CONTEXT-AWARE MOVIE RECOMMENDATIONS WITH LARGE LANGUAGE MODELS

C.B. Attidiya

208523X

Degree of Master of Science in Artificial Intelligence

Department of Computational Mathematics

Faculty of Information Technology

University of Moratuwa

Sri Lanka

April 2024

CONTEXT-AWARE MOVIE RECOMMENDATIONS WITH LARGE LANGUAGE MODELS

C.B. Attidiya

208523X

Thesis/Dissertation submitted in partial fulfillment of the requirements for the degree

Degree of Master of Science in Artificial Intelligence

Department of Computational Mathematics

Faculty of Information Technology

University of Moratuwa

Sri Lanka

April 2024

DECLARATION

I declare that this is my own work, and this thesis/dissertation does not incorporate without acknowledgement any material previously submitted for a degree or diploma in any other University or Institute of higher learning and to the best of my knowledge and belief it does not contain any material previously published or written by another person except where the acknowledgement is made in the text. I retain the right to use this content in whole or part in future works (such as articles or books).

Signature: Date: 2024/07/12

The above candidate has carried out research for the ~~PhD~~/ ~~MPhil~~/ Masters thesis/dissertation under my supervision. I confirm that the declaration made above by the student is true and correct.

Name of Supervisor: Dr. A.T.P. Silva

Signature of the Supervisor: Date:

DEDICATION

It is my pleasure to dedicate this thesis to the cherished individuals in my life, including my mentor Dr. Thushari Silva and my beloved parents, teachers, lecturers, and supportive colleagues who have offered their unwavering guidance and assistance throughout my personal and academic journey.

ACKNOWLEDGEMENT

I am immensely grateful to Dr. Thushari Silva, my supervisor, for his invaluable guidance, constructive comments, and vital support throughout the entire research process, which has contributed significantly to the success of this project.

Furthermore, I extend my gratitude to the Department of Computational Mathematics lecturers, my parents, and supportive colleagues for their invaluable assistance, encouragement, and guidance throughout the research process.

ABSTRACT

The rapid growth of online streaming platforms has resulted in a vast information overload of movie content, presenting a challenge for users in their day to day lives. Effective recommendation systems are vital for improving user experience by providing personalizing content to individual users. This thesis introduces an advanced approach leveraging cutting-edge artificial intelligence technologies, employing ChatGPT along with text-embedding-3-large embedding model, integrated with the Weaviate vector database, to create dynamic recommendation model for movie recommendation. We have generated embedding vectors for 26,942 movies with additionally retrieved metadata, to build this personalized recommendation system. Two distinct recommendation models were developed and were used to compare their recommendation capabilities. The first model creates a centroid reference vector for each user based on their movie ratings of three stars or higher. This reference vector is then used to search for similar movies within the vector database using L2-squared distance, aiming to match user movie preferences. The second model refines this approach by adjusting the centroid vectors according to the weight of user ratings, allowing for a more complete reflection of user tastes. Comparative analysis on recall, precision, F1 score, and hit rate (HR) for the top 5 and top 10 recommendations shows Model 1 achieving a recall of 1.8% and 2.81%, precision of 17.01% and 14.03%, and F1 scores of 3.15% and 4.45% respectively. These metrics demonstrate its effectiveness, with HR in the top 5 and top 10 recommendations reaching 48.97% and 63.66%, respectively. This indicates that Model 1 not only identifies relevant movies with relatively better accuracy but also does so more consistently than Model 2 and is computationally efficient. The results of this study highlight the potential of using advanced AI models to significantly enhance the precision and user satisfaction of movie recommendation systems. This approach allows for a deeper understanding of user preferences and the ability to predict user needs, ultimately leading to a more engaging and satisfying viewing experience.

Keywords: movie recommendation systems, large language model, vector databases, embedding model

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ABBREVIATIONS

Abbreviation	Description
BERT	Bidirectional Encoder Representations from Transformers
CF	Collaborative Filtering
CoT	Chain of Thought
GPT	Generative Pre-trained Transformer
HNSW	Hierarchical Navigable Small World
LLaMA	Large Language Model Meta AI
LLM	Large Language Model
LSTM	Long Short-Term Memory
NLP	Natural Language Processing
RAG	Retrieval-Augmented Generation
RLHF	Reinforcement Learning from Human Feedback
RNN	Recurrent Neural Network

CHAPTER 1

INTRODUCTION

1.1 Prolegomena

The growth of the internet has resulted in an overwhelming surge of data, creating an information overload issue for people that has complicated the navigation of online services. This challenge is obvious on various internet-based platforms that offer a wide variety of entertainment options, making the search for specific content both difficult and time-consuming. Personalized recommendations have become a common feature in online services, ecommerce sites, media platforms and social networks [2]. Ecommerce and streaming platforms such as Amazon and Netflix have become popular not only due to their complex recommendation systems but ease of use, diverse content availability and recently due to the covid pandemic. Currently, the field of recommendation systems is experiencing significant growth due to its high practical relevance. The continuous development of AI technologies has further propelled this growth by enhancing recommendations.

The development of LLMs (Large Language Models) has fast tracked the advancements in the natural language processing field. The models such as BERT [3], GPT [4], LLaMA [5] and other open-source models, leverage the transformer architecture [6]. Due to its attention mechanism, these LLMs can perform a wide range of tasks with minimal task-specific training. Furthermore, the scale of these LLMs has been crucial in their success as larger models perform better compared to small and mid-sized models of the same model family. These LLMs therefore can be used in recommendation systems as they have stronger natural language interpretability. Researchers have utilized LLMs in varying degrees to improve personalized recommendation [1], utilizing methods such as fine-tuning, prompting and RLHF (Reinforcement Learning from Human Feedback). RAG (Retrieval-Augmented Generation) is another technique used for improving LLMs with current facts to improve its outputs as a LLM is trained on a large corpus of data and it doesn't have information past the input data used. Considering all these developments in LLMs it is undoubtedly wise to utilize LLMs in personalized recommendation tasks.

This thesis will outline the literature, technology, approach, design, implementation, and extensive evaluation of the proposed system that employs LLMs within a context-aware framework. We will detail the technological challenges encountered in the development process, the methodology used to overcome it and the fruits of our labor, a personalized recommendation model. Additionally, comparative analyses will be conducted to benchmark the model with an interface to interact with the developed model and its capabilities.

1.2 Objectives

The following objectives have been identified to develop a context-aware movie recommender system to address our research problem.

1. A critical review of literature about recommendation systems using LLMs for personalized recommendation generation.
2. Study of technologies adopted in implementing recommendation systems using LLM for context aware recommendation generation.
3. Design and develop LLM based context-aware movie recommender system.
4. Evaluate the recommender system model.

1.3 Background and motivation

In everyday life there are situations where we must make decisions while having or not having prior knowledge or information about options. To mitigate this, we seek recommendations from an expert who has experience in that specific aspect. This is the thinking behind recommender systems, it provides personalized recommendations using various techniques and algorithms to aid in the decision-making process over various domains. Over the past few decades several techniques have been proposed starting with Collaborative filtering, Content-based filtering, and Hybrid methods [7]. Collaborative filtering assumes that users with common interest in past will have common interests in the future, while in content-based filtering item-based information is used to match user profiles. These have some limitations such as cold start, data sparsity and scalability. Hybrid methods are a combination of the above two to address a part of these limitations.

Now, with the development of LLMs and abundance of meta-data and plethora of item level information, the personalized recommendation problem can be considered an NLP task. LLMs' ability to understand and generate human-like responses combined with large amounts of item specific data, allows for deeper understating of item context, and identifying user preferences. Many researchers have tried their hand in using LLMs in various ways to improve upon recommendation systems and it's still growing and improving as there are daily developments in the LLM development field. Traditional systems, with static user profiles and limited contextual capabilities are unable to efficiently provide personalized recommendations to users.

Therefore, the motivation in this research project is twofold: to harness the capabilities of LLMs to better understand and interpret contextual information in movie item data and to integrate this contextual understating into a context-aware personalized recommendation system. This approach aims to improve the accuracy of personalized movie recommendations with relevant and timely suggestions.

1.4 Problem definition

In view of the above concerns and opportunities, an effective approach to developing a recommendation system utilizing LLMs has been identified. According to the analysis, there are still different ways to approach recommender systems using LLMs. The ultimate objective of this project is to develop a personalized movie recommender system using LLMs and its underlying technology.

1.5 LLM based Context-aware Movie Recommender System

The main objective of this approach is to provide an LLM based personalized recommender system for movies. The hypothesis, input, output process and features are defined as follows.

The hypothesis of our project is that an optimal personalized recommendation can be provided to movie enthusiasts using generative AI technologies. LLMs and its embedding model can be utilized for this. The inspiration for this hypothesis comes from the study of numerous research on LLMs and vector embeddings' application in recommender systems in recent years.

The basic goal is to collect contextual data on movies and vectorize them using an embedding model of an LLM. The input data is collected from MovieLens and TMDb API for matching IMDb IDs. This data is then vectorized using OpenAI's text-embedding-3-large embedding model and stored in Weaviate, the selected vector database. Then two different recommendation models will be generated for users using centroid reference vectors. These two will be compared against a baseline collaborative filtering (CF) approach on different metrics. The interface will help show recommended movies depending on user taste and have capability of semantically searching movies as a perk of using a vector database. The use of OpenAI can help explain why movies were recommended in the semantic search feature. Overall, the recommendation system will be using the best of the LLMs offerings.

1.6 System requirements

The hardware and software requirements for the system development are identified as follows; Personal computer with core i5 processor or higher, Visual studio code and python libraries. MovieLens datasets will be used with open-source software and services where applicable.

1.7 Structure of the thesis

The structure of this research thesis is organized as follows. The next chapters thoroughly go through a critical review of literature about recommender systems using LLMs. Then a thorough go through the technology used to solve the research problem is defined. Then the process of achieving the solution is defined, followed by design, implementation, and evaluation chapters. Finally, the conclusion of the project is given followed by references and the appendix.

1.8 Summary

This chapter presented an introduction to the research topic, the objectives identified to complete this research, the background and motivation of the research, the problem definition, system requirements of the project and the structure of the thesis. More importantly we have mentioned the problem definition of our research. In chapter 2 we give a critical review of literature in recommender systems using LLMs for personalized recommendations.

CHAPTER 2

LITERATURE REVIEW ON RECOMMENDATION SYSTEMS USING LLMs

2.1 Introduction

Chapter 1 has presented the overall picture of this thesis by defining the research problem and related background information to solve the problem. This chapter presents a comprehensive literature review on recommendations systems and technologies used in building them. Here we have structured the literature review under three sections namely, Early developments in recommender systems, latest approaches towards recommender systems and future trends in recommender systems. This chapter also summarizes the literature in terms of their contribution, limitation and the technology/ methodology used. Finally, we define the research problem.

2.2 Early developments in Recommender Systems

We frequently encounter situations in daily life where we must take a decision without having any prior knowledge of the available possibilities. In such situations, relying on advice from those who have knowledge in that area becomes crucial. This necessity gave birth to the concept of recommender systems which began to take shape in the early 1990s. Today it is widely used by many cooperations to recommend products to consumers. The authors of the first recommender system, Tapestry [8] introduced the concept of collaborative filtering, which uses collective insight to improve recommendations. Tapestry was an experimental design to receive, filter, file and browse electronic documents. Recommender systems typically use two types of data; (i) user-item interactions, which includes ratings or purchasing patterns/ history, and (ii) attributes of users and items, which include as profile details or keywords [9]. Collaborative filtering methods use the first type of data, and content-based recommender systems use the second type. There are other approaches as well, such as knowledge-based and demographic recommender systems. Data sparsity, scalability, and gray sheep issue are typically the limitations of collaborative filtering [10]. Additionally, there are several drawbacks to content-based filtering, such as limited content analysis, serendipity, and new users. Other recommender systems integrate these many approaches to create hybrid recommender systems that blend the strengths of various systems and lessen the limitations.

Collaborative filtering models use the ratings data provided by multiple users to make recommendations. The fundamentals of the collaborative filtering method are that missing ratings values can be calculated because the rating values are often correlated among many users and items. Most models of collaborative filtering focuses on using either inter-item or inter-user relationships for the prediction task [11]. In content-based recommendation systems, the items are suggested based on their features and descriptions. These systems are useful when there isn't enough data for items. If a user has rated an item with similar descriptions the system can use these data along with descriptions to recommend new items. Knowledge based systems recommend items according to a users' needs and how well it meets the user's needs. Demographic systems recommend items based on user related demographic data by analyzing user patterns and preferences within demographic user groups. Each different type of recommender system has its advantages and limitations. For example, collaborative filtering relies on user ratings, content-based systems on item descriptions, knowledge-based systems on specific knowledge on user needs, and Demographic system on user demographic data. Hybrid systems combine features from these different types of recommendation systems to improve accuracy and effectiveness, by using multiple data sources [12].

However, all these recommender systems have limitations. The cold start problem occurs when there's no data available for new users or items, making it impossible to predict preferences. The grey sheep problem happens in collaborative filtering when a user's ratings don't align with any group, leading to poor recommendations. Lastly, changing trends and preferences can make data quickly outdated, affecting the reliability of recommendations based on historical data.

