Zero-shot Recommendation System using open-source LLMs for Airline Industry and Optimization techniques in Retrieval Augmented Generation (RAG)

Sajin Mohamed Pallikkathodi Erathali – Student ID: 23037601

Dissertation in partial fulfilment of the requirements of the degree of
Master of Science in Artificial Intelligence and Machine Learning
submitted to the
Department of Computer Science and Information Systems

Supervisor: Nikola Nikolov
August, 2024

Data Source and Generative AI

Data Source:

1. Suthar, S., 2021. Airlines Reviews. [online] Kaggle. Available at: https://www.kaggle.com/datasets/sujalsuthar/airlines-reviews [Accessed 20 March 2024].

2. Kumar, M., 2022. Airline Passengers Booking Data. [online] Kaggle. Available at: https://www.kaggle.com/datasets/manishkumar7432698/airline-passangers-booking-data/data?select=Customer_comment.csv [Accessed 20 March 2024].

3. Massaron, L., 2023. Sherlock Holmes Q&A with Gemma: Fine-Tuning. [online] Kaggle. Available at: https://www.kaggle.com/code/lucamassaron/sherlock-holmes-q-a-with-gemma-fine-tuning [Accessed 20 August 2024]

Generative AI use:

1. No Generative AI is used in the research material presented apart from some paraphrasing.

Table of Contents

Data Source and Generative AI ***2***

Table of Contents ***2***

Table of Figures ***4***

Table of Tables ***6***

Abstract ***7***

Declaration ***8***

Acknowledgments ***9***

1. Introduction ***10***

1.1. Motivation **10**

1.2. Research Question **13**

1.3. Overview of Dissertation **14**

2. Literature Review ***15***

2.1. Recommender Systems **15**

1. Content-based Filtering 16
2. Collaborative Filtering 16

2.2. Transformer Architecture **17**

2.2.1. Input Embedding 18
2.2.2. Positional Encoder 18
2.2.3. Self-Attention (Single-Head Attention) 19
2.2.4. Multi-Head Attention 20
2.2.5. Add and Norm 22

2.2.6. Decoder 22
2.3. Large Language Models 23
2.3.1. Llama-2 (7B) 23
2.3.2. Mistral (7B) 27
2.3.3. Llama-3 (8B) 30
2.3.4. Types of Prompting 31
2.4. Knowledge Graph (KG) 32
2.4.1. Why KG 34
2.5. Retrieval Augmented Generation (RAG) 36
2.5.1. Working of RAG 36
2.5.2. Embedding the Chunks 38
2.5.3. Sentence Transformer 41
2.5.4. Multiple Negatives Ranking Loss 45
2.5.5. Token Optimization 46
2.5.6. Evaluation of RAG 49
3. Methodology 55
3.1 . Dataset 55
3.1.1. Dataset for Recommender System 55
3.1.2. Reasonable Assumptions made 56
3.1.3. Dataset for RAG optimization 59
3.2. Zero-shot Recommender System 61
3.2.1. Simple Recommender System 61
3.2.2. Recommender System with Knowledge Graph 62
3.3. Retrieval Augmented Generation optimization 67
3.3.1. Evaluating the LLMs 69
3.3.2. Fine-tuning the embedding 69
3.3.3. Multi-query context Expansion 72
3.3.4. Re-ranking with Cross Encoders 73
3.4. Token Optimization 74
3.4.1. Byte-Pair Encoding (BPE) 74
3.4.2. Sentencepiece Tokenization and Wordpiece Tokenization and LSA summarizer 75
4. Results 78
4.1. Simple Recommender without Knowledge Graph 79
4.2. Zero-Shot Recommender with KG 82
4.3. RAG Optimization 85
4.3.1. Fine-tuning Embeddings 85
4.3.2. Cross-Encoder evaluation with and without Multi-query expansion 93
4.4. Token optimization techniques 94
4.4.1. Byte-Pair Encoding (BPE) 94
4.4.2. BPE with Latent Semantic Analysis Summarizer 95
4.4.3. Sentencepiece 96
4.4.1. Sentencepiece with LSA 97
4.4.2. Wordpiece 98
4.4.3. Wordpiece with LSA 99

