# Movie Assistant ChatBot - RAG approach

Abhimanyu Aryan[pg51632], Millena Santos[pg54107], and Ricardo Oliveira[pg54177]

University of Minho, 4710-057 Braga, Portugal
pg51632@alunos.uminho.com
pg54107@alunos.uminho.com
pg54177@alunos.uminho.com

**Abstract.** Movies play an important role in many cultures and daily lives, as a source of entertainment and education all over the world. Hollywood and Bollywood have vast and diverse audiences, influencing language, fashion and societal norms. A movie chatbot assistante offers an enhanced, interactive way to engage with this medium.
This paper explores the development and implementation of a movie chatbot assistant through Retrieval-Augmented Generation (RAG) with GPT. The chatbot is designed to provide relevante information about movies by leveraging the knowledge of pre-trained language model and the data retrieval capabilities of recent databases.
This article starts with a brief contextualization and the main goals. The State of the Art section investigates current solutions, then the data used and its sources, techniques, architecture, algorithms and tools are detailed in the Methods section. Finally, the Results are evaluated and debated.

**Keywords:** Large Language Models · Chatbot · Movie Recommendation · Movie Assistant · Bollywood · Hollywood · Retrieval-Augmented Generation · RAG

## 1 Introduction

Movies have been an important part of cultural expression and personal entertainment all over the world. They serve also as a means of storytelling that reflects societal values, culture and imagination. Hollywood films often set trends in filmmaking and popular culture. Bollywood, based in Mumbai, is the largest filme industry in the world in terms of the numbers of films produced and tickets sold.

The importance of movie recommendation chatbots lies in their ability to personalize content delivery according to user's prompts, which improves user engagement and satisfaction. It is also useful to ask simple question about a movie, for example, in which movie a specific song belongs to.

Recent advancements in large language models (LLMs) like GPT, have opened new avenues for enhancing movie recommendation systems. These models can process and generate human-like text, making them well-suited for developing conversational AI that can interact with users in a natural and intuitive manner.

Integrating LLMs with retrieval-augmentation generation techniques promises to create more dynamic and contextually aware recommendation systems.

This article explores the development of a movie chatbot assistant that leverages retrieval-augmented generation and GPT for enhanced movie recommendations. It provides a comprehensive review of current solutions, outlines the methods used to extract and treat data, elaborates the development by describing architecture, algorithms and tools used and, finally, presents the results and evaluation while addressing its limitations.

Our objective with this project entail:

- Providing better recommendations for movies based on context
- The ability to answer movie specific questions
- Up-to-date in-depth information for the latest movies

### 1.1 Largue Language Models

Large Language Models (LLMs) are models trained with a massive amount of textual data, which can have billions of parameters and possess the ability to generate text and understand relationships between words. Figure 1 illustrates the training process of LLMs. The first phase is pre-training, where the model is trained without human supervision using a mixture of proprietary and unlabeled data. In this sense, the model uses the input data as its own supervision. The second phase is fine-tuning, where more specific data and human feedback are introduced to adjust the parameters for better performance. Finally, the fine-tuned model can be further refined by experts using prompting techniques to perform specific tasks [1].

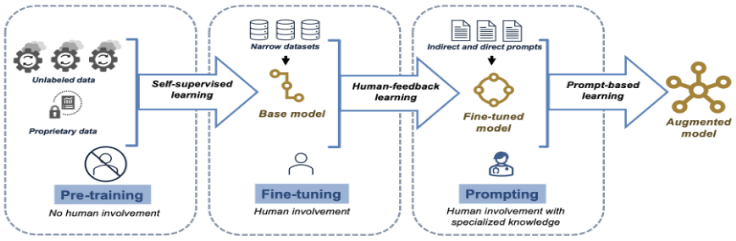


Fig. 1: Processo de treino de LLMs[1].