2.3 Recommender Systems with Artificial Intelligence

Artificial intelligence (AI) is a rapidly evolving field that is used to automate smart behaviors across various areas such as handling knowledge, decision making, planning, interacting perceiving and moving [13]. In particular, AI techniques play a crucial role in recommender systems, which are designed to recommend relevant items to users based on their preferences. In these systems, different AI technologies have been used to handle and improve recommendation. For example, neural networks are used to learn from data to predict user preference effectively. Transfer learning techniques use data from different domains to improve recommendations in domains where data is unavailable. Active learning uses selection of specific items for users to rate which is then used to improve recommendations. Fuzzy logic is also used to deal with uncertain and incomplete data. Evolutionary algorithms optimize recommendations by using different strategies or creating user/ item profiles. Furthermore, recent advancements in deep learning techniques have integrated RNNs and CNNs to utilize language and visual information for enhanced recommendations [7]. Overall,

AI in recommender systems helps personalizing suggestions, makings it easier for users to find item they desire despite the limited data and difference in user preferences. In recommender systems, the main goal is to rank items in a way that matches user preference, which is different form how neural networks operate. Early on, a two-layer restricted Boltzmann machine was used to explore how rating were ordered [14], during the 2009 Netflix prize competition period.

LLMs have emerged as powerful tools in the field of NLP recently and have gained attention in the domain of recommender systems. LLMs are language models that are built using the deep learning technique called transformers, first introduced in a paper titled "Attention is all you need" [6]. The transformer architecture uses the self-attention mechanism which is known for its efficiency and effectiveness in handling sequences of text. As the title describes the attention mechanism is the key component of this technique. It allows the model to weigh the importance of words in a sentence irrespective of the position of words and is more effective than previous architectures of RNNs and LSTMs. This helps the model learn context and relationships between words more effectively. LLMs are trained using large amounts of textual data gained from websites, articles, books, and other forms of written data. These data are tokenized before training, where words are split depending on the models' design. Then each token is mapped to a vector through embeddings. Embeddings are high-dimensional representations of the token which capture the semantic meaning of the tokens. This means words with similar meanings are positioned nearby in the embedding space. The transformer architecture consists of an Encoder and Decoder, where the Encoder process input data and the Decoder generates the output. Different LLM models use different combinations of these key concepts to build their model. This is what makes LLMs such as BERT, GPT, PaLM, LLAMA, etc. different from each other. These models have disrupted the NLP field and have been used in various capacities in recommendation systems across literature. The main advantage of using LLMs in recommendation systems is the ability to obtain high-quality interpretations of source data features by using the underlying embedding model. When utilizing a LLM for specific tasks there are three main approaches widely used, such as Fine-tuning the LLM, Prompting and RLHF. When it comes to fine-tuning, a pre-trained LLM model is further trained on a smaller, task specific dataset such as user-item interactions, item descriptions, and user profiles. This process adjusts the model's weights based on the data used and can be a costly task to undertake. Prompting is where carefully crafted prompts are used to guide the model to generate a desired output. This doesn't modify the LLM's weights and can be done in various methods. Zero-shot learning, Few-shot learning, and Chain of Thought (CoT) prompting are the prompting techniques that can be used. Zero-shot learning only requires a descriptive prompt while few-shot learning uses input-output pairs as an example in the prompt itself. CoT prompting guides the LLM to follow logical reasoning to generate an output. RLHF is a more complex system where the LLM is fine-tuned based on feedback by humans on the LLM outputs. This helps align the LLM output to the preferred output based on human feedback. RAG (Retrieval-Augmented Generation) is another methods used to integrate external knowledge to a LLM [15]. Introduced by researchers at Facebook, it is used to fetch

relevant information for prompts that the LLM model hasn't enough information about. This can be due to the cutoff period of data, or unavailability of domain specific data in the data that has been used to train the LLM. By using RAG, it ensures that the LLMs' outputs are relevant and updated. Although there are many advantages and various methods to incorporate LLMs in day-to-day task we should be aware of its limitations too. LLMs are resource intensive, have data biases, are a black box as any other ML model is, have scalability issues, prone to adversarial attacks and can hallucinate and make up things that do not exist. Keeping that in mind the use of LLMs in recommender systems will be discussed in the following section.

2.4 LLMs in Recommender Systems

The use of LLMs in recommender systems has been identified and broken down into three main categories [1]. Namely LLM Embeddings with Recommender system, LLM Tokens with Recommender system, and LLM as Recommender system. In LLM Embeddings with Recommender system, the LLM acts as the feature extractor where it generates embeddings based on item and user data and then uses traditional recommendation systems to generate recommendations. In LLM Tokens with Recommender system, instead of generating embeddings, tokens are generated for items and user data. These tokens are then used in the decision-making process of recommendation systems for user specific recommendations. In LLM as Recommender system, it uses the pre-trained LLM directly as the recommendation system. This method uses variations of input data and prompts generate recommendations as the output. Also, these three methods are also categorized by the capability of the LLM. The first one is Discriminative LLMs for recommendation, where LLM Embeddings with Recommender systems utilize a Discriminative LLM for recommendation because such LLMs have the capability to classify input data. Both LLM Tokens with Recommender system and LLM as Recommender system utilize Generative LLM for recommendation as these generate textual content for inputs and outputs. Therefore, they are categorized into Generative LLMs for recommendation. Discriminative models are useful for detailed feature analysis while generative models are better suited for creating new textual content and direct recommendations. Figure 2.1 depicts the breakdown of different methods of using LLMs for recommendation.

In real-world applications, the selection of LLM impacts the capabilities and the final design and implementation of the recommender system. Discriminative LLMs are designed to recognize differences between categories or classes of inputs, they are used for tasks where the goal is to predict a specific outcome given a set of features. The BERT family of language models introduced by google research [3] has been widely utilized as the discriminative LLM in the recommender system in many research due to its ability to be classify and categorize textual data. Generative LLMs such as GPT series and LLaMA series language models have

improved natural language generation capacities compared to discriminative models. Generative model-based recommendations are natural language tasks which utilize in-context learning, prompt-tuning and instruct tuning to generate recommendations.

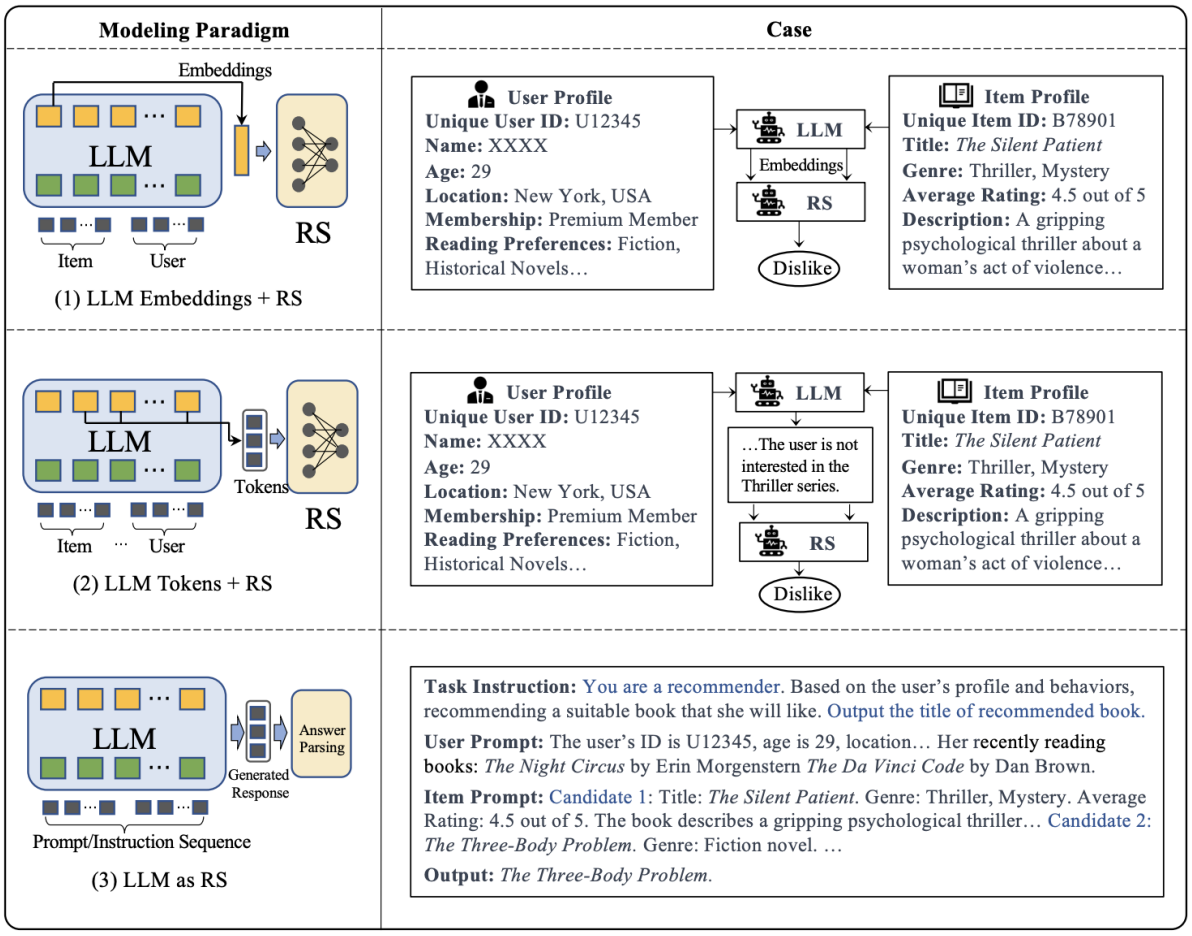

Figure 2. 2 Methods of using LLMs in Recommendation systems [1].

U-BERT [16] model uses the BERT architecture to encode user preferences. It addresses the challenge of sparse behavioral data in less popular domains by using content-rich domains in its pe-training phase. The fine-tuning phase introduces an item encoder and a review co-matching layer to acquire semantic relations between users and reviews. The UserBERT [17] paper introduces a contrastive pre-training approach for user modeling using unlabeled user behavior data. It uses masked behavior prediction as well as behavior sequence matching techniques to identify user behavior. It also uses medium-hard negative sampling framework to select highly negative samples to learn discriminative features of users. These methods are used to improve user models and outperform baseline methods. The BECR [18] model uses a lightweight BERT-based re-ranking approach with traditional lexical term matching to

generate recommendations. It uses pre-computed token embeddings to re-rank the output and achieves fast inference on CPU servers.

The ONCE [19] framework introduces a personalized content-based recommender system which is flexible across different domains by using both open-source and closed-source LLMs. This model helps enhance the item context by using the open-source LLM as a content encoder and closed-source LLM for enriching token-level data. In Qin et al., 2024 [20] a Pairwise Ranking Prompting (PRP) approach is introduced where documents are ranked using mid-sized, open-source LLMs. It uses the same prompt template across multiple datasets to benchmark the performance against closed-source LLMs and is effective for text ranking use cases. The study Dai et al., 2023 [21] explores the application of ChatGPT for recommendation systems. It focuses on point-wise, pair-wise and list-wise ranking methods by using prompting based on the domain. It performs well in list-wise and pair-wise ranking methods while having challenges in point-wise ranking tasks. The KAR [22] framework incorporates reasoning and factual knowledge to the LLM to improve the context and relevance of recommendations. It uses factorization prompting and hybrid-expert adaptor that converts LLM output into vectors that is then used by the recommendation algorithm. The TALLRec [23] framework uses a two-stage tuning process with alpaca tuning and rec-tuning to generate recommendations for movie and book domains. It also has cross-domain generalization capability which is effective across various recommendation scenarios. In Li et al., 2023 [24] a prompt learning technique is used to enhance explainability in recommender systems. It adopts a discrete and continuous prompt learning strategy to use unique identifiers for users and items to generate semantic comprehension between data. The P5 paradigm [25] approaches recommendation systems by unifying various recommendation tasks into a single text-to-text model where knowledge is transfer learned across different tasks. It uses semantic understanding of the LLM for generating recommendations, by zero-shot and few-shot learning techniques. GLRec [26] introduces behavioral graph analysis for job recommendations using LLMs, where meta-path prompt constructor is used to generate graph interactions into natural language which are used in prompting.

In conclusion, LLMs have the capability to improve recommendations by various implementation techniques. The challenges faced by researchers such as complexity in models, quality of prompts, scalability and using mid-size LLMs need to be addressed. As we move forward, it is essential to continue exploring potential capabilities of LLMs to create innovative and impactful experiences for the public in the domain of recommendations.

2.5 Analysis of literature

After the comprehensive review of literature related to the problem domain, generating personalized recommendations using LLMs we have summarized the key finding in the below table. This table shows advantages, limitations, significant contributions, and technology used in research.

Table 2. 1 Analysis of Literature.