5. Conclusion ***101***
6. Future work ***103***
7. References ***104***

Table of Figures

Figure 1. Example of graph-based recommender system (Wang S. et al. 2021). 12
Figure 2. Transformer Architecture describing Encoder and Decoder blocks [source: (Vaswani et al. 2023)]. 18
Figure 3. Q, K and V matrices of dimension $sequence\ length \times ddimension$ multiplied with the parameter matrices WQ, WK and WV with shape $ddimension \times ddimension$ to form Q', K' and V' which are split into smaller matrices of size $ddimension$ such that each head can observe the full sentence while focusing on a smaller portion of the embeddings for each word. The multi-head attention is calculated for the resultant matrices. [source: Jamil, U. (n.d.)] 21
Figure 4. Helpfulness human evaluation results for Llama-2-Chat compared to other open-source and closed-source models The 95% confidence interval for this evaluation ranges from 1% to 2%, indicating a high level of precision and confidence in the estimated parameter. [source: (Touvron et al. 2023)]. 23
Figure 5. BLEU score on an English to German Translation task [source: (Shazeer 2019)] 25
Figure 6. Improvement in Inference time for multi-query attention when compared to multi-head attention [source: (Shazeer 2019)] 26
Figure 7. Architecture of grouped-query method: In multi-head attention, each of the H query heads has its own corresponding key and value heads. Multi-query attention, on the other hand, uses a single set of key and value heads that are shared across all query heads. The grouped-query attention method serves as a middle ground, where a single set of key and value heads is shared within each group of query heads, effectively bridging the gap between multi-head and multi-query attention. [source: (Ainslie et al. 2023)] 26
Figure 8. Effective context length where the first 4 tokens from the input is passed to the 4th token of the Layer 1 which passed to the 7th token of layer 2. Even though the 1st token is not directly related to the 7th token, its related via the layer 1 which contains the information about the 1st token. [source: (Jiang et al. 2023)] 28
Figure 9. Helpfulness human evaluation results for Llama-3 compared to other open-source and closed-source models. [source: (Dubey et al. 2024)]. 30
Figure 10. Example of Knowledge Graph with three nodes and two relationships between them. Passenger entity can have multiple relationships with the flight entity such as Boards and Books. 33
Figure 11. Working of Retrieval Augmented Generation (RAG) [source: (Lewis et al. 2021)]. 36
Figure 12. Bi-encoder and Cross-encoder architecture [source: (Singh 2024)] 41
Figure 13. Data preparation for Multiple Negative Ranking Loss [source: Nicholas Broad, (2023)] 45
Figure 14. Formation of similarity matrix using embeddings created by stacking individual sentence embeddings [source: Nicholas Broad, (2023)] 46
Figure 15. Comparison of NDCG with other metrics [source: Aporia, 2023]. 53
Figure 16. Category of Issues suggested by LLM on the dataset created. 57
Figure 17. Monthly number of flights taking off in the dataset. 57
Figure 18. Percentage of Acceptable Recommendations created by LLM that is coherent with real world scenarios 58

Figure 19. Acceptable Recommendation based on membership level. 59
Figure 20. Architecture of a simple recommender system using LLM. 61
Figure 21. Recommender with Knowledge Graph (KG). 62
Figure 22. A sub-section of the Knowledge Graph (KG). 63
Figure 23. An Enlarged view of the relationships between the nodes in the KG. 64
Figure 24. Attributes of Review Node such as Membership_levelm embedding of the concatenated membership, review, and category, the acceptable recommendation. 65
Figure 25. RAG Pipeline showing the different steps in the retrieval augmented generation and the way the result is returned to the user provided a query from the user is consumed by the system [source: (Özker 2024)]. 68
Figure 26. The embeddings of the Train Dataset projected onto a 2-dimensional vector space with the query (red X) and the retrieved documents (green circle) as highlighted. 70
Figure 27. Illustration of how bi-encoders and cross-encoders work together to improve result retrieval [source: Osanseviero, O., 2023]. 74
Figure 28. Mean value of the different metrics using Llama-2 as the LLM for the Simple Recommender without knowledge graph. 79
Figure 29. Mean value of the different metrics using Mistral as the LLM for the Simple Recommender. 80
Figure 30. Violin plot of the data distribution for Llama-2 results. 81
Figure 31. Violin plot of the data distribution for Mistral results. 81
Figure 32. Mean value of metrics illustrated for a Zero-shot Recommender with KG implementation with Llama-2 as the LLM. The results have an improved score when compared to the one without KG. 82
Figure 33. Mean value of metrics illustrated for a Zero-shot Recommender with KG implementation with Mistral as the LLM. The results have an improved score when compared to the one without KG. 83
Figure 34. Violin Plot showing the distribution of results for Llama-2 for Recommender with KG the median score highlighted. 84
Figure 35. Violin Plot showing the distribution of results for Mistral for Recommender with KG with the median score highlighted. 84
Figure 36. The pictorial representation of vector embedding for Sentence-tranformers/all-MiniLM-L6-v2 for the question "Who is Sherlock Holmes ?" presented on a 2-dimensional plane with the query (marked with red X) and the retrieved relevant documents (marked with green circles). 86
Figure 37. The performance metrics of BAAI/bge-small-en-v1.5 embedding on the test data before fine-tuning. This table provides a detailed comparison of the Cosine and Dot product similarities across various evaluation metrics such as Accuracy, Precision, Recall, MRR, NDCG, and MAP at different k values. The metrics are presented for k=1,3,5,10,100, where k is the number of recommendations, showcasing the performance variations between the two methods. 87
Figure 38. The performance metrics of BAAI/bge-small-en-v1.5 embedding on the test data after fine-tuning. The improved figures show the improvement in performance of the results when using the fine-tuned embedding model. 87
Figure 39. The vector embeddings for the fine-tuned BAAI/bge-small-en-v1.5 model, depicting the question "Who is Sherlock Holmes?" on a 2-dimensional plane. The query is represented by a red X, while the retrieved relevant documents are shown as green circles. 88
Figure 40. The pictorial representation of the vectors on a 2-dimensional plane with the query highlighted with a red X and the relevant documents retrieved highlighted with green circles for Sentence-transformers/all-MiniLM-L6-v2 for a question “Who is Sherlock Holmes ?”. 89