### 1.2 Retrieval-Augmented Generation

Retrieval-Augmented Generation (RAG) provides a solution to mitigate domain knowledge gaps, factuality and hallucination issues presented when working with Large Language Models (LLMs). RAG is particularly useful in domain-specific

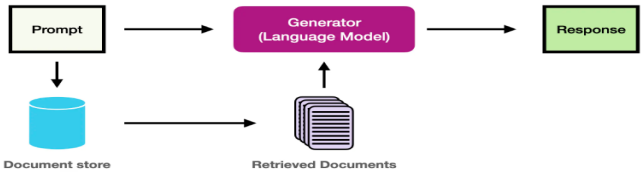


Fig. 2: Retrieval-Augmented Generation functioning.

applications that require knowledge that is continually updating, such as a movie database.

Figure 2 illustrates how it works in a simple way. RAG takes input and retrieves a set of relevant documents from a source. These documents are then concatenated with the original input prompt to provide context, which is fed to the text generator to produce the final output.

Given that the parametric knowledge of large language models (LLMs) is static, RAG enables these models to bypass the need for retraining. Instead, it allows them to access the latest information. [2]

## 2 State of the Art

### 2.1 Search Criteria

The analyzed articles were researched using specialized scientific publication databases such as `Google Scholar`. The primary search terms used included combinations of "large language models", "movie recommendation" and "movie chatbot. The main criteria for selecting articles were based on their publication date, with a focus on articles published in 2024.

### 2.2 Efficacy of Large Language Models in Predicting Hindi Movies' Attributes: A Comprehensive Survey and Content-Based Analysis [3]

This paper investigates the potential of various Large Language Models (LLMs) to predict specific attributes of Hindi movies. The primary focus is on GPT-3.5-turbo-0301, Vicuna, PaLM 2, and Dolly. The study aims to evaluate these models in predicting movie genres from audio transcripts of trailers and extracting meta-information such as director and cast details based on the movie's name and year of release.

Traditional Deep Learning models used in Recommendation Systems typically require substantial amounts of training data. This research explores whether LLMs can reduce these requirements and improve prediction accuracy. The study utilizes the Flickscore dataset, focusing on Hindi movie trailers to test the models' effectiveness. Among the evaluated models, GPT-3.5-turbo-0301 emerged as

the most effective in predicting meta-information, though all models performed better in identifying directors and cast members compared to genres.

The researchers found that GPT-3.5-turbo-0301 showed superior performance in generating meta-information from English-translated audio transcripts of Hindi movie trailers, even for movies released after its training cut-off in September 2021. However, the performance gap between GPT-3.5 and other models was not extremely significant. The study also revealed challenges in genre prediction due to issues with the audio-to-text transcription quality for older Hindi movies.

### 2.3 Large Language Models as Conversational Movie Recommenders: A User Study [4]

This paper explores the efficacy of LLMs in providing personalized movie recommendations through an online field experiment, focusing on user experiences and the quality of recommendations generated by different personalized prompting techniques.

In this study, 160 active users from an online movie recommendation platform interacted with a chatbot interface designed to simulate natural conversations. The researchers employed three types of personalized prompts—zero-shot, one-shot, and few-shot—to evaluate how different levels of personalized context affect user satisfaction. The findings revealed that users appreciated the explainability and interactivity of LLM-generated recommendations. However, these recommendations were perceived as less personalized and diverse compared to traditional systems. The specific prompting technique did not significantly influence user satisfaction, whereas the number of movies a user had watched was a more critical factor.

The study also identified key conversational patterns associated with positive and negative user experiences. Providing personal context and examples during interactions was found to enhance user satisfaction with the recommendations. Despite the limitations in delivering highly personalized and diverse suggestions, LLMs excel in providing explainable recommendations. These insights suggest that future research should focus on incorporating more personal context to improve the user experience with LLM-based recommendation systems.

## 3 Methods

### 3.1 Overall Architecture

The project architecture consists of several key components:

1. **RAG Pipeline**: Handles document indexing and query processing.
2. **Command-Line Interface (CLI)**: Provides a user-friendly way to interact with the RAG pipeline through terminal commands.
3. **Backend API**: Exposes endpoints for interacting with the RAG pipeline over HTTP, enabling integration with other services and applications.
4. **Poetry**: Manages dependencies and simplifies project setup and execution.

### 3.2 Command-Line & Guided User Interface (CLI & GUI)

The CLI is implemented using the Click library, which simplifies the creation of command-line tools. This interface allows users to perform various tasks such as creating an index and querying the RAG pipeline directly from the terminal.