Research	Advantages	Limitations	Technology used
Qiu et al., 2021 [16]	Adopts to sparse data user data.	Complexity in model architecture	BERT, Embeddings for user representation, Pre-training and fine-tuning
Wu et al., 2021 [17]	Enhanced user modelling, Effective pre-training task	Complexity in negative sampling, dependance on quality of unlabeled data	BERT, Contrastive learning, Self-supervision task, Negative sampling framework
Yang et al., 2022 [18]	Efficient in compute, high relevance in recommendation	Costs in disk I/O	BERT, Composite token encoding, CPU-based inference
Liu et al., 2023 [19]	Better content understanding, Flexible across domains	Dependent on LLM used, Complex integration	ChatGPT, BERT, Semantic comprehension
Qin et al., 2024 [20]	Pairwise Ranking Prompting (PRP) to reduce computational cost. Improved efficiency	Using mid-sized open-source LLMs and not reaching full potential.	FlanUL2, Pairwise Ranking Prompting (PRP)
Dai et al., 2023 [21]	Ranking using point-wise, pair-wise and list-wise approaches in different domains	Dependency on quality of prompts	ChatGPT, Domain aware prompts
Xi et al., 2023 [22]	Integrating open-world knowledge	Complexity in implementation	Factorization prompting, Hybrid-Expert adapter

Bao et al., 2023 [23]	Cross domain generalization, Efficient	Hardware dependency, Use of mid-sized open-source LLM	LLaMA-7B, Alpaca tuning, rec-tuning
Li et al., 2023 [24]	Enhanced Explainability	Complexity in integrating user and item IDs for prompting	GPT2, Discrete and continuous prompt learning
Geng et al., 2023 [25]	Unified framework for different recommendation tasks	Complexity in implementation, Dependance on LLM model quality	T5, Prompting, zero-shot and few-shot learning, text-to-text transformations
Wu et al., 2023 [26]	Understanding behavioral graphs	Scalability and Bias	Meta-Path Prompt constructor, Path augmentation module

Fine-tuning pre-trained language models has gained significant attention in NLP tasks, including recommendation systems. It involves adapting a language model, which has learned from large-scale text data to a specific task by training it on relevant data such as user-item interactions, item descriptions, and user profiles. During fine-tuning, the model's parameters are updated based on the task-specific data, allowing it to specialize for recommendation tasks. Most BERT-enhanced recommendation methods follow this track. For instance, U-BERT [16] leverages content-rich domains to enhance user features with insufficient behavior data and includes a review co-matching layer to capture implicit semantic interactions. Similarly, UserBERT [17] uses self-supervision tasks for pre-training on unlabeled behavior data, incorporating medium-hard contrastive learning, masked behavior prediction, and behavior sequence matching to model user representations accurately. Additionally, the pre-trained BERT model has achieved breakthroughs in ranking tasks, as demonstrated by BECR [18], which combines deep contextual token interactions and traditional lexical term-matching features with a novel composite token encoding, balancing ad-hoc ranking relevance and efficiency through pre-computable token embeddings based on uni-grams and skip-n-grams.

Generative language models (LLMs) have demonstrated superior natural language generation capabilities compared to discriminative models, leading to a growing interest in applying generative models to recommendation tasks. This transformation leverages techniques like in-context learning, prompt tuning, and instruction tuning, with various strategies explored for adapting LLMs for recommendation, focusing on both non-tuning and tuning paradigms. For instance, GENRE [19] introduces three prompts to enhance news recommendation by refining

news titles, extracting profile keywords, and generating synthetic news, significantly improving traditional news recommendation models. PRP (Pairwise Ranking Prompting) [20] addresses list-wise recommendation challenges by scoring items through pairwise comparisons, though it is time-consuming during inference due to the complexity of processing these comparisons for large item sets. ChatREC [21] provides an empirical analysis evaluating ChatGPT's recommendation abilities on point-wise, pair-wise, and list-wise ranking tasks, proposing different prompts and role instructions for each task to enhance domain adaptation and overall performance. KAR (Knowledge-Aware Reasoning) [22] improves the understanding of user preferences by decomposing topics in textual descriptions, designing effective prompts to reduce noise and enhance recommendation accuracy, focusing on specific tasks to improve the reasoning and recommendation abilities of LLMs.

TALLRec [23] employs a two-stage tuning process. Initially, it is fine-tuned using self-instruct data from Alpaca. Subsequently, TALLRec undergoes recommendation tuning, where the input consists of a user's historical sequence, and the output is binary feedback ("yes or no"). This dual-stage approach leverages both general and recommendation-specific training data, enhancing the model's ability to predict user preferences accurately. GenRec [24] utilizes the generative capabilities of LLMs to directly produce the next item for recommendation. By converting user interaction data into prompts, the model generates potential items of interest. This approach highlights the use of LLMs' generative power to forecast user preferences, offering a novel method for generating recommendations based on historical interaction data. Instruction Tuning for Multitask Learning [25] in this work, a T5 model is fine-tuned on multiple types of instructions, including sequential recommendation, rating prediction, explanation generation, review summarization, and direct recommendation. This multitask instruction tuning enables the model to generalize better to unseen prompts and new items, showcasing enhanced zero-shot capabilities and adaptability to various recommendation tasks. GLRec [26] introduces a meta-path prompt constructor to interpret behavior graphs, enhancing the interpretability and performance of recommendations. This method also incorporates a path augmentation module to address prompt bias. Furthermore, it aligns unpaired low-quality resumes with high-quality generated ones using Generative Adversarial Networks (GANs), refining resume representations and improving recommendation outcomes in job-resume matching scenarios.

2.6 Problem definition

Our literature review has revealed many limitations in early developments of recommender systems utilizing LLMs such as complexity, quality of prompts, scalability and LLM model quality. More importantly many researchers have recognized that LLM related development is still ongoing and LLM related recommender systems are still an open research area. Therefore, we define our research problem as an approach to utilize the embedding model of LLMs and vector databases to improve personalized movie recommendation systems.

Therefore, there is a high curiosity to investigate the applicability of RAG architecture to model a personalized movie recommendation system solution using LLMs underlying technology.

2.7 Summary

This chapter presented a critical review of research in recommendation systems using LLMs for personalized recommendation generation. Based on the review we have summarized advantages, limitations and technology used in major research in recommender systems. More importantly we have defined our research problem as improving personalized recommendation using context-aware recommendation with LLMs as it is an open research area with new developments every day. Additionally, we have completed our first objective out of the four that we have identified in chapter one. In chapter 3 we give an in-depth study of the technology adopted for solving the research problem.

CHAPTER 3

TECHNOLOGY ADOPTED

3.1 Introduction

In the second chapter, we did a comprehensive review of literature on LLMs in personalized recommendation generation tasks. This chapter delves into the technologies used in solving our research problem. From LLMs and their underlying technology, vector databases, to model evaluation techniques.

3.2 Overview of the Technologies Used for LLM based context-aware Movie Recommendation

LLMs are generative models where the model understands and generates human-like responses. These models are trained on diverse datasets of books, articles, websites, and other textual sources, which allow them to develop broad understanding of human knowledge and language. LLMs can assist in a variety of applications depending on their architecture, training data and model size. The transformer architecture [6] introduced in 2017 is the breakthrough that has been able to facilitate all this development in LLMs. OpenAI's ChatGPT series, Meta's LLaMA series, Google's T5, BERT and other models and various open-sourced models available in Hugging Face platform have contributed to democratizing LLMs and allowing researchers' and developers' access state-of-the-art models. LLMs utilizes a tokenization process which breaks words down into smaller tokens. Tokenization methods include word, sub-word and character-level break down of words, and different LLMs use different methods. The change in tokenization method improved OpenAI's ChatGPT series from ChatGPT3 to ChatGPT4 with enhanced capabilities. Therefore, tokenization is an essential part of LLMs. The next crucial component of LLMs is the embedding model. This is where words are converted into numerical vectors that capture their semantic properties based on their context in the training data. This enables the model to process and understand text inputs and generate relevant outputs.

RAG is a hybrid model that combines the retrieval and generation capability of LLMs. Developed by Facebook AI, it operates by retrieving relevant documents from a dataset depending on semantic search as vectors and then using these vector documents with the LLM model to generate relevant responses. This process allows incorporating external knowledge to pre-trained LLMs, making answers detailed and factual. In our approach we use this vectorization part of RAG to create vector embeddings for movies, where all its context

information is condensed to one single vector. These vector embeddings are stored in vector databases.

Vector databases are specialized databases designed for storing, indexing, and retrieving vector embeddings efficiently. These vectors are basically a list of float values that represent the input data depending on the LLMs tokenization technique used. By organizing these vectors in a vector database, we can retrieve semantically similar items, when needed. When it comes to the indexing mechanism of vector databases the most widely used algorithm is HNSW (Hierarchical Navigable Small World). It creates a layered structure that allows navigating to the closest vectors efficiently. Vector databases use Cosine distance, Dot product, L2-Squared distance, Hamming distance, and Manhattan Distance metrics to measure the distance between two vectors. Multiple open-source vector databases that support real-time vector search and efficient storage of vector data have emerged in recent years, namely Weaviate, Chroma DB, Pinecone etc. We use Weaviate as our vector database in our research.

3.3 Embedding Model

Embeddings are numerical representations of input information, which simplify the processing for LLMs. These vectors capture the semantic meaning of text and can be used for various applications such as semantic search, clustering, and machine learning tasks. OpenAI has a variety of embedding models and we have chosen Text-embedding-3-large embeddings model which has a vector of 3072 dimensions. This means a single input generates a 3072-dimension vector to represent the input data. We have chosen this model by experimenting on a sample dataset and comparing evaluation metrics against other models. OpenAI provides embedding models such as Ada-002, Text-embedding-3-small and Text-embedding-3-large which has 1536, 1536, 3072 dimensions respectively. This embedding model offers superior performance compared to the other models. Also, these embeddings have been fine-tuned to work with ChatGPT, giving us additional features such as explainability.

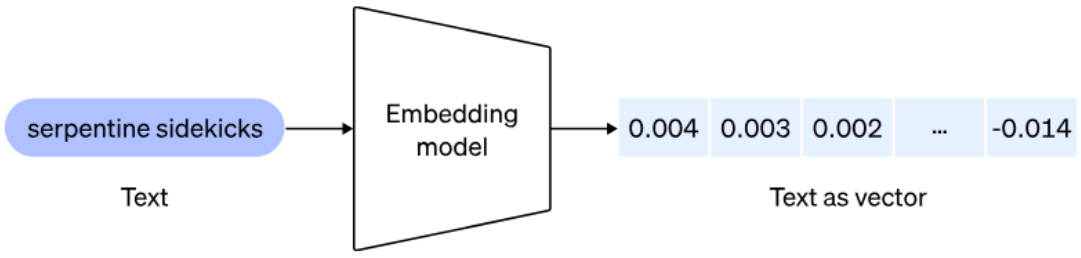

Figure 3. 1 Text to vector conversion by an Embedding model.

3.5 Vector Databases

Vector databases are specialized databases used for storing, indexing, and retrieving vectors efficiently. For this task we use the Weaviate vector database which is an open-source vector database. When ingesting data, we must make sure to model the data appropriately to get the best output possible. Weaviate does the indexing real-time as we ingest data to the vector database using HNSW algorithm mentioned above. We also let Weaviate create a UUID for each movie ID rather than adding it ourselves. When generating centroid reference vectors, we only need to get relevant vectors with UUID and IMDb ID data to do the vector calculations. We ingest IMDb ID and several other data that we have set not to vectorize when creating the vector database so that when generating recommendations and displaying them in the interface the retrieval is fast. Also, in the generating recommendations step, we retrieve 10 closest vectors near the centroid reference vector of the user with required information to display title and poster. When searching for the 10 closest vectors we use the L2-squared distance, so the lowest distance means the closest vector to the centroid reference vector. The distance metric was chosen among cosine, dot, l2-squared, hamming and Manhattan metrics by experimenting on a sample dataset and comparing evaluation metrics against other models.

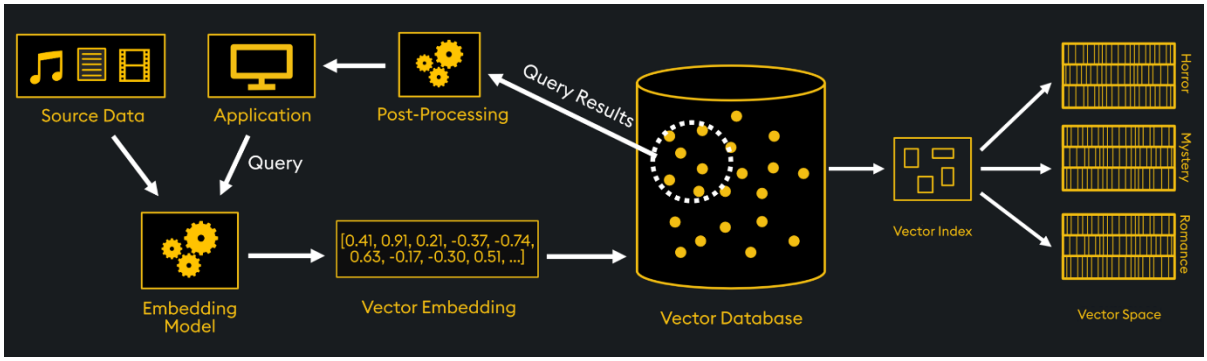

Figure 3. 2 Vector Database process architecture

3.6 Model Evaluation Methods

To assess the effectiveness of our recommendation models, we employ a variety of model evaluation methods. These include traditional metrics like precision, recall, F1 score as well as more specialized measures such as hit rate. All these metrics are calculated for top K at depths K = 5 and K = 10. These metrics help quantify how well the system performs on user recommendation based on user-rating data of movies in the MovieLens dataset.

3.7 Summary

This chapter has outlined the core technological components utilized in the research problem to develop a context-aware personalized recommender system for movies. This concludes the second objective identified in the first chapter of this document. In the following chapter we will discuss the approach taken to develop this system in detail.

CHAPTER 4

LLM BASED CONTEXT-AWARE MOVIE RECOMMENDER SYSTEM APPROACH

4.1 Introduction

In chapter 3 we presented an in-depth study of technologies adopted for LLM based context-aware movie recommender system with an emphasis on LLMs and their embedding models. This chapter reports on our approach to LLM based context-aware movie recommender system. This chapter is sub-divided into six primary sections, namely hypothesis, input, output, process, users, and features.

4.2 Hypothesis

In this project we hypothesize that an optimal personalized recommendation can be provided to movie enthusiasts using generative AI technologies. LLMs and its embedding model can be utilized for this. The inspiration for this hypothesis comes from the study of numerous publications on LLMs and vector embeddings' application in recommender systems in recent years. The recommender system will provide personalized movie recommendations based on user-movie interactions.