Figure 41. The performance metrics of Sentence-transformers/all-MiniLM-L6-v2 for the Sherlock Holmes Dataset on the test data. All of the metrics values shows a decrease when compared to the fine-tuned BAAI/bge-small-en-v1.5. 89
Figure 42. The performance metrics of int/float/multilingual-E5-small embedding for the Sherlock Holmes Dataset on the test data at different values of k. The metrics demonstrate a notable similarity in results when compared to the fine-tuned BAAI/bge-small-en-v1.5 model. 90
Figure 43. The performance metrics of fine-tuned int/float/multilingual-E5-small embedding for the Sherlock Holmes Dataset on the test data. 91
Figure 44. The pictorial representation of the vectors represented on 2-dimensional plane for the fine-tuned int/float/multilingual-E5-small. 92
Figure 45. Comparison of vectors of BAAI/bge-small-en-v1.5 and int/float/multilingual-E5-small embeddings fine-tuned when plotted on a 2-dimensional plane with the query “Who is Sherlock Holmes ?” highlighted in red X and the retrieved documents highlighted using green circles. 92

Table of Tables

Table 1. RAGAS metrics for a simple RAG. 85
Table 2. RAGAS metrics with cross-encoder re-ranking and without multi-query expansion and int/float/multilingual-E5-small. 93
Table 3. RAGAS metrics with cross-encoder re-ranking and with multi-query expansion and int/float/multilingual-E5-small. 93
Table 4. Performance metrics with cross-encoder re-ranking and without multi-query expansion and int/float/multilingual-E5-small embedding. Here K = 3 93
Table 5. Performance metrics with cross-encoder re-ranking and with multi-query expansion and the fine-tuned int/float/multilingual-E5-small embedding. Here K = 3 93
Table 6. Byte Pair Encoding evaluated with RAGAS metrics. 94
Table 7. Byte-Pair Encoding with LSA evaluated with RAGAS metrics 95
Table 8. Sentencepiece evaluated with RAGAS metrics 97
Table 9. Sentencepiece with LSA evaluated with RAGAS metrics 97
Table 10. Wordpiece evaluated with RAGAS metrics 98
Table 11. Wordpiece with LSA evaluated with RAGAS metrics 99

Abstract

Recommender systems have become an essential tool in various online applications, providing personalized content and services to individual users. The use of Large Language Models (LLMs) in recommender systems has gained significant attention due to their capacity to comprehend and produce human natural language.

Airline passengers often face issues like flight cancellations, delays, or seating problems, which they discuss on various online platforms such as Trustpilot. These issues can lead to heated exchanges between passengers and airlines, ultimately causing customers to lose their loyalty. To retain passenger loyalty, airlines can use a recommender system that suggests alternative solutions such as complimentary stays, meals, or refunds based on the review of the passenger to address these concerns effectively.

Currently, such issues are handled with same coupons for every customer which may not be the ideal solution as every customer would not be happy with the option. Hence, this dissertation investigates the applicability of Open-source Large Language Models (LLMs) namely, Llama2 and Mistral to solve this issue by analyzing the passenger reviews and the data which the airlines possess and provide top 3 recommendations using the travel history of the passengers and the rating given by the passengers that already faced these issues and provide an explanation for the recommendation. The reason for using LLMs here is that this process can be done without any preprocessing of the data. Additionally, we look at the advantage provided by introducing a Knowledge Graph Database into the system architecture and evaluating that via the traditional metrics such as NDCG, MRR, MAP etc. as well as RAGAS framework to measure faithfulness, answer relevancy, context recall, context precision.

Moreover, the study also explores and compares various ways through which can optimize the output for Retrieval Augmented Generation (RAG) such as context enhancements, reranking relevant documents to obtain most relevant information and token optimization techniques which helps in forming more efficient and accurate responses. Reducing the number of tokens help speed up response time, and provide more accurate and relevant information, thereby providing clear and concise output. This also make the application scalable as it contributes to better resource management thereby reducing the infrastructure cost.

Declaration

I hereby certify that the material I am submitting for assessment toward the Master of Science degree is entirely my own work. I have made every effort to ensure that the work is original and, to the best of my knowledge, does not breach any copyright laws. Any collaborative contributions have been clearly indicated and acknowledged, and all supporting literature and resources have been properly referenced.

Name : Sajin Mohamed Pallikkathodi Erathali
Student ID : 23037601

Signature: …… Date: ……..

Acknowledgments

I would like to thank my supervisor, Dr. Nikola Nikolov for his guidance, support and valuable insights throughout the course of my research. His expertise in NLP was instrumental in helping form this dissertation and encouraging me to explore new avenues when working with the dissertation. It was a privilege to work under his supervision.

I would also like to extend my gratitude to Dr. Emil Vassev, the course director who helped me create a schedule for planning out the dissertation to be completed within the targeted timeline. I am also grateful to Dr. Tabea De Wille for helping me understand how to write a dissertation through the Research Methods and Specifications module.

I extend my special thanks to my family for their endless encouragement, understanding, patience and their blessings throughout.

1. Introduction

1.1. Motivation

Airline passengers occasionally encounter pre-boarding challenges at airports due to maintenance, security, late flight arrivals, pilot rest requirements, and other issues. These circumstances can result in flight cancellations, delays, reroutes, or postponements, leading to a loss of customer loyalty and impatience as the airline representative at the airport has limited actions to pursue to address the concerns for each passenger.

Current airline practices often involve offering food coupons, accommodation vouchers, or full refunds etc. to all the affected passengers. However, such compensation may not satisfy every customer, for instance, passengers who travel for business might face significant financial implications, or passengers who travel due to health issues could experience added stress due to delays, or people travelling with infants would find it extremely unpleasant, whereas others may be more flexible. As a result, customers often post negative reviews on platforms such as TrustPilot, Airline Ratings, etc. This can result in huge economic loss for the airlines when these issues happen during peak travel seasons, such as during Christmas due to operational costs, damage in reputation, legal and regulatory issues, employee morale and productivity, and even new customer acquisition and reduction in profit margins.