The CLI commands are defined in `client.py`:

- **Command Definitions**: Each command is decorated with `@click.command()` and associated with a function.
- **Argument Handling**: Click handles command-line arguments and options, making it easy to pass parameters to the functions.
- **Command Execution**: Users can run commands to create indexes or query the system, which internally calls the respective methods of the `RAGProvider` class.

**Example Command Usage** An example command to create an index might look like this: `poetry run moviegpt index`

This command processes the documents in the specified directory and creates an index.

As for the GUI, we have implemented our frontend of our application in VueJS with easy to deploy 1-click Dockerfile. The VueJS uses axios to make request to fastapi backend endpoint i.e. **generate**

### 3.3 Backend API

The backend API is built using FastAPI, a modern web framework for building APIs with Python. FastAPI provides automatic generation of interactive API documentation and supports asynchronous request handling for improved performance.

The API endpoints are defined in `server.py` and `web.py`: **Endpoint Definitions**: Each endpoint is decorated with `@app.get()`, `@app.post()`, etc., and linked to a function. **Request Handling**: FastAPI handles incoming HTTP requests, parses parameters, and routes them to the appropriate functions. **Response Generation**: Functions process the requests using the `RAGProvider` methods and return responses to the client.

The endpoint that generates response for query is:

```
@app.post("/generate", tags=["Generate"])
async def generate(req: Request,prompt_obj: Prompt)
```

This endpoint accepts a query prompt, processes it using the RAG pipeline, and returns the generated response.

### 3.4 Data Gathering

In order to obtain the latest information about movies, including 2024, Wikipedia pages titled “Lists of Hindi films” and “Lists of American films”, specifically

focusing on the sections from the 2020s, were used. Detailed information for each movie listed between 2020 and 2024 was extracted through the `Wikipedia API`. This process involved systematically retrieving data from each link associated with the films, ensuring a thorough collection of relevant details. The extracted data was saved in PDF format so it can be fed to the RAG pipeline.

### 3.5 Tools

**Text-embedding-ada-002 v2**: Used to generate high-quality text embeddings of movie-related queries and responses. Text embeddings capture the semantic meaning of text, which is crucial for understanding user queries and retrieving relevant information from a knowledge base or database of movies. It helps in encoding user queries about movies, actors, genres, etc., into numerical representations that the chatbot can use to search for relevant information and provide accurate responses.

**ChatGPT 3.5 Turbo 0125**: As a conversational AI model, ChatGPT 3.5 Turbo 0125 generates human-like responses based on the input it receives. It is optimized for generating coherent and contextually appropriate dialogue, which is essential for interacting naturally with users. The chatbot uses this model to respond to user queries, provide recommendations, engage in casual conversation about movies, and offer explanations or summaries of plot details based on the context provided.

**Llama index**: An indexing system optimized for efficient retrieval of movie-related information. It helps in quickly searching through a large database of movies, actors, directors, genres, and other relevant details. The chatbot utilizes it to quickly fetch specific information about movies mentioned by the user, such as release dates, cast members, ratings, plot summaries, and related movies. This indexing system ensures that it can respond promptly with accurate and relevant information.

### 3.6 RAG Pipeline

**Overview** :

The RAG pipeline constitute of two main components:

1. **Index Creation**: This involves processing and indexing a collection of documents to facilitate efficient retrieval.
2. **Query Execution**: This involves using the indexed documents to retrieve relevant information in response to a query, which is then used to generate a contextually accurate response.

**Index Creation** :

The index creation process involves several steps:

– **Node Parsing**: Documents are parsed into smaller chunks or nodes to manage the context size effectively.

- **Embedding Generation**: Each node is converted into a vector representation using an embedding model. This helps in efficiently searching for relevant documents.
- **Vector Store Indexing**: The vector representations of the nodes are stored in a vector index, which facilitates quick retrieval of relevant chunks during querying.

**Query Execution** :
When a query is made, the following steps are performed:

- **Loading the Index**: The stored vector index is loaded to enable document retrieval.
- **Query Formatting**: The query is formatted appropriately to match the structure expected by the retrieval engine.
- **Document Retrieval**: Relevant documents or chunks are retrieved from the vector index based on their similarity to the query.
- **Response Generation**: The retrieved information is provided as context to the generative model, which then produces a response.