4.3 Input to the Recommender System

The input data required for the system comes from two separate sources. This data will be static throughout the research project. The first and main source of data is the MovieLens 20M dataset by GroupLens. This dataset is widely recognized as a benchmark dataset that has been cited in several publications. GroupLens is headed by the department of computer science and engineering at the University of Minnesota. In addition to this dataset to improve upon the context of the movie, we retrieve movie-related data from TMDb (The Movie Database) a community-built movie and TV database. By combining these two datasets we create a comprehensive dataset with additional context which is trivial for the personalized recommendation process. In the following subsections, we provide a more thorough explanation of the datasets used in our analysis.

4.3.1 The MovieLens dataset

The MovieLens 20M dataset contains 20 million user-movie ratings and 465,000 tags for 27,000 movies. This data was generated by 138,000 unique users from January1995, to March 2015. Therefore, this data only contains movies released before March 3, 2015. Users have been selected at random and have rating values for a minimum of 20 movies per user. No user-related information is available for users and each user is represented by an ID. The data are in six csv files, namely genome-scores, genome-tags, link, movies, ratings, and tags.

There are two main IDs to consider when it comes to interconnecting all these files. The userId is an integer value that been anonymized and is consistent between ratings and tags files. The movieId is also an integer value that has at least one rating or tag included in the dataset. The movieId is consistent between ratings, tags, movies, and links files.

Movie information is in the file named movies and contains the columns movieId, title, and genres. All ratings are in the ratings file and have the column names userId, movieId, rating, timestamp. The ratings are on a five-star scale with half-star increments. The links file has links to other movie databases and has the column name format of movieId, imdbId, and tmdbId. For every movieId in movieLens data, there is the imdbId which is an identifier for movies used by IMDb and tmdbId which is an identifier for movies used by TMDb. This file is another major reason for choosing this dataset as the imdbId and tmdbId values can be used to gather additional context.

All tags are in the tags file and have tags applied by users to movies. Tags are metadata generated by users regarding the movies. Each tag is having either a single word or short tag line. The tag genome encodes how a movie presents a particular property. The genome data are into two separate files. The file genome-scores has movie-tag data while the second file, genome-tags has the tag descriptions.

4.3.2 The TMDb dataset

The TMDb dataset is generated using a script to improve the context of the movie data that is present in the MovieLens dataset. The links file in MovieLens dataset contains both imdbId and tmdbId for a given movie and we use the imdbId to retrieve additional movie related data. The following is a list of available data points in TMDb.

1. Title: The official title of the movie.
2. Original Title: The title in the original language.
3. Genres: Categories like action, comedy, horror, etc.
4. Release Date: The official release date of the movie.
5. Runtime: The duration of the movie in minutes.

6. Overview: A summary of the movie's plot.
7. Tagline: A notable expression or slogan from the movie.
8. Cast: List of actors in the movie.
9. Crew: Details about the director, writers, and other crew members.
10. Production Companies: Names of the companies that produced the movie.
11. Production Countries: Country/ countries where the movie was produced.
12. Spoken Languages: Languages used in dialogues of the movie.
13. Budget: The budget of the movie.
14. Revenue: Box office revenue generated by the movie.
15. Popularity: A metric that reflects the current popularity of the movie on TMDb.
16. Vote Average: Average user rating on TMDb.
17. Vote Count: The number of votes that contributed to the average rating.
18. Poster Path: URL to the movie's poster image.
19. Backdrop Path: URL to a background image associated with the movie.
20. Status: Status of the movie (e.g., Released, Postproduction).
21. Adult: Boolean indicating whether the movie is for adults.
22. IMDb ID: The IMDb ID we used to query.
23. External Links: Links to the movie's pages on other sites like IMDb, Rotten Tomatoes, etc.
24. Videos: Trailers, teasers, and other videos related to the movie.
25. Keywords: Related keywords or themes associated with the movie.

4.4 Output of the Recommender System

The LLM based context-aware movie recommender system has been designed to produce a personalized movie recommendation based on the user's movie interaction. The key output of movie recommendations will be generated by the weaviate vector database with respect to the individual user vector. The user vector is generated as a centroid vector of the users interacted movie vectors. The final output is made available via a web page interface.

4.5 Process

Here a comprehensive approach to developing an LLM based context-aware movie recommender system is described. The overall top-level architecture of the recommender system is as follows. Data gathering and preprocessing, data vectorization and implementing vector database, generating user vectors, generating recommendations, and visualizing recommendations.

Our modelling process begins with cleaning the MovieLens data. We use the movies and links file to generate a dataset with movielens movie data. Then using the imdbId and TMDb API we retrieve additional data to improve movie context using a python script. Once the input data is prepared, we use Weaviate with the OpenAIs text-embedding-3-large embedding model to generate and store vector data for each movie. This step maps the context-rich movie data to a multi-dimensional vector space where a movie can be depicted using one single vector. Weaviate can perform vector similarity and querying capabilities which is used in the generation of recommendations.

To generate user vectors, we employ two distinct methods. The first method involves selecting movies that a user has rated 3 or above on the 5-star scale, and then generating a centroid vector that represents the user's taste. In Figure 4.2 we have depicted this scenario in 3-dimensions (the text-embedding-3-large embedding model generates 3072-dimensions). Here movie vectors are denoted in black, and user rated movie vectors are denoted in green. The resulting centroid vector for user according to model 1 is depicted in red. The second method considers all the user-movie rating interactions by generating the centroid vector on a rating-weighed movie vector. These centroid vectors are fundamental for recognizing user preferences (user vector) and can be used to retrieve similar movie vectors based on the centroid vector. This allows the generation of personalized movie recommendations.

Both approaches are evaluated using the following metrics to calculate the effectiveness and accuracy of our recommendations. We use hit rate at 5 and 10 (hr@5, hr@10) as well as normalized discounted cumulative gain at 5 and 10 (ndcg@5, ndcg@10) to measure the performance of our recommendation system. These metrics help us understand how well our system performs in real-world scenarios.

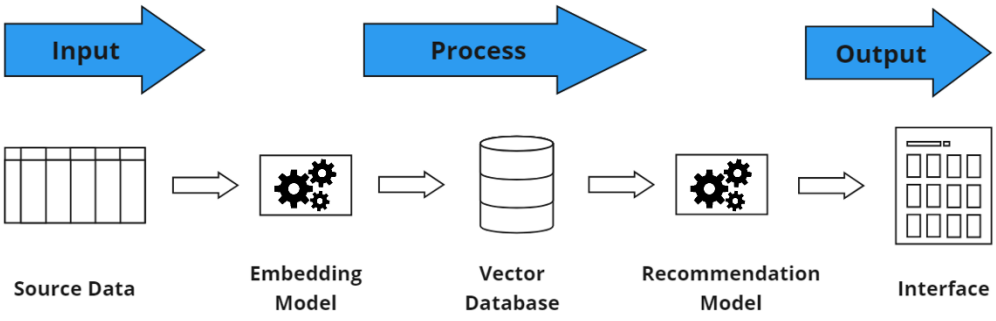

Figure 4. 1 High-level framework of proposed solution.

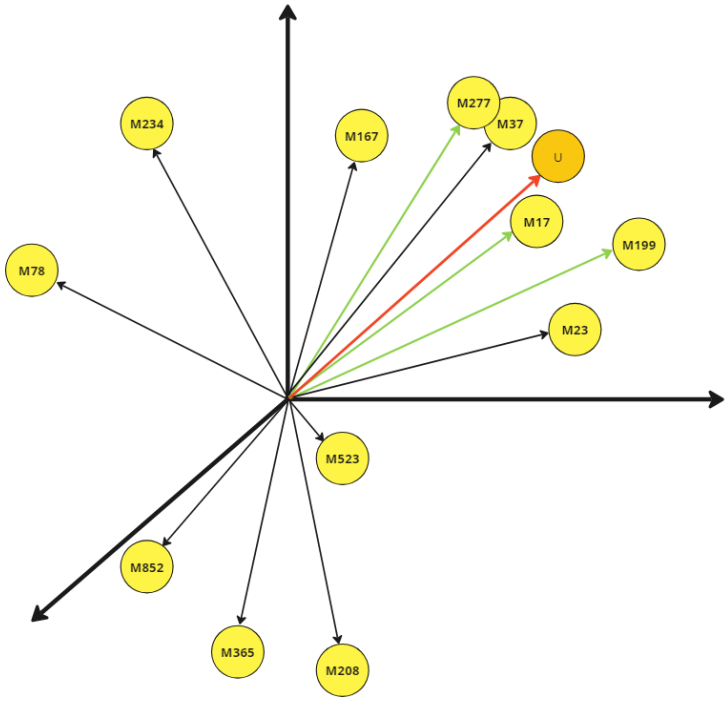

Figure 4. 2 Centroid reference vector generation in embedding space.

4.5.1 Model 1 Algorithm

The algorithm definition for the first model is as follows.

Input: U (user ratings), where $U[u]$ is the set of movies rated 3-stars or above by user u.

Vector Representation: Each movie m has a vector representation V_m in the vector database.

Centroid Calculation:

$$C_u = \frac{1}{|U[u]|} \sum_{m \in U[u]} V_m$$

Here, C_u is the centroid vector for user u, calculated as the mean of the vectors of the movies that user u has rated 3-stars or above.

Recommendation Generation:

$$\boldsymbol{R_u = top_k\{-\| C_u - V_m \|_2^2 \mid m \in M\}}$$

R_u is the set of top k recommended movies for user u, selected based on the smallest L2-squared distance between C_u and all movie vectors V_m in the database M.

4.5.3 Model 2 Algorithm

The algorithm definition for the second model is as follows.

Input: *U* (user ratings), where *U[u]* includes all movies rated by user *u* and their ratings r_m.

Vector Representation: Each movie *m* has a vector representation V_m in the vector database.

Centroid Calculation:

$$C_u = \frac{1}{|U[u]|} \sum_{m \in U[u]} (\frac{r_m}{5}) \cdot V_m$$

Here, each movie vector V_m is weighted by the normalized rating $(r_m /5)$ where r_m is the rating given by the user *u* to the movie *m*, and 5 is the maximum possible rating. This weighting scales the influence of each movie based on how much the user liked it. It is normalized to a scale of 0 to 1.

The denominator simply counts the number of movies rated by the user *u*, used to normalize the weighted sum of vectors.

Recommendation Generation:

$$\boldsymbol{R_u = top_k\{-\| C_u - V_m \|_2^2 \mid m \in M\}}$$

R_u is the set of top *k* recommended movies for user *u*, selected based on the smallest L2-squared distance between C_u and all movie vectors V_m in the database *M*.

4.6 Users

Our approach and model outcome have the potential to serve a wide range of users. Primarily movie enthusiasts who seek personalized movie recommendations tailored to their taste. Also, this system is valuable for families or groups with diverse tastes as they can search for movies using keywords. This makes the movie selection process more efficient, enjoyable and ensures that users spend less time searching and more time enjoying content.

4.7 Features

Our approach has several features that enhance user interaction and satisfaction.

- Providing personalized movie recommendations that align closely to the users' interests.
- Keyword search functionality where a user can find movies by the context data used to generate the movie vector. This is useful when people know what they want to watch or look for movies like their favorites or in a group setting.
- Movie recommendation explainability, by utilizing OpenAIs ChatGPT model and vectorized movie data in weaviate.
- Visually appealing recommender dashboard consisting of movie posters.

4.8 Summary

This chapter presents the approach adapted for developing a LLM based context-aware movie recommender system under the subheadings of hypothesis, input, output, process, users, and features. In the forthcoming chapter, we will discuss the design of our research approach.

CHAPTER 5

DESIGN OF CONTEXT-AWARE MOVIE RECOMMENDER SYSTEM USING LLMS

5.1 Introduction

The previous chapter described the approach for the LLM based context-aware movie recommender system, outlining the hypothesis, input, output, process, users, and features. This chapter focuses on the system design and their functionality for system development and operation. The design of the implemented recommender system will be described in detail, elaborating the main system modules in this chapter.

5.2 Top Level Architecture

The proposed LLM based context-aware movie recommender system is focused on providing personalized recommendations based on user-movie interaction data by utilizing LLMs embedding models. As shown in Figure 5.1, the recommender system consists of four main layers. The data gathering layer is the first layer, which includes the data collection and enrichment module for gathering additional contextual information for movies. The user-movie rating data is also handled by this module, by updating it as user-movie interactions come in. The second layer consists of the primary functional component of the system, which is the vector database that stores movie data and provides search capabilities within the stored data. The vector database stores embedding vectors of movies that are generated by OpenAIs text-embedding-3-large embedding model. The third layer consists of the model for personalizing movie recommendations. The model calculates a user-vector based on the users-movie interaction data and produces the final recommender outputs. There are two major user-vector calculations based upon weaviate's ref2vec-centroid implementation. The final layer is the presentation layer which is a friendly user interface for providing personalized recommendations. This can give real-time movie recommendations based upon user-movie ratings. The user interface also incorporates a movie discovery section where the user can search for movies semantically which is an additional capability of using a vector database.

