Research Project I

CSCI 9301 Final Report

EnReLLM: Enhanced Recommendation Systems through Integration of Large Language Models

Name	Krishna Modi
B00 Number	B00931943
Term	Winter 2024
Supervisor	Dr. Ga Wu

Table of Contents

Executive Summary

This project explores the integration of Large Language Models (LLMs) into recommender systems, aiming to enhance recommendation accuracy and contextual understanding. Diverging from conventional approaches, we emphasize leveraging pre-trained LLMs without fine-tuning, focusing on augmenting their capabilities through innovative prompting strategies. The primary motivation is to infuse common sense and global knowledge into recommendations for improved outcomes.

We conducted experimentation with prompting techniques, including Instructions + Input, Instructions + Question, Questions + Example, and Role Playing, to understand their impact on recommendation quality. Our system architecture integrates the Movie Lens database, Surprise library for recommendation algorithms, and GPT-3.5 Turbo for contextual analysis, ensuring a user-centric approach.

System evaluation focuses on comparing ratings and rankings between the Recommender System Library (RS) and LLM, assessing rating discrepancies and recommendation diversity. This research underscores the potential of integrating LLMs with recommender libraries, paving the way for enhanced recommendation scenarios and personalized user experiences.

Motivation and Problem Statement

Leveraging Large Language Models (LLMs) in recommender systems provides a promising opportunity to enhance recommendation scenarios. The project diverges from traditional recommender system research by shifting away from extensive model architecture searches and in-house training. Although conventional methods yield satisfactory outcomes, there is room for improvement in incorporating common sense and global knowledge, which could lead to enhanced results. This highlights the significance of innovative solutions that utilize LLMs to define items and establish meaningful user relationships.

Our research follows the current trend of using generic, pre-trained, large language models for recommendation tasks without fine-tuning. We specifically investigate augmenting Language Models (LLMs) with well-defined prompts, focusing on long-context injection and formatting for effective context incorporation. The challenges include developing generic LLM-friendly context formats, leveraging pre-trained knowledge for smarter inference, and generalizing prompt-driven recommender systems for tasks with minimal or zero-shot prompting.

By leveraging LLMs' advanced capabilities, we have the potential to revolutionize recommendation systems. These systems are important in different domains, including e-commerce platforms and content streaming services, where personalized recommendations can significantly affect user experience, engagement, and satisfaction. Improving the accuracy, relevance, and contextual understanding of recommendations is a critical goal that directly influences the success and competitiveness of such platforms.

Our research addresses this challenge using LLMs to analyze and define items in the recommendation ecosystem. We aim to create a system that can intelligently distinguish item characteristics, user preferences, and their relationship by leveraging the vast knowledge of the LLMs. This approach improves recommendation accuracy and relevance and generates personalized and context-aware suggestions based on individual users' needs and preferences.

Approaches and Design Choices

During our system development journey, we experimented with different prompting techniques to solve challenges inherent in recommendation scenarios. Our goal was to find a functional solution to understand in detail the prompting methodologies to determine their strengths, weaknesses, and overall impact.

At first, we tried the Instructions + Input, Instructions + Question, and Questions + Example approaches [1]. These techniques provided the foundation for the model to generate contextually relevant responses. We evaluated each approach's ability to develop meaningful insights and encourage better recommendation outcomes.

Another approach we tried was role-playing prompts [1], in which we simulated various user scenarios and interactions to assess the model's ability to understand complex user intents and provide tailored recommendations. This immersive approach improved the model's reasoning abilities and helped it adapt to different user contexts.

Including the movie lens database improved our recommendation framework. With the help of a good database and the vast knowledge of the LLMs, we achieved improvements in enhancing recommendation accuracy and diversity. This allowed our system to retrieve relevant movie information and generate personalized recommendations resonating with users' preferences.

During our development process, we also tried experimenting with methods, like Advanced Chain of Thought and Tree of Thought [1]. Though these approaches are promising in theory, they were complicated for our particular use case setting. This served as teaching moments that helped us understand the complex relationship between prompting techniques and the effectiveness of the recommendations.

Our initial attempt to use the LLM's capabilities to extract top-N recommendations directly from the database did not work due to the LLM's token limits and contextual awareness. We decided to use Python Recommender Libraries in response, which gave us a more reliable and customized solution for our recommendation needs.