**Components and Classes** :
**RAGProvider Class**: The `RAGProvider` class is the core of the RAG pipeline. It includes methods for both creating the index and querying it.

**Initialization**: The initialization method sets up the language model and embedding model using the provided API key and model name. This prepares the class to create indexes and process queries.

**Creating the Index**: The **create_index** method handles the document processing, chunking, and indexing:

- It configures the node parser to break documents into manageable chunks.
- It sets up the prompt helper to handle context windows and token limits.
- It creates the vector store index from the parsed documents and persists it for later use.

**Querying the Index**: The `query` method processes incoming queries:

- It loads the stored index.
- It formats the query to match the expected input.
- It retrieves relevant information using a query engine.
- It uses the retrieved context to generate a response from the generative model.

### 3.7 Advanced RAG Techniques and Trulens Evaluation

We built an advanced pipeline using advanced indexing and retrieval techniques, such as sentence windowing and auto-merging retrieval. We employed Trulens for evaluation. The metrics we focused on were Groundedness, Context Relevance, and Answer Relevance.

| Method | Groundedness | Context Relevance | Answer Relevance |
|---|---|---|---|
| Basic Pipeline | 0.43 | 0.15 | 0.46 |
| Sentence Window Retrieval | 0.29 | 0.51 | 0.60 |
| Auto Merging Retrieval | 0.49 | 0.29 | 0.69 |

## 4 Results and Discussion

Looking now at the tests conducted to understand the performance of the different LLMs in this topic. Our methodology consisted of varied queries organized in the following topics:

- Depth - In depth questions about movies from recent years
- Diverse - Diverse questions about various movies and themes
- Personalized - Questions that take into account context provided
- Recent - Questions relating to movies in released in the last 3 years
- Recommendations - Similar to personalized but the context is now based on user preferences

All of the above categories were done both for Hollywood and Bollywood and all of the the tests were conducted on the 14th of June, 2024. Bellow we can observe the results for our two versions: V1 with the simple pipeline and V2 with the advanced pipeline.

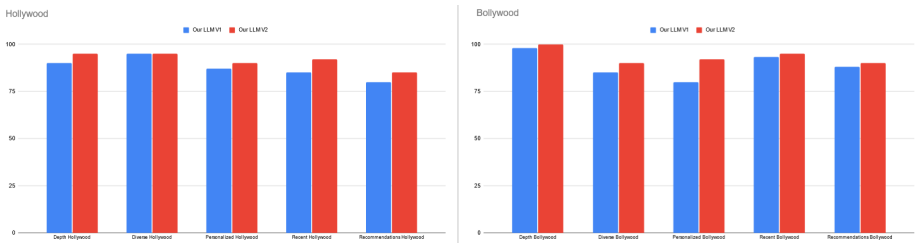


Fig. 3: Our Results

Already we can observe outstanding results. Both versions performed with more than 75% accuracy in all categories. The difference between the normal and advanced pipeline is, in most cases, lest noticeable due to the already good results. Non the less, in the categories where the first version lacked a bit more, such as diverse and personalized Bollywood, it got improved into the 90% average accuracy.

Comparing both movie markets, Hollywood and Bollywood, we can notice a small difference in results. While both present a good and similar average, Bollywood queries seem to be much more accurate when it comes to in depth information. This is due to the volumes of data in the LLMs. Even though both markets provide a similar amount of documents in order to improve the

models knowledge base, the LLM itself has already been trained before. Due to this model's training already containing information on movie data, especially Hollywood, some queries tend to hallucinate more when referring to in depth topics. With this into account, the difference in the depth category, while small, is noticeable.