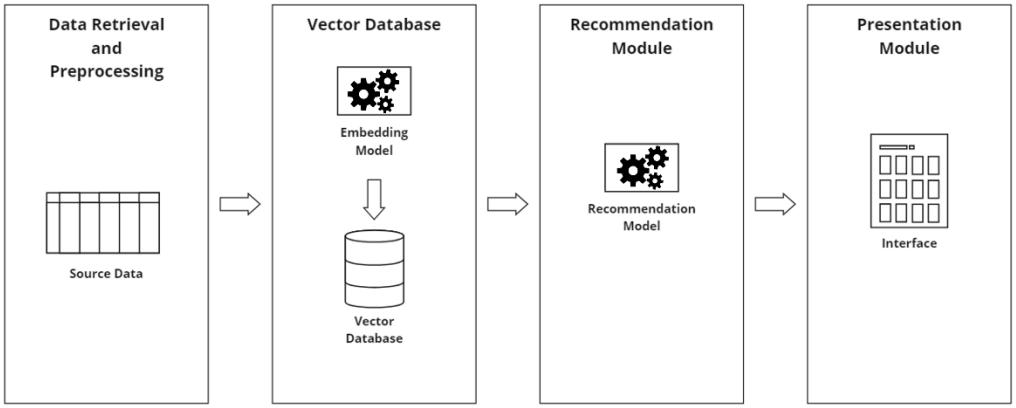

Figure 5. 1 System module architecture for proposed system.

5.3 Data Retrieval and Preprocessing

Data gathering and preprocessing is the crucial first step in building the LLM based context-aware movie recommender system. This step ensures that the data is clean, relevant, and structured properly for vector database ingestion. The publicly available MovieLens 20M dataset is used as the main data source for user-movie rating data. For movies are present in the movies file and the accompanying imdbId is retrieved from the links file. With the relevant movielens movie_id and imdbId combination we can retrieve movie related additional data from TMDb service. This data will remain static and will remain in the local environment. This data will not get updated as time goes on, therefore this is a one-time process. If this system is developed further without having constraints on the MovieLens data, then there is the possibility of regularly retrieving movie related data. Although the TMDb dataset doesn't include Oscar/ golden globes nominations and winning, it has dynamic data such as box office revenue, popularity, vote average, vote count, and external links which we are not interested in the current scope of the project.

5.4 Vector Database

The architecture of our model is centered around the use of vector embeddings generated from context rich movie data. Therefore, the core component of this project is a vector database. With recent developments in generative AI, vector databases have become increasingly popular among LLM related application developers. Weaviate is one such vector database that is open source, scalable and has real time indexing capabilities. One other important aspect was the ref2vec-centroid implementation that we have taken inspiration from to develop the

personalized recommendation for users. The Weaviate vector database creation is also a one-time implementation because the input data is static. The data ingestion to weaviate triggers the process of creating the vectors through the use of OpenAI's text-embedding-3-large embedding model through API calls. This also does the real-time indexing for the input vectorized data. Once the vector database is created, we use the movie vector embeddings related to a users' rating history to create a user-vector and this vector in turn is used to get recommendations by querying near vectors in the vector database. Therefore, the quality of input data to the vector database is important. One additional benefit of using a vector database is its inherent semantic search capability. This capability is later used as one of the features where the user can search for movies using keywords and phrases. Also, since the vector database is connected to OpenAI for vector creation, it can also be used as a generative system where explainability can be provided to recommendations. This can be a costly endeavor, but still possible as the vector database acts as a RAG (Retrieval Augmented Generation) method where we are supplying the ChatGPT model with additional information about the movies.

5.5 Recommendation Module

The recommendation module produces personalized recommendations for each individual user. The module is developed based on the ref2vec-centroid implementation by weaviate. This implementation is built on generating a vector based on the centroid of all the reference vectors. Basically, the average of all the reference vectors. Based on this we assume that the centroid vector represents the user's movie taste. To implement this and for comparison purposes we implement two separate user-vectors using the centroid vector approach. Before we go into the centroid vector creation, we should understand that the user-ratings are based on a five-star system and have half-star increases. We consider anything equal and above 3 as a good rating. Keeping this in mind, in our first centroid vector approach we get the centroid vector of 3-star-plus movies for each user. Also, it is verified that each user has at least 20 movies rated after the 3-star-plus ratings filter is applied. After the centroid vector is found for each user, we query the vector database for the 10 nearest movie-vectors based on L2-squared distance. The second centroid vector creation relies on multiplying the rating weight (rating/5) by the movie vector. Then generating the centroid for the updated vectors for each user. After this step, like in the first approach we retrieve 10 nearest movie vectors based on the L2-squared distance to the centroid vector. For this method also, we make sure that at least 20 user-movie ratings are present.

5.6 Presentation Module

The projects presentation module is built on streamlit, a python-based web app builder for sharing data-science models. The main web page consists of 3 pages, the first one being the recommender page. This page shows the recommended movies for all the users, where the user can be filtered. It consists of movie posters ordered in ascending order of L2-squared distance. This page has a toggle to show movies that the user has already watched and to show other movies that the user has never seen. It also can switch in between the two types of centroid vector approaches. This page also has the real-time demo capability so that real-time recommendations can be generated by rating movies based on the two centroid vector approaches. The next tab is about the movie search capability based on keywords. This also has the capability to explain the recommendation and can be demoed. The final tab has the user's movie rating history as the movielens data has timestamp values, which can be used as a reference for recommendations generated.

5.7 Summary

This chapter has elaborated in detail, the design of our LLM based context-aware movie recommender system, from top-level architecture, data gathering and preprocessing, vector database, recommendation module, to presentation module.

CHAPTER 6

IMPLEMENTATION OF LLM BASED CONTEXT-AWARE MOVIE RECOMMENDER SYSTEM

6.1 Introduction

This chapter provides an in-depth explanation of how LLM based context-aware movie recommender system was implemented. Focusing on the tools, technologies, methods, and algorithms used in each module identified and described in the previous design chapter. As presented in the Figure 6.1 the four main components of the system are data preprocessing layer, data vectorizing layer, recommendation generation layer, and presentation layer. This chapter will discuss the implementation process, the challenges encountered, and strategies adopted to overcome those obstacles.

6.2 Data Preprocessing

As the title of the project describes, when it comes to movie recommendation systems the MovieLens data is the go-to dataset as it is widely regarded as a benchmark dataset. GroupLens has different variants of the movielens datasets ranging from 100K, 1M, 10M, 20M, to 25M movie ratings. We had to choose the 20M ratings dataset as it is the first dataset that contains the movieId to imdbId and tmdbId mappings built in. This is required to proceed in our project as the other aspect of our topic is context-awareness of the movie. It is also helpful that the mapping information is present in the same source dataset. Although the movielens 20M dataset contains six files, we only utilize three. They are namely movies, ratings, and links. They are all csv files. We use a python script to retrieve and extract the movie lens 20m dataset.

The movies file has the columns movieId, title, and genres which is pipe separated. The title has the release year in parentheses, because of this we don't use this title. Also, we don't use the genres data here. The file contains 27,278 unique movieIds. The links file has the columns movieId, imdbId, and tmdbId. We join these two csv files after reading them as pandas data frames on the common column of movieId.

The ratings file contains the column userId, movieId, rating and timestamp denoting the time the rating was done. This file has 20,000,263 rows, hence the mentioned as the movielens 20M dataset. There are 138,493 unique userIds rating 26,744 unique movieIds on a scale of 5 stars with half-star increments. The ratings distribution is as follows, and it is followed by the user-rating distribution and movie-rating distribution.

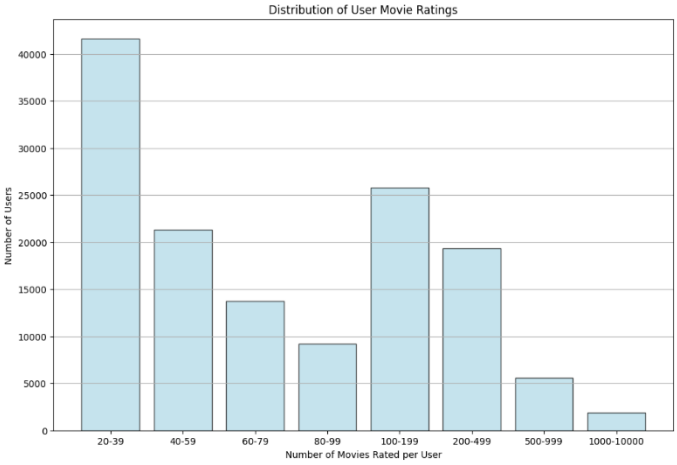

Figure 6. 1 Distribution of Movie Ratings.

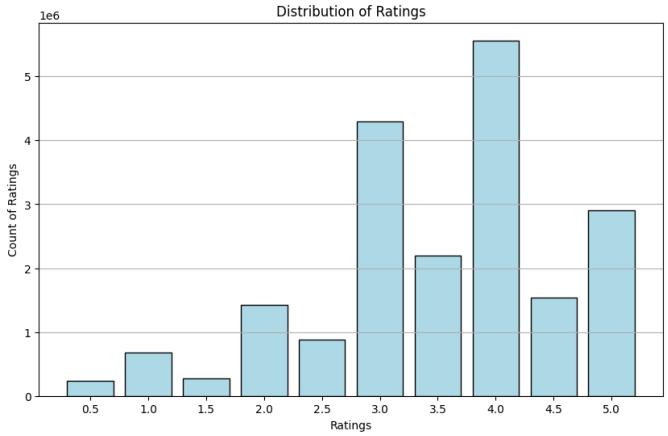

Figure 6. 2 Movie Rating Distribution in millions.

The next step of retrieving data is to get movie-context information from TMDb. To begin with this process first we need to sign up on the website https://www.themoviedb.org/signup. Then we need to describe the application and request the API key. There is a 50 requests per second upper limit in using the API. Figure 6.3 shows API usage for movie data context retrieval.

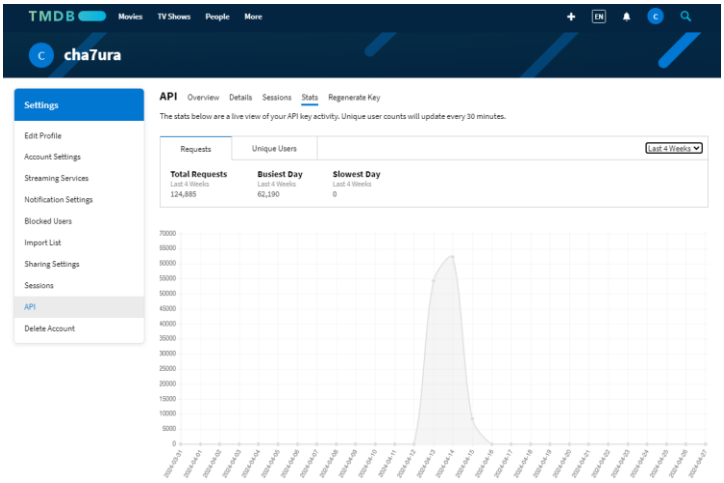

Figure 6. 3 TMDb API usage for movie-context information.

From the data available in TMDb we have only retrieved the following data using the imdbId as reference. Before we could start sending requests, we had to change the imdbId format to begin with "tt" and have 7 digits following it. After that we retrieved data for title (as movielens data had the released year in parentheses after the name in the title column), release date (this has the date,, not just the year), genres (which is more complete than movielens data), language (languages the original movie was released in), actors (the top 5 names from the cast key value pair), Director (from the crew key value pair), plot summary from overview, keywords and poster link for presentation module. We had to parallelize the process of data retrieval as we were requesting data for 26,774 movies. Among the data retrieved we found out some were related to TV series, and some didn't have data. So, we had this data omitted. Following is a sample dataset of the preprocessed data.

	ImdbId	Title	Genres	Plot	Tagline	SpokenLanguages	AverageRating	PosterPath	ExternalLinks	Keywords	Director	BasedOnNovelBy	Actors	MovieReleaseYear
0	tt0114709	Toy Story	[Animation, Adventure, Family, Comedy]	Led by Woody, Andy's toys live happily in his ...	Hang on for the comedy that goes to infinity a...	[English]	8.0	https://image.tmdb.org/t/p/w500/uXDfjJbdP4jW5...	https://www.imdb.com/title/tt0114709	[rescue, friendship, mission, martial arts,...			[Tom Hanks, Tim Allen, Don Rickles, Jim Var...	1995.0
1	tt0113497	Jumanji	[Adventure, Fantasy, Family]	When siblings Judy and Peter discover an encha...	Roll the dice and unleash the excitement!	[English, French]	7.2	https://image.tmdb.org/t/p/w500/bdHG5Mo83VPobe...	https://www.imdb.com/title/tt0113497	[giant insect, board game, disappearance, j...	Joe Johnston	Chris Van Allsburg	[Robin Williams, Kirsten Dunst, Bradley Pier...	1995.0
2	tt0113228	Grumpier Old Men	[Romance, Comedy]	A family wedding reignites the ancient feud be...	Still Yelling. Still Fighting. Still Ready for...	[English]	6.5	https://image.tmdb.org/t/p/w500/1FSXpj5e8l4KH6...	https://www.imdb.com/title/tt0113228	[fishing, sequel, old man, best friend, we...	Howard Deutch		[Walter Matthau, Jack Lemmon, Ann-Margret, ...	1995.0
3	tt0114885	Waiting to Exhale	[Comedy, Drama, Romance]	Cheated on, mistreated and stepped on, the wom...	Friends are the people who let you be yourself...	[English]	6.3	https://image.tmdb.org/t/p/w500/qJU6rfil5xLVb5...	https://www.imdb.com/title/tt0114885	[based on novel or book, interracial relation...	Forest Whitaker		[Whitney Houston, Angela Bassett, Loretta De...	1995.0

Figure 6. 4 Sample data after preprocessing.