This dissertation proposes to develop a Zero-shot recommendation system using two prominent open-source Large Language Models (LLMs), namely Mistral (7B) and Llama-2 (7B) and compare their adaptability to Airline industry. The idea for a zero-shot recommender for tackling this problem arises from the paper (Wang and Lim 2023).While Deep Neural Networks (DNNs) were used for recommender systems due to their ability to comprehend high-order dependencies (Zhang *et al.* 2020), they possess some major disadvantages such as difficulty in understanding the customers interest and to obtain textual information and lacking the ability to generalize to various seen or unseen scenarios and cannot reason on their predictions (Zhao *et al.* 2024) as they often act as black boxes, and their inability to form multi-step reasoning. The advantage offered by Large Language models is that they are pretrained with a few billion parameters, they provide good reasoning abilities and generalization capabilities. Due to this ability, LLMs do not need extensive fine-tuning on every task that it has to handle, although, advanced techniques like in-context learning (Wei *et al.* 2023) can help better generalize the results. Hence this work is an investigation into Large

Language Models without explicitly fine-tuning the model and determine the accuracy of recommendation provided by the LLM using some traditional metrics and RAGAS metrics.

There exists two approaches to building a recommendation system namely, Content-based filtering and Collaborative Filtering (CF). Content-based filtering on the other hand, utilizes other information about the users or the items, such as item description or users age etc. to provide a recommendation, that is more accurate. Collaborative Filtering models user preferences for items by leveraging historical user-item interactions, such as ratings (Sarwar *et al.* 2001) to estimate the probability of future interactions. One of the commonly used CF based methods is Matrix Factorization (Fan *et al.* 2019), where it represents both items and users with vectors derived from item rating patterns. A high alignment between item and user factors results in a recommendation (Koren *et al.* 2009). However, the traditional approaches for content-based filtering have issues when the data is sparse. Due to sparsity in the data, difficulty arises during calculation of the similarity between users and items. Another problem that is faced by the CF is the cold start problem, where there exists a new user or a new item because of it having no interactions between the user or the items. In order to solve this, many proposals that lie around transmitting additional information such as item attributes or item reviews have been proposed (Zhen *et al.* 2009).

The proposed recommendation system leverages user feedback extracted from platforms such as Trust Pilot or Airline Rating to recommend personalized solutions tailored to individual travel histories and the experiences of previous customers who faced similar issues and their ratings for the journey. Even though, LLMs could reason and generalize without requiring any preprocessing of the data, one of the well-known issues with the LLMs is "hallucination". This phenomenon occurs when the model generates output that may sound true but are factually incorrect and may even contradict the input data from which it has to generate the output (McKenna *et al.* 2023).

To address the hallucination problem inherent in LLMs, and to provide additional information about the items, Knowledge Graphs (KGs) are employed and compare the results with an architecture that does not possess the KGs in their system.

> *"A KG organizes entities and relations of the real world with a graph, which effectively expresses semantic relations between entities. By constructing the item KG, user KG, or user-item KG, more accurate users' preferences can be captured by mining the relations between the entities. Compared with traditional Recommender*

Systems (RSs), KG-based RSs (KGRSs) make the reasoning process available, which can improve the explainability of RS"

(Gao *et al.* 2023)

