the film that they might want to see to the user chat box, hit Enter and press the Search button for system to generate the desired films.

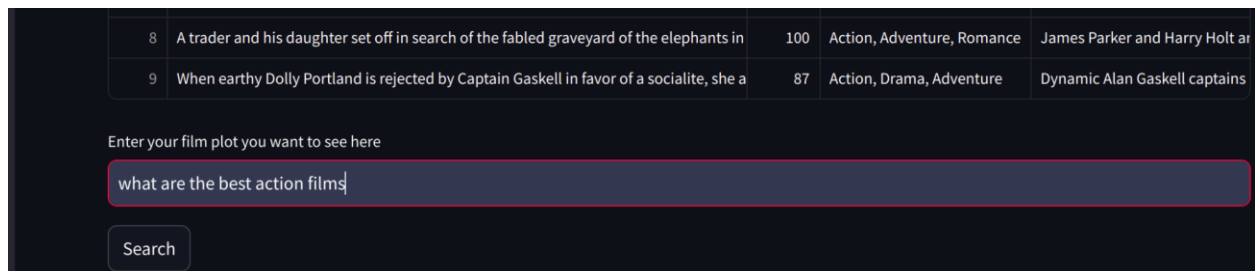


Figure 11 - Chat area for user to input film plot

B. AI Integration

Once users have done entering the input for plot, the system would leverage AI models for generating recommendations. One model is to create embedding from the user input and get it compared to the others which are stored in vector DB. Next step is handled by another LLM (llama 3.2) to be based on the embeddings of both user and existing ones to make a natural human language output to display.