6.3 Embedding model and Vector distance

OpenAI provides 3 different embedding models such as Ada-002, Text-embedding-3-small and Text-embedding-3-large which has 1536, 1536, 3072 dimensions respectively. Ada-002 is an older model whereas the other two are somewhat new. Therefore, we must choose which model is best for our dataset by experimenting on a sample dataset. Similarly, the Weaviate vector database provides 5 different measures to calculate vector distances, namely Cosine distance, Dot product, L2-Squared distance, Hamming distance, and Manhattan Distance metrics. Therefore, a sample dataset of 1000 users (20336 ratings) are used to calculate the metrics for Hit Rate, Recall, Precision, F1 score at K= 5 and 10. Table 6.1 shows the experimental values for Hit rate and F1 score. Among these 15 approaches using the Text-embedding-3-large embedding model with L2-Squared vector distances gives out the best overall values. Therefore, going forward this embedding model and distance calculation method is used.

Table 6. 1 - Sample data evaluation

Embedding model	Vector distance	HR@5		HR@10		F1@5		F1@10	
		Model1	Model2	Model1	Model2	Model1	Model2	Model1	Model2
Ada2	Cosine	0.6	0.63	0.75	0.77	0.08	0.08	0.11	0.11
Ada2	Dot	0.69	0.7	0.83	0.83	0.1	0.1	0.12	0.12
Ada2	Hamming	0.03	0.03	0.05	0.05	0	0	0	0
Ada2	L2-squared	0.67	0.69	0.81	0.82	0.09	0.09	0.12	0.12
Ada2	Manhattan	0.57	0.56	0.72	0.71	0.07	0.07	0.09	0.09
T-E-3-small	Cosine	0.74	0.75	0.86	0.88	0.09	0.09	0.11	0.12
T-E-3-small	Dot	0.75	0.77	0.88	0.89	0.09	0.09	0.12	0.12
T-E-3-small	Hamming	0.26	0.26	0.28	0.28	0.03	0.03	0.02	0.02
T-E-3-small	L2-squared	0.77	0.78	0.88	0.9	0.09	0.09	0.12	0.13
T-E-3-small	Manhattan	0.55	0.57	0.7	0.73	0.05	0.06	0.07	0.08
T-E-3-large	Cosine	0.78	0.8	0.87	0.89	0.13	0.14	0.17	0.18
T-E-3-large	Dot	0.89	0.89	0.95	0.95	0.17	0.17	0.21	0.22
T-E-3-large	Hamming	0.03	0.03	0.33	0.33	0	0	0.02	0.02
T-E-3-large	L2-squared	0.89	0.91	0.97	0.96	0.17	0.18	0.22	0.23
T-E-3-large	Manhattan	0.73	0.76	0.86	0.86	0.12	0.12	0.15	0.16

6.4Vectorizing Data

The vector database used in this project is weaviate. It is an opensource vector database which supports numerous ways of implementation. We have chosen the weaviate cloud service as it can be implemented free of charge. But the caveat of this is that the cluster expires automatically in 14 days. We created a collection named “Movie” using the WCS_CLUSTER_URL, WCS_API_KEY and OPENAI_API_KEY in the connection parameters. When creating the collection we configure the vectorizer as “text2vec_openai()” and generative capability as “openai()” as well. The vectorizer is used to generate the vector database corresponding to the movie data and the generative configuration is used for the explainability feature. The embedding task is a one-time run and creates a 3072-dimensional

vector for each movie. There are better embedding models if we investigate the huggingface website, but we have opted to use OpenAI due to the relative easiness in implementation.

```
client.collections.delete('Movie')
movie = client.collections.create(
    name = 'Movie',
    vectorizer_config=wvc.config.Configure.Vectorizer.text2vec_openai(),
    generative_config=wvc.config.Configure.Generative.openai(),
    properties=[
        wvc.config.Property(name='imdbId', data_type=wvc.config.DataType.TEXT, description='The IMDb id of the movie', vectorize_property_name=False, skip_vectorization=True),
        wvc.config.Property(name='title', data_type=wvc.config.DataType.TEXT, description='The name of the movie'),
        wvc.config.Property(name='runtime', data_type=wvc.config.DataType.INT, description='The runtime of the movie', vectorize_property_name=False, skip_vectorization=True),
        wvc.config.Property(name='posterLink', data_type=wvc.config.DataType.TEXT , description='The poster link of the movie', vectorize_property_name=False, skip_vectorization=True),
        wvc.config.Property(name='genres', data_type=wvc.config.DataType.TEXT_ARRAY , description='genres of the movie'),
        wvc.config.Property(name='language', data_type=wvc.config.DataType.TEXT , description='original language of the movie'),
        wvc.config.Property(name='actors', data_type=wvc.config.DataType.TEXT_ARRAY , description='actors of the movie'),
        wvc.config.Property(name='director', data_type=wvc.config.DataType.TEXT , description='director of the movie'),
        wvc.config.Property(name='keywords', data_type=wvc.config.DataType.TEXT_ARRAY , description='main keywords of the movie'),
        wvc.config.Property(name='year', data_type=wvc.config.DataType.INT , description='year in which movie was published'),
        wvc.config.Property(name='plot', data_type=wvc.config.DataType.TEXT , description='Plot of the movie'),

    ]
)

client.close()
```

Figure 6. 5 Weaviate Vector Database creation.

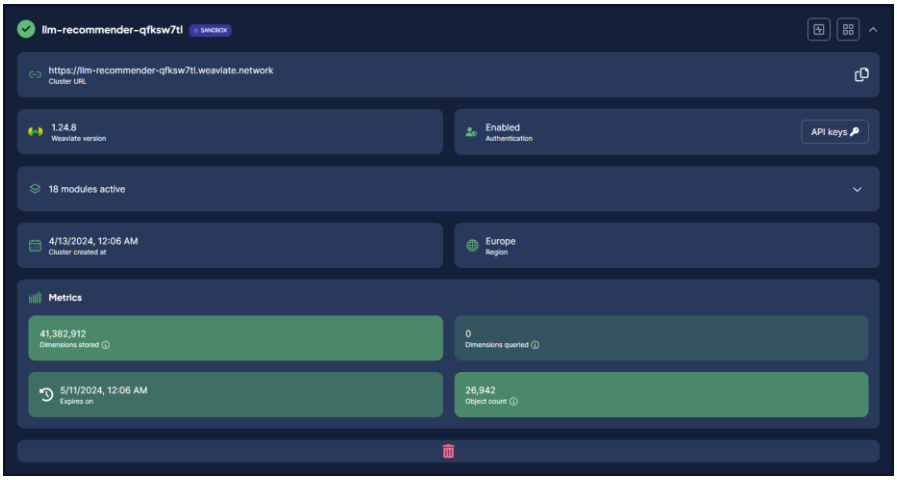

Figure 6. 6 Weaviate Dashboard

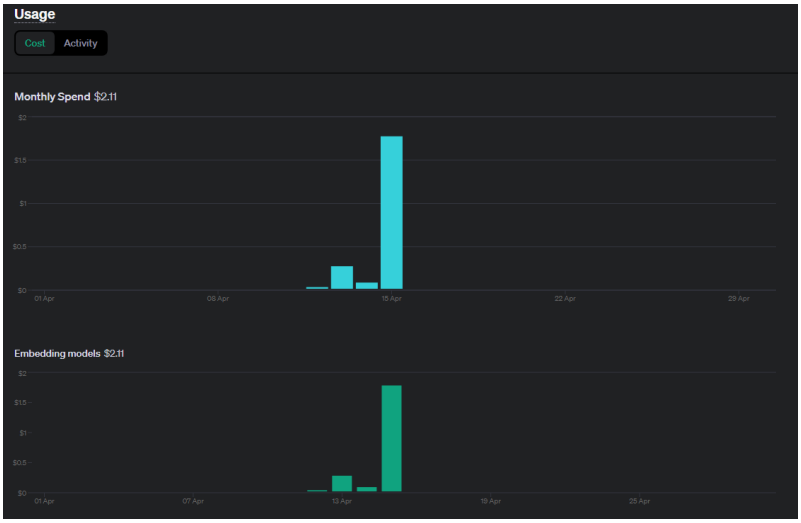

Figure 6. 7 OpenAI usage cost.

Figure 6. 8 OpenAI API usage

6.5 Recommendation Module

As mentioned in the vectorizing data step, we decided to make the imdbId the primary key of the dataset. Therefore, before generating the centroid vectors, we downloaded the imdbId and relevant vector generated by the weaviate vector database to the local development environment. After retrieving this data, we started on generating the two centroid vectors.

The first centroid vector was developed by using the ratings dataset filtered to only contain movies rated 3-starts and above by users. The change in data counts is shown below.

Then we had to convert the data to vectors as when read from csv the format had changed. Reading 16M rows of data and converting them to vectors was a resource intensive task. Therefore, this task was parallelized, and the calculated centroid vector was sent to weaviate to query the nearest 10 movie vectors to get data for the evaluation step and was saved in increments of 100 users and joined together to a single file later.

The second centroid vector development used the entire dataset of ratings data due to using the average weighted rating value multiplied by the vector. This dataset had the same issue of format change of vectors due to reading from a csv. This task was also parallelized to calculate the centroid vector, and then retrieve the nearest 10 movie vectors for evaluation. This step also involved saving files in increments of 100 users to generate the final dataset.

6.6 Presentation Module

The presentation module was developed using streamlit python-based data representation module. The poster link was used to display the images and was ordered in the vector distance ascending order. The live demo function for rating moves was developed so that the user-vector was calculated instantly when a rating is given, and the nearest vectors were retrieved from the vector database. The keyword search feature was just a search page where for each key word the closest movies were displayed, with the ability to explain the movie recommendation. Also, the final tab is a pandas data frame filter and ordered display.

The development of the model architecture involved integrating the Weaviate vector database with our backend systems. We used Python's Flask framework to develop a RESTful API, which allowed for efficient communication between the frontend and the vector database. This API handled requests from the user interface, queried the Weaviate database for relevant movie vectors, and then returned the results to the user.

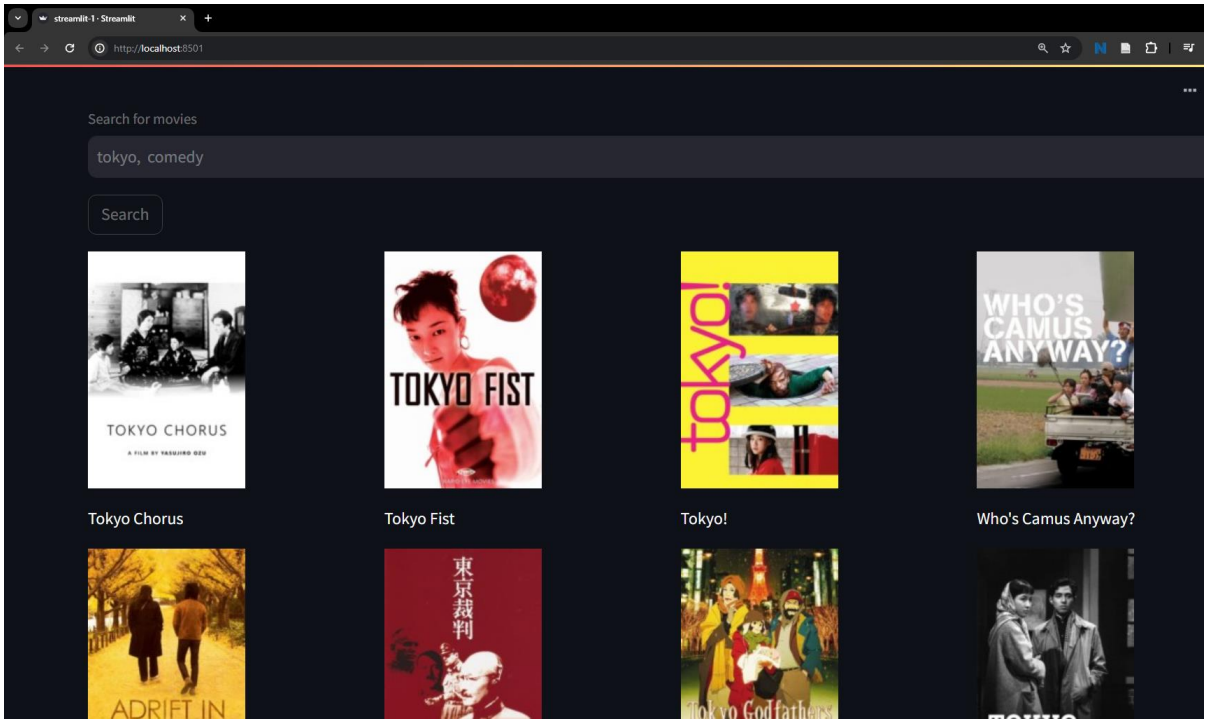

Figure 6. 9 Presentation module implementation

6.7 Summary

This chapter has elaborated in detail the implementation steps of our LLM based context-aware movie recommender system. The chapter focuses on the tools, technologies, methods, and algorithms used in data preprocessing layer, data vectorizing layer, recommendation generation layer, and presentation layer. This chapter also discusses the challenges encountered, and strategies adopted to overcome those obstacles. In conclusion we have achieved the third objective identified in the first chapter of this document.

CHAPTER 7

EVALUATION

7.1 Introduction

In the previous chapter, we discussed the implementation of our study. In this chapter we are going to elaborate on the experimental setup and evaluation of the developed LLM based context-aware movie recommender system. The evaluation is performed for the two recommendation models that were created. We discuss the novelty of our research and findings in this chapter.