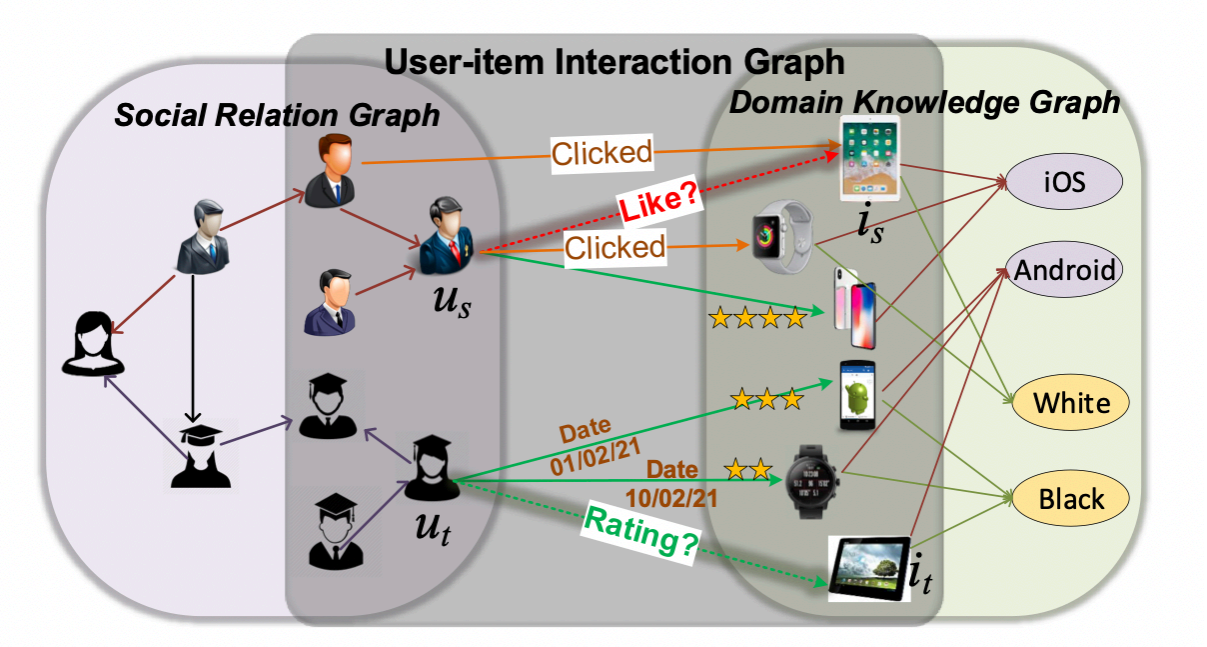

Figure 1. Example of graph-based recommender system (Wang S. et al. 2021).

Knowledge Graphs were introduced in the 1982, but by the end of 1980s they were adapted into medical and social science domain (Nurdiati, S. 2008). It was later brought into linguistic domain, which was taken by Google in 2012 into their search engine optimization.

Since any recommender system can be picturized as a graph, where objects under consideration can be viewed as tightly connected nodes using relationships as edges, graph-based learning can effectively model the relationships between users and items. This allows for better understanding of user preferences and item similarities. Representations like this eliminates the problem of each user getting the same recommendations regardless of their preferences and get a more personalized response. A knowledge graph can be interpreted as a heterogenous network as it contains multiple nodes and relationships between them and provides a representation ability as different attributes of each node can be observed by following multiple edges in the graph (Zhang *et al.* 2016).

Hence, Knowledge Graphs provide more information about the problem under consideration and are ideal for designing a recommender system. The Figure 1 shows an example of user's

interaction with different devices and their ratings. The results presented in the study evaluates the accuracy of the recommendation using the metrics like the NDCG_score, Recall@3, Precision, MAP_score, MRR, hit rate, F1_score.

In the last 2 years (2023, 2024), LLMs have become a popular tool for creation of Chatbots due to their adaptability and comprehension of textual input. These systems are placed into production with the use of Retrieval Augmented Generation (RAG) techniques. RAG includes the combination of retrieval-based methods such as finding the relevant documents from a corpus and generation-based methods such as generating text that are coherent with the retrieved information. This study looks at the optimization of the retrieval-based methods such as fine-tuning embeddings used to retrieve documents, context enhancement for better retrieval, reranking the retrieved documents. The study also looks at token optimization techniques such as Byte-Pair Encoding (BPE), Sentencepiece, Wordpiece, and Latent Semantic Analysis applied with these to make the text provided to the LLMs shorter to generate the response faster and even reduce the cost in case proprietary models such as GPT 3.5, or GPT 4 need to be used.

The process of fine-tuning LLMs requires a lot of time and resources. It needs a significant amount of training data, which might be difficult to come by in domains with a lack of high-quality data. For instance, in the case of airline data, it may be difficult to build a big enough dataset for training due to the restricted availability of reviews.

1.2. Research Question

The three research questions considered here are :

1. Which of the two large language models, Mistral or Llama2, is best suited for application in the airline industry as a zero-shot recommender? The focus is on recommending alternatives to passengers based on their travel history and ratings from other customers with similar issues.
2. How much of an improvement in the predicted recommendations evaluated using metrics such as precision, recall etc. can be brought about by introducing Knowledge Graphs into the architecture?
3. Different ways to improve Retrieval-Augmented Generation (RAG) through optimization techniques such as fine-tuning embedding, context enhancements such as

multi-query expansion of the question, re-ranking the documents obtained from the vector store, and token optimization to reduce the number of tokens?

1.3. Overview of Dissertation

The research paper is structured as follows. Chapter 2 reviews the literature surrounding the topics discussed, starting with an introduction to recommender systems and a detailed exploration of Transformer architecture, including the components of the encoder and decoder. It then examines various language models such as Llama-2, Mistral, and Llama-3, with an in-depth look at Mistral's architectural components like sliding window attention and KV caching. The concept of Knowledge Graphs (KG) and their necessity is explored, followed by the Retrieval Augmented Generation (RAG) system and its workings. This section also discusses different types of embeddings, particularly Sentence Transformers, and their origins, including Cross-encoders and bi-encoders used in the implementation section. Additionally, it covers three embeddings used for fine-tuning and comparison, various token optimization techniques, and the evaluation of RAG using metrics like RAG and NDCG, as well as the Information Retrieval Evaluator for scores like F1, MAP, and MRR.

Chapter 3 details the methodology for implementing the Zero-shot recommender, beginning with exploratory data analysis of the two datasets used and modified. It compares the architectures of the recommender system with and without the KG and outlines the optimization methodologies implemented. Chapter 4 presents the findings, analyzing which of the two LLMs is best suited for a Zero-shot recommender in the Airline Industry, with an examination of the two different architectures used and the improvements from using the knowledge graph. It also evaluates the optimizations introduced by different RAG approaches using the RAGAS framework. Chapter 5 concludes with an overview of the project, a summary of the findings, and proposals for future research.

2. Literature Review

In this chapter, we delve into various types of recommender systems and examine the large language models (LLMs) used, focusing on their architecture and distinguishing features. The chapter includes an in-depth review of transformer architecture and the significance of knowledge graphs, as well as a discussion on the optimization techniques that can enhance the efficiency of Retrieval Augmented Generation (RAG) systems. Additionally, we explore and compare different token optimization methods used in this study. Finally, the chapter concludes with an overview of key evaluation metrics.