Finally, we settled on a combination of Instructions + Input, Instructions + Question, Questions + Example and Role Playing approaches [1]. These methods formed the backbone of our system's ability to generate relevant responses, allowing us to understand user preferences and deliver tailored recommendations. We integrated the movie lens database and LLM knowledge to enhance recommendation accuracy and diversity. This final approach successfully addressed the challenges posed by traditional methods and positioned our system for optimal performance in recommendation scenarios.

Tools and Technologies

In developing our movie recommendation system, we carefully selected tools and technologies aligned with our project's objectives and requirements. Our choice of SQLite for the Movie Lens database was because of its performance in data analysis. It is easy to install and use and allows raw data to be imported from CSV files which can be used to do complex analysis using programming languages like Python [2]. This allowed us to efficiently store and retrieve movie information for generating personalized recommendations.

Python is the programming language we chose for implementing our system. It's popular in Artificial Intelligence and Machine Learning [3]. It includes an extensive library ecosystem offering tools for various tasks allowing one to focus on the actual problem [3]. Python is also known for its scalability and readability making it the ideal choice for our use case as it facilitates complex operations on huge datasets [3].

For recommendations, we used the Surprise, a Python Scikit, known for its work in recommender systems dealing with rating data. It has been trained on the movie lens database, making it effective using our Movie Lens database[4]. Further, it provides built-in algorithms for prediction like matrix factorization-based algorithms that interest us [4].

'Movie Lens Small' database describes 5-star rating and free-text tagging activity from Movie Lens, a movie recommendation service [5]. It contains 100836 ratings and 3683 tag applications across 9742 movies [5]. These data were created by 610 users selected at random for inclusion. All selected users had rated at least 20 movies [5]. We are using this dataset to develop the system and see possibilities to report any research results if the full dataset is used.

To enhance the system and enable user context generation, we integrated GPT-3.5 Turbo [6] as our chat model. GPT-3.5 Turbo models can understand and generate natural language or code and have been optimized for chat [6]. We are trying to use it to understand the natural language and give more context to the user's preferences to improve the system's performance.

These tools were integrated carefully to provide smooth interaction and functionality within our system architecture. Each component uniquely contributes to our system's capabilities to create an effective and user-centric movie recommendation system.

System Architecture Overview

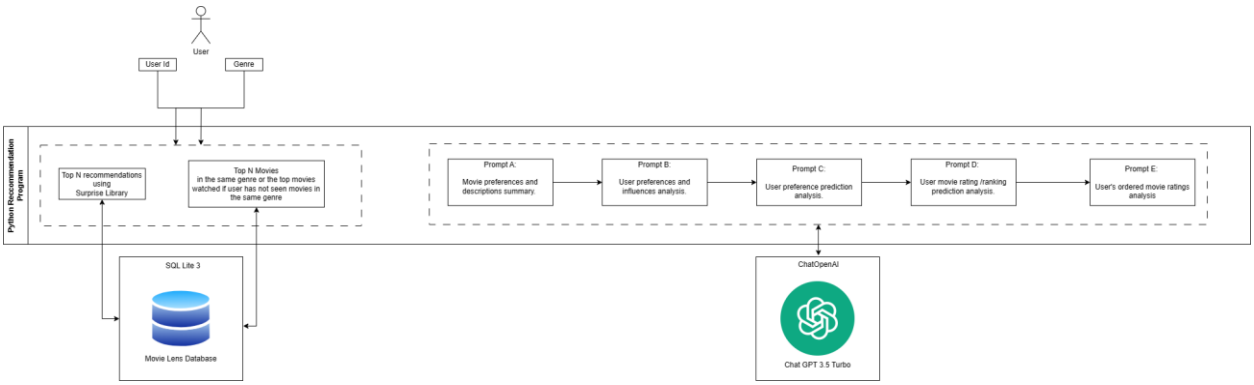

Figure 1. The System Architecture for the Python Recommendation Program [7]

The movie recommendation system works as an ecosystem (See Figure 1.), using various tools and technologies to deliver personalized movie recommendations to users. The process begins with movie-related data from the Movie Lens Database, including movie titles, genres, ratings, and user interactions, stored within a structured SQLite database. This data serves as the central repository for the recommendation engine.