7.2 Evaluation parameters

7.2.1 Recall@K

Recall at K is defined as the number of relevant movies recommended from top K over total number of relevant items. In the dataset we have used each user has at least 20 or more movie-ratings.

$$Recall\ at\ K = \frac{|R_K|}{|R|}$$

Where $|R_K|$ is the number of relevant items in top K recommendations and |R| is the total number of relevant items (which is always 20 or more in this case). We calculate Recall@5 and Recall@10 for both our recommendation models.

7.2.2 Precision@K

Precision at K is defined as the number of relevant movies in top K over K. We calculate Recall@5 and Recall@10 for both our recommendation models.

$$Precision\ at\ K = \frac{Number\ of\ Relevant\ Items\ in\ Top\ K}{K}$$

7.2.3 F1@K

The F1 score is a metric used to evaluate accuracy of the recommender system. Since we calculate both Recall and Precision at K we can easily calculate this. The approach we have taken to calculate the F1 score is to retrieve F1 score for each individual user and then produce the final score by finding the average.

$$F1@K = 2 \times \left(\frac{Precision@K \times Recall@K}{Precision@K + Recall@K}\right)$$

7.2.4 HR@K

HR@K or Hit Rate at K measures whether at least one relevant movie is included in the top K recommendations. This is a binary metric. HR@K for all the users is defined as follows.

$$HR@K = \frac{Number\ of\ users\ with\ at\ least\ one\ relevant\ item\ in\ top\ K}{Total\ number\ of users}$$

7.3 Model Evaluation

The model evaluation is done by comparing the two recommendation models against each other. The first model or Model 1 is based on user-ratings where we only consider relevant movies. This means only movies rated 3 stars or above by users are considered. The user-centroid vector is calculated by the average of relevant movie-vectors for a user. And each user has at least 20 of these types of ratings. Therefore, when it comes to relevant movies for the metric Recall and Hit Rate, we only consider movies with ratings 3 stars or above for user.

The second model or Model 2 is also based on user-ratings, but the user-centroid vector is calculated by the average vector of weighted vectors for all rated movies. To ease the comparison between the two models, we only compare data for common users in both models and as mentioned above the relevant movies are movies rated 3 stars or above by each user.

To make the two model comparisons realistic we use data for 126,826 common users to calculate the metrics mentioned above. For both models' values are calculated for K=5 and K=10. This means top 5 and top 10 movie recommendations have been generated for both models, for the number of common users mentioned above. The following discussion is based on Top K values.

7.3.1 Model Evaluation for Top 5

	Recall@5	Precision@5	F1@5
Model 1	0.0180	0.1701	0.0315
Model 2	0.0180	0.1691	0.0317
CF	0.1358	0.0271	0.0452

Table 7. 1 Recall, Precision and F1 score for models at top 5.

Comparing the performance metrics for Model 1 and Model 2, we find that overall scores for both models are low compared to collaborative filtering (CF) approach. Models 1 and 2 have a recall of 1.08% compared to 13.58% of CF, meaning it is way better at identifying relevant movies. When it comes to precision, Model 1 shows slightly higher precision of 17.01% compared to 16.91% of Model 2 compared to 2.71% for CF, this is a low value. However, the F1 score also shows that the CF approach has better accuracy of 4.52% compared to 3.15% and 3.17% for Model 1 and Model 2 respectively. Given the very similar accuracy metrics, between Model 1 and Model 2, we haven't achieved an improvement compared to traditional recommendation methods.

	HR@5
Model 1	0.4897
Model 2	0.4872
CF	0.1358

Table 7. 2 Hit Rate for models at top 5.

The Hit Rate comparison at 5 shows that Model 1 includes relevant movies in the top 5 recommendations about 48.97% of the time compared to 48.72% for Model 2. But here the traditional approach gets 13.58% which is comparatively lower. This metric provides a direct measure of how often a user might find a relevant recommendation quickly, and Model 1 marginally better at it.

Given Hit Rate along with Recall, Precision and F1 scores, Model 1, Model 2 and collaborative filtering approach shows mixed results.

7.3.2 Model Evaluation for Top 10

	Recall@10	Precision@10	F1@10
Model 1	0.0281	0.1403	0.0445
Model 2	0.0282	0.1397	0.0446
CF	0.2476	0.0247	0.0450

Table 7. 3 Recall, Precision and F1 score for models at top 10.

When comparing the performance metrics for Model 1 and Model 2 for top 10 recommendations, we find similar scenario as top 5 recommendations. Models 1 and 2 have similar recall of 2.81% and 2.82% while CF approach has 24.74%. All values are higher than the values for recall at top 5, but traditional approach having significantly improve. When it comes to precision, Model 1 shows slightly higher precision of 14.03% compared to 13.97% for Model 2, while CF approach has 2.47%. Here all precision values have lowered compared to the top 5. The F1 score also shows that the traditional approach is slightly better compared to Model 1 and Model 2. Here also the F1 scores have higher values than of the top 5 F1 scores. Overall, given the three metrics of recall, precision and F1 score, the models show mixed results.

	HR@10
Model 1	0.6266
Model 2	0.6235
Collaborative filtering	0.2476

Table 7. 4 Hit Rate for models at top 10.

The Hit Rate comparison at 10 shows significant improvements compared to HR@5. Model 1 has a percentage of 62.66% compared to 62.35% of Model 2 when recommending relevant movies, compared to 24.76% for collaborative filtering approach. Here also Model 1 is marginally better at it, compared to Model 2. When considering these recommendations are for 126,826 users it is a considerable difference.

Based on the provided metrics, collaborative filtering approach stands out with significantly higher recall values at both top 5 and top 10 levels compared to Model 1 and Model 2, indicating that CF retrieves more relevant items within the top 5 and top 10 predictions. However, CF's precision is much lower than that of Model 1 and Model 2, suggesting a lower proportion of relevant items among those retrieved. This results in lower F1 scores for CF, which balances precision and recall. When considering hit rates (HR), Model 1 and Model 2

perform nearly identically, with Model 1 showing a slight edge. Given these observations, if maximizing recall is the priority, CF would be the preferred model. On the other hand, if a balance between precision and recall is crucial, Model 1 emerges as the slightly better option. Model 2 is also competitive but marginally behind Model 1 in hit rate metrics.

7.4 Summary

This chapter focuses on the evaluation of the two different models, Model 1, and Model 2, in the movie recommendation context. Both models use Recall, Precision, F1 score and Hit Rate and depths of top 5 and top 10 recommendations respectively. In comparing the metrics in both tiers, we find that Model 1 has better performance across the board even though it is marginal, compared to Model 2. Since movie recommendations are given in a larger list than 5 or 10 this will perform better in a real-world scenario.

In conclusion, we have marginally achieved our final objective of evaluating the proposed models and finding out Model 1 performs better overall in the top 5 and top 10 recommendation scenarios.

CHAPTER 8

CONCLUSION AND FUTURE WORK

8.1 Introduction

This chapter focuses on the conclusion of the developed system, which integrates an LLM for building a context-aware movie recommender system. It highlights the conclusion of the findings through evaluation and methods to improve the model under future work.

8.2 Conclusion

In conclusion, the integration of a vector database to a recommendation system for movies has provided a comprehensive solution for recommending movies to users based on their previous movie ratings. The user vector generation using the centroid-vector of rated movies by a user shows us that this is a viable method for fast recommendations. Compared to the two models of user-vectors we can conclude this method has shown great promise in using LLMs as a part of the recommendations. Using a vector database and L2-squared distance between user-vector and movie-vectors has shown that ranking recommendations can be done easily using the vector database. Compared to the two-tower approach of traditional recommendation systems where only a subsection of movies is selected for ranking, the vector database uses all the movie-data (vectors) when considering a recommendation. This is not a slow process as the vector indexing is done at data writing time to the database. Having a vector database also helps with the cold-start problem that is inherent to recommender systems as the user can also semantically search a movie and rate it, which in turn creates a user-vector that can start recommending movies to the user. What is important in this recommendation system is that context-aware movie data is present in the input data. The embedding model creates the embedding vector dependent on this data, and it has been key in generating recommendations in this development.

The developed solution for approaching recommendations with LLMs provide an excellent example of the capabilities of LLM and their underlying technologies.

8.3 Limitations and Further work

Regarding future work, there are numerous opportunities for enhancing the user-vector model. One improvement that can be done is using unsupervised learning to cluster the movie rating vectors and create multiple user-vectors that identify different likings of the user and showing recommendations in multiple levels depending on the number of clustered movies in each cluster. This will show a variety of different recommendations that are dependent on user taste and ratings. Furthermore, one limitation is that award related data is not present in TMDb dataset, which we think would improve upon the vectorization of movie-data.

8.4 Summary

In this chapter, we have presented the overall conclusion of our study on the LLM based context-aware movie recommender system. Our study shows that LLMs and their underlying technology have great potential in generating movie recommendations. We have provided a comprehensive summary of our findings and discussed the future work that can be done to improve our findings. We believe that continued research and development in this area can lead to applying recommendations to a wider domain area, benefiting various stakeholders. We hope our study's contributions can pave the way for further research in these combinatory fields to ultimately improve the quality of life for individuals.

REFERENCES

[1] L. Wu *et al.*, "A Survey on Large Language Models for Recommendation." arXiv, Aug. 18, 2023. Accessed: Aug. 21, 2023. [Online]. Available: http://arxiv.org/abs/2305.19860

[2] P. Castells and D. Jannach, "Recommender Systems: A Primer." arXiv, Feb. 06, 2023. Accessed: Apr. 27, 2024. [Online]. Available: http://arxiv.org/abs/2302.02579

[3] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding." arXiv, May 24, 2019. Accessed: Apr. 26, 2024. [Online]. Available: http://arxiv.org/abs/1810.04805

[4] OpenAI, "GPT-4 Technical Report." arXiv, Mar. 27, 2023. Accessed: Sep. 03, 2023. [Online]. Available: http://arxiv.org/abs/2303.08774

[5] H. Touvron *et al.*, "Llama 2: Open Foundation and Fine-Tuned Chat Models." arXiv, Jul. 19, 2023. Accessed: Sep. 03, 2023. [Online]. Available: http://arxiv.org/abs/2307.09288

[6] A. Vaswani *et al.*, "Attention Is All You Need." arXiv, Aug. 01, 2023. Accessed: Sep. 03, 2023. [Online]. Available: http://arxiv.org/abs/1706.03762

[7] Q. Zhang, J. Lu, and Y. Jin, "Artificial intelligence in recommender systems," *Complex Intell. Syst.*, vol. 7, no. 1, pp. 439–457, Feb. 2021, doi: 10.1007/s40747-020-00212-w.

[8] D. Goldberg, D. Nichols, B. M. Oki, and D. Terry, "Using collaborative filtering to weave an information tapestry," *Commun. ACM*, vol. 35, no. 12, pp. 61–70, Dec. 1992, doi: 10.1145/138859.138867.

[9] C. C. Aggarwal, *Recommender Systems*. Cham: Springer International Publishing, 2016. doi: 10.1007/978-3-319-29659-3.

[10] X. Su and T. M. Khoshgoftaar, "A Survey of Collaborative Filtering Techniques," *Adv. Artif. Intell.*, vol. 2009, p. e421425, Oct. 2009, doi: 10.1155/2009/421425.

[11] B. Sarwar, G. Karypis, J. Konstan, and J. Reidl, "Item-based collaborative filtering recommendation algorithms," in *Proceedings of the tenth international conference on World Wide Web - WWW '01*, Hong Kong, Hong Kong: ACM Press, 2001, pp. 285–295. doi: 10.1145/371920.372071.

[12] D. Pavlov and D. Pennock, "A Maximum Entropy Approach to Collaborative Filtering in Dynamic, Sparse, High-Dimensional Domains.," presented at the Advances in Neural Information Processing Systems, Jan. 2002, pp. 1441–1448.

[13] S. J. Russell, P. Norvig, and E. Davis, *Artificial intelligence: a modern approach*, 3rd ed. in Prentice Hall series in artificial intelligence. Upper Saddle River: Prentice Hall, 2010.

[14] R. Salakhutdinov, A. Mnih, and G. Hinton, "Restricted Boltzmann machines for collaborative filtering," in *Proceedings of the 24th international conference on Machine learning - ICML '07*, Corvalis, Oregon: ACM Press, 2007, pp. 791–798. doi: 10.1145/1273496.1273596.

[15] P. Lewis *et al.*, "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks." arXiv, Apr. 12, 2021. Accessed: Apr. 26, 2024. [Online]. Available: http://arxiv.org/abs/2005.11401

[16] Z. Qiu, X. Wu, J. Gao, and W. Fan, "U-BERT: Pre-training User Representations for Improved Recommendation," *Proc. AAAI Conf. Artif. Intell.*, vol. 35, no. 5, Art. no. 5, May 2021, doi: 10.1609/aaai.v35i5.16557.

[17] C. Wu, F. Wu, Y. Yu, T. Qi, Y. Huang, and X. Xie, "UserBERT: Contrastive User Model Pre-training." arXiv, Sep. 02, 2021. doi: 10.48550/arXiv.2109.01274.

[18] Y. Yang, Y. Qiao, J. Shao, M. Anand, X. Yan, and T. Yang, "Composite Re-Ranking for Efficient Document Search with BERT," in *Proceedings of the Fifteenth ACM International Conference on Web Search and Data Mining*, Feb. 2022, pp. 1234–1244. doi: 10.1145/3488560.3498495.