2.1. Recommender Systems

Recommender systems have become popular in the recent years, especially due to growing popularity of online media platforms like over-the-top (OTT) platforms like Netflix, Amazon Prime Video, Hulu etc. and due to increase in online shopping and these platforms replacing traditional markets for entertainment. The companies providing these services need users to interact with their products to generate revenue as per their business models (Yawalkar *et al.* 2022). This is especially true for the Airline industry as customer satisfaction is a critical factor which may lead to increased loyalty and repeat business thereby increasing their profit shares.

Artificial Intelligence (AI) has played a significant role in advancing recommender systems by integrating AI techniques such as Natural Language Processing (NLP) to interpret multilingual natural languages and generate text that can be human comprehensible (Zhang *et al.* 2021). A recommender system can be formulated as $u : C \times S \rightarrow R$, where C is the set of users, S is all the acceptable recommendations, and R is a recommendation list containing a set ranked items, and u being the utility of the item for the user. For each user $c \in C$ we want an output so that item $s' \in S$ have a maximum value of utility for the user.

$$\forall c \in C, \qquad s'_c = argmax_{s \in S}\, u(c, s)$$

(Adomavicius and Tuzhilin 2005)

For a recommender the rating of the item is considered as the utility of the item.

There exist various techniques to creating a recommendation system:

1. *Content-based Filtering*

Content-based filter uses only the content of the item description to calculate the recommendation probability for a user and aims to provide recommendations similar to that of the user's previous preference. The items are represented by attributes such as flight information, arrival date, departure date, departure airport, arrival airport etc. The commonly used techniques include creating a vector space with Term Frequency-Inverse Document Frequency (TF-IDF) weighting (Salton *et al.* 1975). A user profile is created based on the interaction of the user with the items and the system compares items attributes with the user's profile and finds the item to recommend. The advantage of this approach is that it critically depends on the item representation hence less dependent on the user data and therefore, can recommend new items to users, which eliminates the cold start problem for the items being recommended. It however suffers from cold start problem when a new user comes into the picture, since this user has not profile information as the user has had no interactions to the recommendations. Another problem faced by content based recommenders is that they only recommend items similar to what the user has interacted with thereby overlooking newly added items (Zhang *et al.* 2021).

2. *Collaborative Filtering*

Collaborative Filtering like the name suggests depends on user's attributes like ratings unlike the content-based filtering that only depends on user's historical interactions. Due to this fact, it is being widely adopted in the recommender systems, and users with similar interest can acquire similar contents. In order to achieve this, CF technique relies on the information provided by the users who have similar preference to the target user.

Memory-based CF, an early generation CF, uses the nearest neighbour algorithm to calculate user or item similarity, and is valued for its simplicity, efficiency, and accuracy. Cold start problem is inherent here as well, when a new user or item is ingested by the system, as there exists no rating or interaction for the item or the user. Model-based CF on the other hand uses machine learning techniques such as Matrix Factorization (MF), Singular Value Decomposition (SVD) etc. to resolve the issues of Memory-based CF where it uses other side information review, tags, location in addition to the user-item rating matrix (Shi *et al.* 2014). For instance, in MF the user-item matrix dimensions are reduced significantly by decomposing the user-item matrix into two lower-dimension matrices, which represent the

underlying characteristics that explain the ratings obtained. This helps in projecting the user and item space on to a latent factor space where each user and item is represented as a vector in the space. The advantage of this is that sparsity is reduced whereby we can obtain any user item interaction with the dot product of these vectors due to the projection onto the latent factor space (Luo *et al.* 2016).

The development of advanced large language models (LLMs) has driven considerable advancements in recommender systems, introducing innovative approaches like leveraging LLMs for recommendations (referred to as LLM4Rec) (Bao *et al.* 2023). These LLMs are distinguished by their exceptional capabilities in understanding and generating language, generalizing across different contexts, and planning (Achiam et al 2023). An advantage possessed by LLMs are that, they can leverage the information or knowledge gained and reasoning abilities to adapt to new tasks without extensive fine-tuning for each task. This can be achieved through providing clear instructions, and the generalization capabilities can be further improved by using in-context learning, which involves the model generating text based on a context or prompt provided (Brown *et al.* 2020).

Typically, LLMs such as those using traditional transformer architectures.

2.2. Transformer Architecture

Since the problem at hand involves understanding the user review to suggest a recommendation, language comprehension is critical aspect of success for the recommendation. Earlier this was accomplished using sequence-to-sequence models formed by using Recurrent Neural Networks (RNNs), where the input is passed on to the successive hidden layers with the successive input tokens on each time steps (Lipton 2015). However, this suffered from a few critical disadvantages such as slower computation as we need to do the computation for every token from the input. Hence, for longer input, the training time for the network increases. Another problem is the Vanishing Gradient problem (Hochreiter 1998), where calculation of derivative of the loss function with respect to the weights, in longer networks would lead to a very small value thereby leading to the network learning very slowly. When the input text becomes longer, the effect of the first token from the input would be negligible when considering the last token. This would not be ideal for textual processing as the context may be established in the first few tokens. Hence, the introduction of Transformer.

The Transformer architecture is composed 2 blocks, namely encoder and decoder as highlighted in red colour blocks in the Figure 2. The output of the encoder is sent as the input to the decoder.