When a user inputs their desired movie genres, the system processes this input and filters the movie data accordingly, focusing on movies that align with user preferences. This data processing step ensures that the recommendation engine generates relevant recommendations tailored to the user's tastes.

Next is the integration of the Surprise library. This library uses algorithms to analyze user-item interactions, and movie attributes such as genres and ratings to generate top-N movie recommendations based on user input. The system also employs a Singular Value Decomposition (SVD) algorithm within the recommender system to train the recommendation model, allowing it to learn from user ratings and predict preferences.

Once the recommender system generates top-N movie recommendations, the system uses an LLM (ChatGPT-3.5 Turbo used here) to analyze these recommendations for the user preferences. This multi-step prompting includes relating the user’s movie preferences with recommended movies. Based on this analysis, the system assigns new ratings to the recommended movies. It also ranks the recommended movies based on the newly assigned ratings and user preferences, ensuring that the most relevant movies are presented to the user.

System Evaluation

Our system evaluation aims to highlight the differences in ratings and rankings between the RS (Recommender System Library) and the LLM. These differences will signify the importance of the introduction of LLM in the recommendation process.

Evaluation Criteria:

Rating Discrepancy Analysis: We will analyze the ratings assigned by the RS and the LLM to the same set of items in the dataset. The evaluation will focus on identifying instances where the two systems assign significantly different ratings to the same item.
Ranking Comparison: We will compare the rankings of items generated by the RS and the LLM for a sample of users. The evaluation will assess how often the top-ranked items differ between the two systems.

Evaluation Process:

Data Preparation: We will select a subset of users and items from the dataset to conduct the evaluation. For each user-item pair, we will record the ratings assigned by both the RS and the LLM.
Rating Discrepancy Analysis: We will calculate the Mean Absolute Error (MAE) and Root Mean Squared Error (RMSE) between the ratings assigned by the RS and the LLM for each user-item pair. Identify user-item pairs where the MAE and RMSE values exceed a predefined threshold, indicating significant rating differences.
Ranking Comparison: Generate recommendation lists using the RS and the LLM for the selected users. Compare the top-ranked items in each recommendation list and calculate the percentage of overlap and divergence.

Interpretation of Results:

Rating Discrepancy Analysis: Higher MAE and RMSE values suggest larger differences in ratings between the RS and the LLM. These differences indicate that the LLM provides alternative perspectives or insights into item preferences compared to the RS.
Ranking Comparison: A lower percentage of overlap in top-ranked items between the RS and the LLM signifies diverse recommendation choices. This diversity suggests that the LLM considers different factors or patterns in user behavior, contributing to a more varied and personalized recommendation experience.

Related Work

Recent developments have highlighted the importance of recommender systems in delivering personalized recommendations [8]. With the emergence of Large Language Models (LLMs) in natural language processing, there has been a growing interest in leveraging their contextual understanding and adaptability for recommendation tasks [9]. Traditional recommender systems often face challenges such as poor interactivity and limited explainability [10].

The approach outlined in "Zero-Shot Recommendation as Language Modeling" states that by harnessing the capabilities of language models trained on extensive web corpora, they can estimate the likelihood of items being grouped together in a prompt, thus improving recommendation accuracy [11]. This concept of using large language models existing knowledge aligns with our idea of using them to understand item data. Considering the relationship between users and items within these models, we can give more personalized suggestions to users.

“Large Language Models are Zero-Shot Rankers for Recommender Systems” uses GPT 4 and diverges from traditional recommendation techniques [12]. This work frames the recommendation problem as a conditional ranking task with historical interactions as conditions and candidate generation models as candidates [12]. It highlights the power of zero-shot LLMs' ranking and offered strategies for their effective integration into recommendation frameworks, paving the way for enhanced user experiences and system performance [12]. There are also some studies have tested the capabilities of models like ChatGPT in recommendation scenarios, aligning them with information retrieval techniques to achieve optimal ranking performance [10]. There are studies that have evaluated ChatGPT's abilities as a zero-shot recommender system, leveraging user preferences and reranking strategies to provide effective recommendations [13]. This inspired us to follow a hybrid approach of using libraries that employ techniques like Collaborative filtering along with an LLM to improve the recommendation task.