[19] Q. Liu, N. Chen, T. Sakai, and X.-M. Wu, "ONCE: Boosting Content-based Recommendation with Both Open- and Closed-source Large Language Models." arXiv, Aug. 31, 2023. Accessed: Sep. 02, 2023. [Online]. Available: http://arxiv.org/abs/2305.06566

[20] Z. Qin *et al.*, "Large Language Models are Effective Text Rankers with Pairwise Ranking Prompting." arXiv, Mar. 28, 2024. Accessed: Apr. 27, 2024. [Online]. Available: http://arxiv.org/abs/2306.17563

[21] S. Dai *et al.*, "Uncovering ChatGPT's Capabilities in Recommender Systems," in *Proceedings of the 17th ACM Conference on Recommender Systems*, Sep. 2023, pp. 1126–1132. doi: 10.1145/3604915.3610646.

[22] Y. Xi *et al.*, "Towards Open-World Recommendation with Knowledge Augmentation from Large Language Models." arXiv, Dec. 04, 2023. Accessed: Apr. 27, 2024. [Online]. Available: http://arxiv.org/abs/2306.10933

[23] K. Bao, J. Zhang, Y. Zhang, W. Wang, F. Feng, and X. He, "TALLRec: An Effective and Efficient Tuning Framework to Align Large Language Model with Recommendation," in *Proceedings of the 17th ACM Conference on Recommender Systems*, Sep. 2023, pp. 1007–1014. doi: 10.1145/3604915.3608857.

[24] L. Li, Y. Zhang, and L. Chen, "Personalized Prompt Learning for Explainable Recommendation." arXiv, Jan. 13, 2023. Accessed: Apr. 27, 2024. [Online]. Available: http://arxiv.org/abs/2202.07371

[25] S. Geng, S. Liu, Z. Fu, Y. Ge, and Y. Zhang, "Recommendation as Language Processing (RLP): A Unified Pretrain, Personalized Prompt & Predict Paradigm (P5)." arXiv, Jan. 02, 2023. Accessed: Aug. 23, 2023. [Online]. Available: http://arxiv.org/abs/2203.13366

[26] L. Wu, Z. Qiu, Z. Zheng, H. Zhu, and E. Chen, "Exploring Large Language Model for Graph Data Understanding in Online Job Recommendations." arXiv, Dec. 23, 2023. Accessed: Apr. 27, 2024. [Online]. Available: http://arxiv.org/abs/2307.05722

APPENDIX A

DATA RETRIEVAL AND PREPROCESSING SOURCE CODE

MovieLens Data Retrieval

```python
import pandas as pd
import requests
import zipfile
import io

def download_and_extract_movielens(url):
    response = requests.get(url)
    with zipfile.ZipFile(io.BytesIO(response.content)) as zip_file:
        zip_file.extractall(path='data')

def load_movielens_data(path):

    titles = pd.read_csv(f'{path}/movies.csv', delimiter=',', header=0)
    links = pd.read_csv(f'{path}/links.csv', delimiter=',', header=0)
    return titles, links

def format_imdb_id(num_id):
    return f"tt{num_id:07d}"

if __name__ == "__main__":
    url = 'http://files.grouplens.org/datasets/movielens/ml-20m.zip'
    download_and_extract_movielens(url)

    titles, links = load_movielens_data(path = 'data/ml-20m')

    titles_with_imdb_id = pd.merge(titles, links, on='movieId')
    titles_with_imdb_id['imdbId'] = titles_with_imdb_id['imdbId'].apply(format_imdb_id)

    titles_with_imdb_id.to_csv('data/titles_with_imdb_id.csv', index = False)

```

TMDb Data Retrieval

```python
import requests
import pandas as pd
from tqdm import tqdm
from concurrent.futures import ThreadPoolExecutor, as_completed
import os
import numpy as np

tmdb_api_key=os.getenv("TMDB_API_KEY")

def fetch_movie_data(imdb_id, api_key):
    base_url = "https://api.themoviedb.org/3"
    find_url = f"{base_url}/find/{imdb_id}?api_key={api_key}&external_source=imdb_id"
    response = requests.get(find_url)
    movie_data = response.json()

    if 'movie_results' in movie_data and movie_data['movie_results']:
        movie_id = movie_data['movie_results'][0]['id']
        details_url = f"{base_url}/movie/{movie_id}?api_key={api_key}&append_to_response=credits,keywords,videos,external_ids"
        movie_response = requests.get(details_url).json()

        return {
            'IMDbID': imdb_id,
            'Title': movie_response.get('title'),
            'OriginalTitle': movie_response.get('original_title'),
            'Genres': ', '.join(genre['name'] for genre in movie_response.get('genres', [])),
            'ReleaseDate': movie_response.get('release_date'),
            'Runtime': movie_response.get('runtime'),
            'Overview': movie_response.get('overview'),
            'Tagline': movie_response.get('tagline'),
            'Cast': {cast['name']: cast['character'] for cast in movie_response.get('credits', {}).get('cast', [])},
            'Crew': {crew['name']: crew['job'] for crew in movie_response.get('credits', {}).get('crew', [])},
            'ProductionCompanies': ', '.join(company['name'] for company in movie_ (variable) movie_response: Any s', [])),
            'ProductionCountries': ', '.join(country['name'] for country in movie_ s', [])),
            'SpokenLanguages': ', '.join(language['english_name'] for language in movie_response.get('spoken_languages', [])),
            'Budget': movie_response.get('budget'),
            'Revenue': movie_response.get('revenue'),
            'Popularity': movie_response.get('popularity'),
            'VoteAverage': movie_response.get('vote_average'),
            'VoteCount': movie_response.get('vote_count'),
            'PosterPath': f"https://image.tmdb.org/t/p/w500{movie_response.get('poster_path')}" if movie_response.get('poster_path') else None,
            'BackdropPath': f"https://image.tmdb.org/t/p/w500{movie_response.get('backdrop_path')}" if movie_response.get('backdrop_path') else None,
            'Status': movie_response.get('status'),
            'Adult': movie_response.get('adult'),
            'ExternalLinks': f"https://www.imdb.com/title/{imdb_id}",
            'Videos': ', '.join(video['name'] for video in movie_response.get('videos', {}).get('results', [])),
            'Keywords': ', '.join(keyword['name'] for keyword in movie_response.get('keywords', {}).get('keywords', []))
        }
    else:
        return None

```

```python
def update_movie_data(df, api_key, save_path):
    num_workers = 10  # Number of threads
    batch_size = 1000  # Adjust batch size to a reasonable number for your use case

    with ThreadPoolExecutor(max_workers=num_workers) as executor:
        futures = []
        results = []

        for start in tqdm(range(0, len(df), batch_size), desc='Batches'):
            end = start + batch_size
            # Submit a batch of jobs to the executor
            futures.extend([executor.submit(fetch_movie_data, imdb_id, api_key) for imdb_id in df['imdbId'][start:end]])

        # Retrieve results as they complete
        for future in tqdm(as_completed(futures), total=len(futures), desc='Downloading'):
            result = future.result()
            if result:
                results.append(result)

        # Convert results to DataFrame
        result_df = pd.DataFrame(results)
        result_df.to_csv(save_path, index=False)

    return result_df

if __name__ == "__main__":
    titles_with_imdb_id = pd.read_csv('data/titles_with_imdb_id.csv')
    num_splits = 15
    df_split = np.array_split(titles_with_imdb_id, num_splits)

    for i in range(num_splits):
        df_input = df_split[i]
        save_path = f'data/tmdb/tmdb_data_{i}.csv'
        update_movie_data(df_input, tmdb_api_key, save_path)

    folder_path = 'data/tmdb'
    csv_files = [f for f in os.listdir(folder_path) if f.endswith('.csv')]
    dataframes = []

    for file in csv_files:
        file_path = os.path.join(folder_path, file)
        df = pd.read_csv(file_path)
        print(df.shape)
        dataframes.append(df)

    concatenated_df = pd.concat(dataframes, ignore_index=True)

    concatenated_df.to_csv('data/tmdb_data_all.csv', index = False)
```

Data Preprocessing for upload

```python
import pandas as pd

titles = pd.read_csv('data/titles_with_imdb_id.csv')
movie_data = pd.read_csv('data/tmdb_data_all.csv')

df = movie_data.copy()

df['genres'] = df['Genres'].apply(lambda x: x.split('|') if pd.notna(x) else [])
df['actors'] = df['Actors'].apply(lambda x: x.split('|') if pd.notna(x) else [])
df['keywords'] = df['Keywords'].apply(lambda x: x.split('|') if pd.notna(x) else [])
df['ReleaseDate'] = pd.to_datetime(df['ReleaseDate'])
df['Year']= df['ReleaseDate'].dt.year
df.rename({'formatted_imdbId':'imdbId', 'Title':'title', 'Duration':'runtime', 'PosterLink':'posterLink', 'Language':'language', 'Year':'year', 'PlotSummary':'plot', 'Director':'director'},
          axis=1, inplace=True)
df.drop(columns=['ReleaseDate', 'Keywords', 'Actors', 'Genres'], inplace=True)
df = df[['imdbId', 'title', 'runtime', 'posterLink', 'genres', 'language', 'actors', 'director', 'keywords', 'year', 'plot']]
df.to_csv('data/input_data_for_weaviate.csv', index = False)
```

APPENDIX B

VECTOR DATABASE RELATED SOURCE CODE

Vector database creation

```python
import pandas as pd
import weaviate
import weaviate.classes as wvc
import os
import requests
import json
import numpy as np

client = weaviate.connect_to_wcs(
    cluster_url=os.getenv("WCS_CLUSTER_URL"),
    auth_credentials=weaviate.auth.AuthApiKey(os.getenv("WCS_API_KEY")),
    headers={
        "X-OpenAI-Api-Key": os.environ["OPENAI_API_KEY"]  # Replace with your inference API key
    },
    additional_config=wvc.init.AdditionalConfig(timeout=wvc.init.Timeout(init=10))
)

if __name__ == "__main__":
    client.connect()
    client.collections.delete('Movie')
    movie = client.collections.create(
        name = 'Movie',
        vectorizer_config=wvc.config.Configure.Vectorizer.text2vec_openai(),
        generative_config=wvc.config.Configure.Generative.openai(),
        properties=[
            wvc.config.Property(name='imdbId', data_type=wvc.config.DataType.TEXT, description='The IMDb id of the movie', vectorize_property_name=False, skip_vectorization=True),
            wvc.config.Property(name='title', data_type=wvc.config.DataType.TEXT, description='The name of the movie'),
            wvc.config.Property(name='runtime', data_type=wvc.config.DataType.INT, description='The runtime of the movie', vectorize_property_name=False, skip_vectorization=True),
            wvc.config.Property(name='posterLink', data_type=wvc.config.DataType.TEXT , description='The poster link of the movie', vectorize_property_name=False, skip_vectorization=True),
            wvc.config.Property(name='genres', data_type=wvc.config.DataType.TEXT_ARRAY , description='genres of the movie'),
            wvc.config.Property(name='language', data_type=wvc.config.DataType.TEXT , description='original language of the movie'),
            wvc.config.Property(name='actors', data_type=wvc.config.DataType.TEXT_ARRAY , description='actors of the movie'),
            wvc.config.Property(name='director', data_type=wvc.config.DataType.TEXT , description='director of the movie'),
            wvc.config.Property(name='keywords', data_type=wvc.config.DataType.TEXT_ARRAY , description='main keywords of the movie'),
            wvc.config.Property(name='year', data_type=wvc.config.DataType.INT , description='year in which movie was published'),
            wvc.config.Property(name='plot', data_type=wvc.config.DataType.TEXT , description='Plot of the movie'),
        ]
    )

    client.close()
```

Data upsert to vector database

```python
import pandas as pd
import weaviate
import weaviate.classes as wvc
import os
import requests
import json
import numpy as np

client = weaviate.connect_to_wcs(
    cluster_url=os.getenv("WCS_CLUSTER_URL"),
    auth_credentials=weaviate.auth.AuthApiKey(os.getenv("WCS_API_KEY")),
    headers={
        "X-OpenAI-Api-Key": os.environ["OPENAI_API_KEY"]  # Replace with your inference API key
    },
    additional_config=wvc.init.AdditionalConfig(timeout=wvc.init.Timeout(init=10))
)

if __name__ == "__main__":
    client.connect()
    df = pd.read_csv('data/input_data_for_weaviate.csv')

    num_splits = 26
    df_split = np.array_split(df, num_splits)

    for i in range(num_splits):
        print(i)
        # print('--------')
        df_input = df_split[i]
        data_rows = df_input.to_dict('records')
        client.connect()
        collection = client.collections.get("Movie")

        with collection.batch.fixed_size(100) as batch:
            for i, data_row in enumerate(data_rows):
                batch.add_object(
                    properties=data_row,
                )
        client.close()
    client.close()
```

UUID retrieval

```python
import pandas as pd
import weaviate
import weaviate.classes as wvc
import os
import requests
import json
import numpy as np

client = weaviate.connect_to_wcs(
    cluster_url=os.getenv("WCS_CLUSTER_URL"),
    auth_credentials=weaviate.auth.AuthApiKey(os.getenv("WCS_API_KEY")),
    headers={
        "X-OpenAI-Api-Key": os.environ["OPENAI_API_KEY"]  # Replace with your inference API key
    },
    additional_config=wvc.init.AdditionalConfig(timeout=wvc.init.Timeout(init=10))
)

if __name__ == "__main__":
    client.connect()
    movie = client.collections.get("Movie")
    uuid_data = []

    for item in movie.iterator(include_vector=True):
        item_data = {
            'uuid': item.uuid,
            'imdbId': item.properties.get('imdbId'),
            'vector': item.vector.get('default')
        }

        uuid_data.append(item_data)
    uuid_df = pd.DataFrame(uuid_data)
    print(uuid_df.shape)
    uuid_df.to_csv('data/uuid_data.csv', index = False)
```