Let's now take a look on how our models compare to the most popular LLMs openly available:

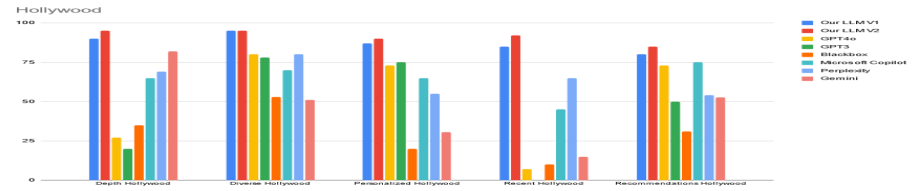


Fig. 4: Results Hollywood

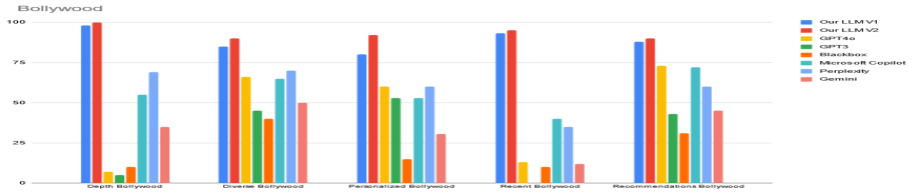


Fig. 5: Results Bollywood

At a first glance we can already tell how our models are performing relative to the other, more popular, LLMs. This is a good sign but, due to the diversity of these models, a lot comes into play when analyzing the results.

Both GPT models and Blackbox are more general and widely recognized LLMs but, when looking at the global results, seem to be under-performing across the field. GPT3 under-performance can be attributed to the training data being more limitted than the rest, not having any information in recent years. As for GPT4o and Blackbox, while they have integrations to search the web, both seem to have results bellow the expected (with GPT4o still taking the lead due to the larger amounts of training data).

Moving now to Copilot, Perplexity and Gemini, all of this models have their strongsuits tied to their capabilities on gathering information dinamically. This allowed them to, while bringing somewhat overall avarage results, be reliable sources of information. The hallucinations brought by this models seem to come, in a large sum, due to misleading or out right wrong sources. This may seem contradictory, having false sources and being reliable, but due to the direct reference to the sources it was much easier to verify the answers and their validity.

The global trend, as mentioned previously, is that every model seems to be more accurate in Hollywood information than Bollywood. While it's hard to pin

point the reason for this, as the training of these models isn't public, the more likely motive is lack of training data on the Bollywood movie scene.

But how come none of the largest and more popular models don't seem to reach the performance of our LLM. This is attributed to our pipeline. With the new and in depth information about current and past movies, our LLMs were able to better understand and give proper answers to the questions presented by the user, out-performing the general models in every category.

## 5 Conclusion

In conclusion, the development of a movie recommendation chatbot using Retrieval-Augmented Generation (RAG) with large language models (LLMs) like GPT has shown significant promise in enhancing user interactions and providing accurate information. The integration of a updated and relevant database allows our chatbot to deliver more relevant and contextually aware responses to the user's questions, outperforming many popular models in this particular sector.

Our analysis demonstrates that while general models like GPT-3 and GPT-4o have broad applications, they often fall short in specialized domains such as movie recommendations due to their limitations in training data or web searching. Our specialized LLM, benefiting from a tailored pipeline and enriched with detailed movie data, excels in delivering precise answers and personalized recommendations, even for newly released movies.

Overall, this study highlights the potential of combining advanced LLMs with robust data retrieval systems to create highly accurate conversational agents for specific subjects. Future work can explore further refinements in the pipeline, improving, for example, the user-specific answers/suggestions. Furthermore, in the future, the pipeline can be expanded to not only entail different movie markets, such as Portuguese/Brazilian movies, but also other entertainment forms, like TV Series.

## References

[1] Jesutofunmi A. Omiye et al. *Large language models in medicine: the potentials and pitfalls*. 2023.

[2] Yunfan Gao et al. *Retrieval-Augmented Generation for Large Language Models: A Survey*. 2024. arXiv: 2312.10997 [cs.CL].

[3] Prabir Mondal et al. "Efficacy of Large Language Models in Predicting Hindi Movies' Attributes: A Comprehensive Survey and Content-Based Analysis". In: *Companion Proceedings of the ACM on Web Conference 2024*. New York, NY, USA: Association for Computing Machinery, 2024, pp. 947–950. ISBN: 9798400701726. DOI: 10.1145/3589335.3651496. URL: https://doi.org/10.1145/3589335.3651496.

[4] Ruixuan Sun et al. *Large Language Models as Conversational Movie Recommenders: A User Study*. 2024. arXiv: 2404.19093 [cs.IR].