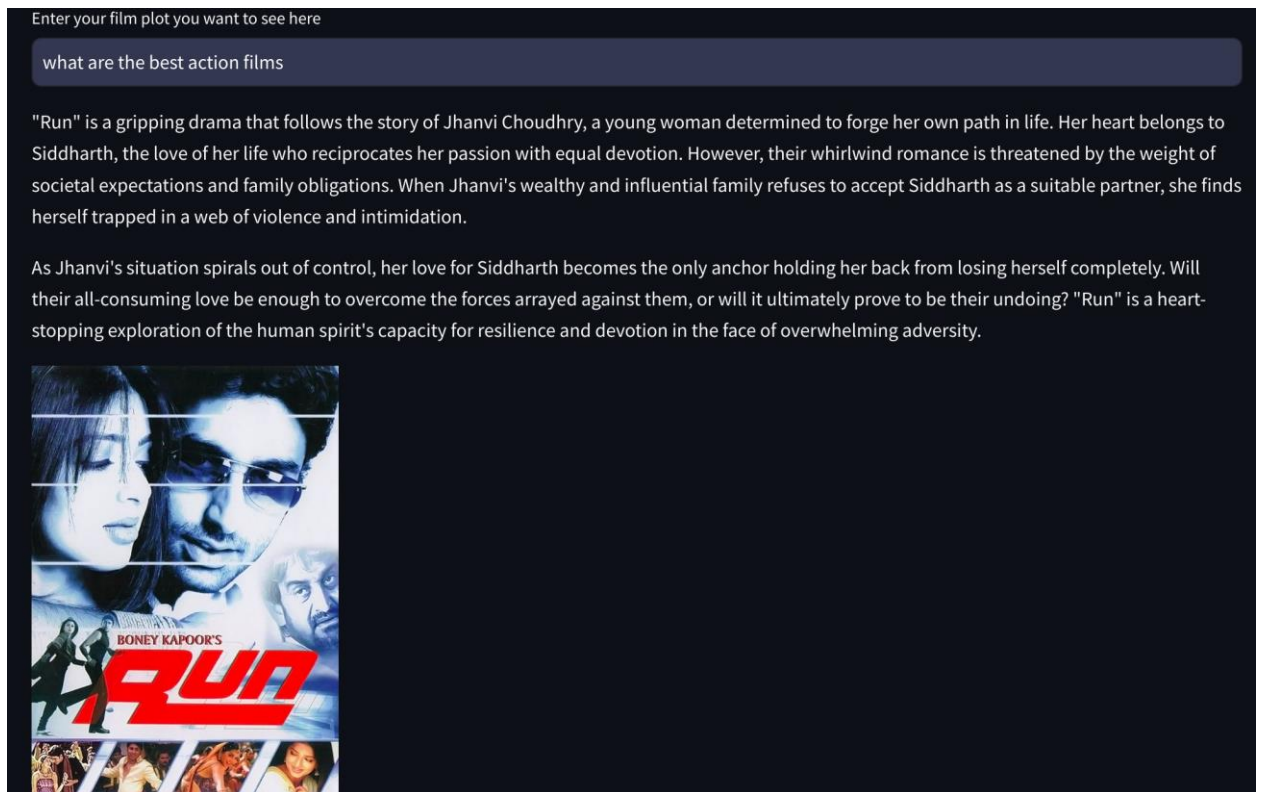


Figure 12 - LLM answer generation

CHAPTER 6: CONCLUSION

A. Accomplishment

1. Effective Embedding Implementation

The system successfully replaced pre-existing embeddings with custom embeddings generated using the Nomic-embedded-text model. These embeddings were specifically tailored to capture the subtle nuances of movie plots, ensuring more accurate and contextually relevant recommendations. A key accomplishment of this implementation is its ability to run smoothly on a local machine, requiring minimal computational resources while maintaining high efficiency. This makes the system accessible for smaller-scale deployments without the need for expensive infrastructure.

2. End-to-End System Design

A complete workflow was designed and implemented, covering every stage from raw data preprocessing to final user interaction. This includes steps such as cleaning and structuring the input data, generating embeddings, storing them in a Vector Database, and retrieving refined recommendations. The design ensures a seamless flow of data and an efficient recommendation process, with a focus on simplicity, robustness, and scalability. The system's modular architecture allows for easy updates and enhancements, making it both adaptable and future-ready.

3. **Refined Recommendation Generation**

The Llama 3.2 local language model was leveraged to refine and contextualize movie recommendations, providing users with clear and engaging suggestions. This model enhances the system's ability to deliver personalized outputs by polishing and formatting the retrieved information into user-friendly recommendations. Furthermore, it was optimized to run efficiently on a local machine, requiring minimal computational resources. This lightweight design makes the system practical for personal or small-scale use without compromising the quality of the recommendations.

4. **Real-World Application**

The system demonstrates practical value by successfully integrating advanced machine learning techniques with modern database technologies. By focusing on semantic plot similarities rather than superficial metadata, the recommender provides meaningful and personalized suggestions. Its lightweight and resource-efficient design ensures that it can be deployed on local machines, making it suitable for a variety of real-world scenarios. This project showcases how cutting-edge technologies can be utilized to solve practical problems and enhance user experiences, bridging the gap between research and application.

B. Limitations

1. **Dependence on Dataset Quality**

The effectiveness of the system heavily relies on the quality and completeness of the dataset. Incomplete or inaccurate data can significantly affect the accuracy of recommendations and limit the system's ability to provide meaningful results.

2. **Limited Multi-Lingual Support**

Currently, the system has limited support for multi-lingual movie descriptions. This restricts its usability for non-English datasets or international users who prefer recommendations in their native language.

3. **Restricted Search Indexing**

The system is limited by the free-tier indexing constraints of MongoDB Atlas, which restricts the number of search indexes that can be created for different datasets. This can pose challenges when scaling or adapting the system to accommodate larger or more diverse datasets.

4. **Plot-Based Recommendations Only**

The system is designed to generate recommendations solely based on the similarity of movie plots. It does not consider other factors like awards, ratings, or suggestions based on directors or actors. This focus on plot content, while useful, limits the range and versatility of the recommendations.

C. Future works

1. **Enhanced Embedding and Search Indexing**

To support a wider range of queries and use cases, future improvements will focus on increasing the availability of embeddings and expanding the number of search indexes. This could involve upgrading to a higher tier of MongoDB Atlas or exploring alternative vector databases with fewer limitations.

2. **Expanding the Dataset**

The dataset will be expanded to include more diverse and multi-lingual movies, enabling the system to cater to a broader audience and deliver recommendations in multiple languages. This will also enhance the system's global applicability.

3. **Improved Embedding Models**

Future iterations will involve upgrading the embedding models to better capture deeper semantic relationships in movie plots. This could include using more advanced models or fine-tuning existing ones to improve accuracy and contextual understanding.

4. **Incorporating Additional Recommendation Algorithms**

To provide a richer recommendation experience, the system will integrate additional algorithms, such as user profile-based filtering or collaborative filtering. Comparative evaluations will be conducted to optimize the performance and match recommendations more closely to user preferences.

D. Reference

1. MongoDB. (n.d.). Create an Atlas Search Index. [Website]. Retrieved from <https://www.mongodb.com/docs/atlas/atlas-search/create-index/>
2. Streamlit Docs. (n.d.). Retrieved from <https://docs.streamlit.io/>
3. ProtonX. (2024, May 23). Chi tiết về RAG - Retrieval Augmented Generation. [Video]. YouTube. <https://www.youtube.com/watch?v=6523788337e7f90012890145>
4. llama3.2. (n.d.). Retrieved from <https://ollama.com/library/llama3.2>
5. (N.d.). Retrieved from https://huggingface.co/datasets/MongoDB/embedded_movies
6. nomic-embed-text. (n.d.). Retrieved from <https://ollama.com/library/nomic-embed-text>