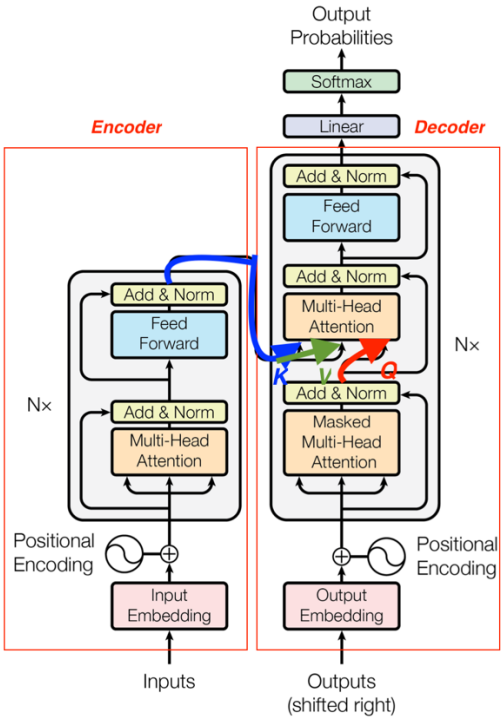

Figure 2. Transformer Architecture describing Encoder and Decoder blocks [source: (Vaswani et al. *2023)].*

2.2.1. ***Input Embedding***

The input is a list of words that are converted into tokens, which serve as the building blocks of large language models (LLMs). Tokens can be thought of as the fundamental units of a sentence, and when combined, they can form meaningful text. These tokens are mapped into numbers which represent the position of the words in the vocabulary which comprises of all the possible words in the training set. The result of this step is called as *Input IDs*. These are mapped into a vector of $d_{dimensions}$. These embeddings can be thought of as numbers that represent the information contained in the words. These embeddings are modified as the model gets trained to better represent each token, as per the loss function chosen to train the model. In the work done by Vaswani *et al.* (2023), the dimension of the embedding is chosen as 512.

2.2.2. ***Positional Encoder***

The next step in the encoder is to pass the result to positional encoder to represent a pattern that the model can learn and distinguish the context between the words. This operation is

completed by creating another embedding (vector of size 512) which is only computed once, and the result is summed with the embedding created earlier to make the model understand information about the position of each word. The positional encoding is computed by using the formula :

$$PE_{(pos,\,2i)} = sin(pos/10000^{2i/\,dmodel}) \tag{2.1}$$

$$PE_{(pos,\,2i+1)} = cos(pos/10000^{2i/\,dmodel}) \tag{2.2}$$

where pos denotes the position and i denotes the dimension. Equation (*2.1*) is used to calculate the positional encoding of the even positions of the created embedding, and equation (*2.2*) is used to calculate the positional encoding of the odd positions. This approach was chosen by because it would make it simple for the model to learn how to attend (pay attention to each word) by relative positions. This is only done only once when the model is created and for every sentence we use the same positional encoding.

The next step is passing this information into multi-head attention. However, to understand that, it's important to first understand single-head attention

2.2.3. Self-Attention (Single-Head Attention)

Self-Attention allows the models to relate the tokens to each other. For instance, the words in the same sentence would be related to each other and this is to be understood by the model. The attention function is described as mapping a set of key-value (K, V) pairs and a query (Q) to an output :

$$Attention(Q,K,V) = softmax\left(\frac{QK^T}{\sqrt{d_k}}\right)V \tag{2.3}$$

The output is the weighted sum of values, where the weights are calculated by a compatibility function of the query with the key associated with it (Vaswani *et al.* 2023).

The dot product of Q and K^T would lead to LxL attention matrix, where L is the sequence length. The dot product yields the dot product of the embedding of each word with every other word in the sentence, representing a score of the relationship between the words. This value is normalized with the dimension of the matrix ($\sqrt{d_k}$), which denotes the part of the

embedding to which this particular attention head will attend to. The result of the multiplication of Q, K and V leads to a matrix that denotes :

1. meaning of the word provided by the input embedding,
2. the position of the word provided by the positional encoding,
3. and capturing the relationship of the word with all other words.

In order to generate an Auto-regressive model, which means that the next token depends only on the previous tokens, causal mask can be applied on the matrix. This involves multiplying all the values that come after the current token with a $-\infty$, which would be converted by the application of softmax to the matrix which would result in 0 values. Another functionality achieved by the application of softmax is that all the values in each row to sum up to 1.
Some of the properties of the Self-Attention are :

1. The Self-Attention is permutation invariant ie. changing the order of the words changes the position of the order of the words in the result and does not result in change of values.
2. It does not require any parameters
3. Since the Self-Attention is comprised of a matrix which calculates the dot product of embeddings of each of the words with the others, the maximum values can be expected across the diagonals of the matrix.

2.2.4. Multi-Head Attention

Multi-head attention allows the model to simultaneously focus on information from different representation subspaces and positions. In contrast, using just one attention head would average the information, limiting its effectiveness.

The input after positional embedding is copied into 3 identical matrices Q, K and V of dimension $sequence\ length \times d_{dimension}$. The Multi-Head Attention multiplies the Q, K and V values with parameter matrices W^Q, W^K and W^V with shape $d_{dimension} \times d_{dimension}$. Hence, multiplying these matrices will yield a matrix with the same dimension as Q, K and V. The resultant matrices Q', K' and V' are split by $d_{dimension}$, so that every head will see the full sentence but a smaller part of embedding of each word and not the entire embedding. This smaller part is calculated as $d_{dimension}$/number of heads in the multi-head attention as demonstrated in Figure 3.