There are frameworks like RAH [14] that have tried using LLM-based agents in their own way to Learn, Act, Critic, with a strong emphasis on user alignment [14]. Such studies have been successful and have shown efficiency in real world as well, which makes us know that is it possible to make LLM dig deep and find the connection by self-reflecting and learning the user preferences based on the options given to it.

Conclusion

In conclusion, our research efforts have resulted in designing and conceptualizing a movie recommendation system that combines the strengths of Large Language Models (LLMs) and well-established recommendation libraries. We have developed a system to generate personalized and context-aware recommendations based on individual user preferences through experimentation and design decisions.

The following part of our research will concentrate on the system's integration and evaluation, during which we will validate its effectiveness and performance. This detailed study highlights the various contributions of LLMs and recommender libraries, emphasizing their complementary roles in improving recommendation situations and user experiences.

References

[1] X. Amatriain, "Prompt design and engineering: Introduction and advanced methods,"*arXiv.org*.[Online]. Available: https://arxiv.org/pdf/2401.14423.pdf [Accessed: April 8, 2024].

[2] "Appropriate Uses For SQLite," *SQLite*. [Online]. Available: https://www.sqlite.org/whentouse.html [Accessed: April 8, 2024]

[3] T. Karl, "6 Reasons Why is Python Used for Machine Learning," *New Horizons*. [Online]. Available: https://www.newhorizons.com/resources/blog/why-is-python-used-for-machine-learning [Accessed: April 8, 2024]

[4] N. Hug, "Surprise: A Python scikit for recommender systems," *Surprise*. [Online]. Available: https://surpriselib.com/ [Accessed: April 8, 2024]

[5] *Grouplens.org.* [Online]. Available: https://files.grouplens.org/datasets/movielens/ml-latest-small-README.html [Accessed: April 8.2024]

[6] "Documentation: GPT-3.5 Turbo," *OpenAI*. [Online]. Available: https://platform.openai.com/docs/models/gpt-3-5-turbo [Accessed: April 8.2024]

[7] *Draw.io.* [Online]. Available: https://app.diagrams.net/ [Accessed: April 8, 2024]

[8] D.D. Palma, "Retrieval-augmented Recommender System: Enhancing Recommender Systems with Large Language Models," in *RecSys '23: Proceedings of the 17th ACM Conference on Recommender Systems*, September 2023, pp. 1369- 1373, doi:10.1145/3604915.3608889. [Accessed: April 8, 2024]

[9] Y. Gao, T. Sheng, Y. Xinag, et. al, "Chat-REC: Towards Interactive and Explainable LLMs-Augmented Recommender System," *arXiv.org*.[Online]. Available: https://arxiv.org/pdf/2303.14524.pdf . [Accessed: April 8, 2024]

[10] S. Dai, N. Shao, H. Zhao, et. al, "Uncovering ChatGPT's Capabilities in Recommender Systems," *arXiv.org*.[Online]. Available: https://arxiv.org/pdf/2305.02182.pdf . [Accessed: April 8, 2024]

[11] D. Sileo, W. Vossen and R. Raymaekers, "Zero-Shot Recommendation as Language Modeling," *arXiv.org*.[Online]. Available: https://arxiv.org/pdf/2112.04184.pdf . [Accessed: April 8, 2024]

[12] Y. Hou, J. Zhang, Z. Lin, et. al, “Large Language Models are Zero-Shot Rankers for Recommender Systems,” *arXiv.org*.[Online]. Available: https://arxiv.org/pdf/2305.08845.pdf . [Accessed: April 8, 2024]

[13] D. D. Palma, G. M. Biancofiore, V. W. Anelli, et. al, “Evaluating ChatGPT as a Recommender System: A Rigorous Approach,” *arXiv.org*.[Online]. Available: https://arxiv.org/pdf/2309.03613.pdf. . [Accessed: April 8, 2024]

[14] Y. Shu, H. Zhang, H. Gu, et. al, “RAH! RecSys-Assistant-Human: A Human-Centered Recommendation Framework with LLM Agents,” *arXiv.org*. [Online]. Available: https://arxiv.org/pdf/2308.09904.pdf . . [Accessed: April 8, 2024]