The attention between these smaller matrices Q_1', K_1', V_1' can be calculated using the equation (*2.3*). and

$$head_i = Attention\ (Qw_i^Q, Kw_i^K, Vw_i^V) \tag{2.4}$$

These head information can be combined or concatenated using

$$Multihead\ (Q, K, V) = Concat\ (head_1, head_2, \dots, head_n)W^O \tag{2.5}$$

Where the projections are parameter matrices $W_i^Q \in R^{d\ dimension \times dk}$, $W_i^K \in R^{d\ dimension \times dk}$, $W_i^V \in R^{d\ dimension \times dv}$ and $W^O \in R^{hdv \times d\ dimension}$.

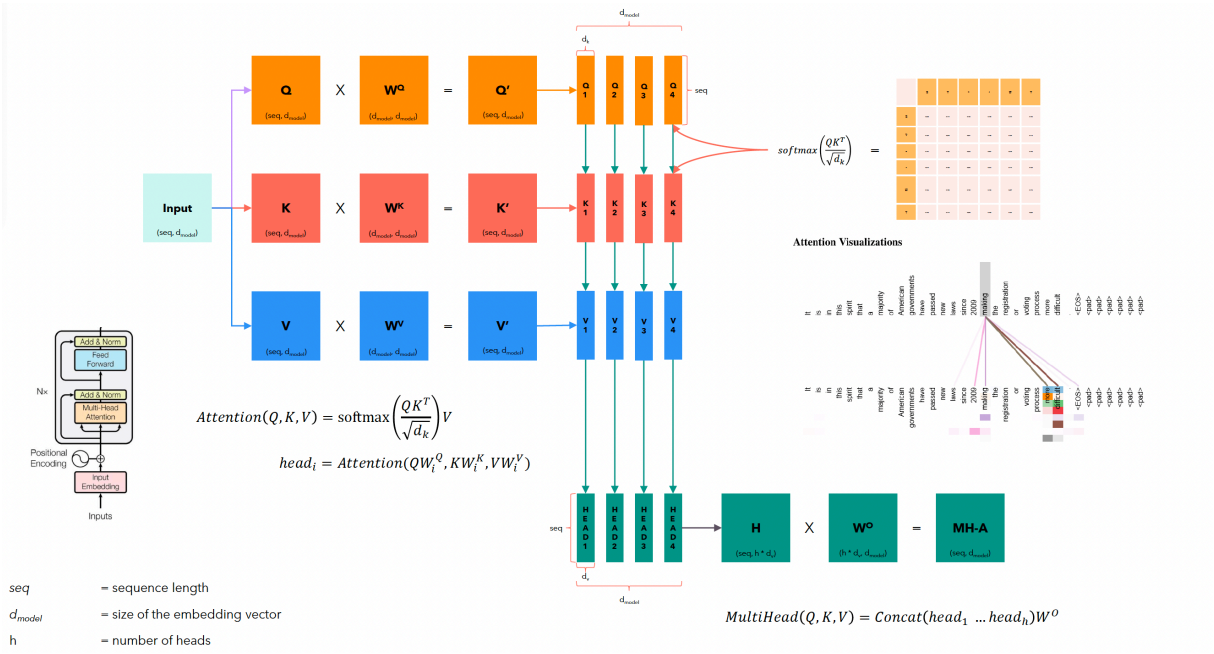

Figure 3. Q, K and V matrices of dimension sequence length $\times$ $d_{dimension}$ multiplied with the parameter matrices W^Q, W^K and W^V with shape $d_{dimension} \times d_{dimension}$ to form Q', K' and V' which are split into smaller matrices of size $d_{dimension}$ such that each head can observe the full sentence while focusing on a smaller portion of the embeddings for each word. The multi-head attention is calculated for the resultant matrices. [source: Jamil, U. (n.d.)]

The resultant matrix MH-A, instead of calculating the attention between Q', K' and V', splits them along the dimension d into smaller matrices and calculates the attention between them. This enables each head to watch the different aspects of the embedding of each word in the full sentence, to enable each head to watch different parts of speech of same word. For example, the same word can act as a noun, verb, pronoun or an adverb in different sentences, and each head would learn the context from each of these perspectives.

2.2.5. Add and Norm

Add and Norm are operations for maintain the flow of the gradients during backpropagation for efficient training of the model. In short they can be explained as :

1. **Residual Connections**: The "Add" part refers to the residual connections (He et al. 2016), which directly pass the input of a layer to the output of that same layer. This helps in preserving the original input signal and assists in training deep networks by mitigating the vanishing gradient problem.
2. **Normalization**: The "Norm" part stands for Layer Normalization (Ba et al. 2016), which stabilizes and accelerates the training by normalizing the inputs across the features. This ensures that the distribution of inputs remains consistent, which helps in improving convergence.

The output of Add and Norm are passed to the Feed Forward Network which is a fully connected layer and Add and Norm is applied again to generate the output of the Encoder.

2.2.6. Decoder

The output of the encoder is passed to the decoder in the form of Keys (K)[Blue arrow] and Values (V) [Green Arrow], and the Query (Q) [Red Arrow] is passed from the Masked Multi-Head Attention of the Decoder as shown in Figure 2 to produce the Q, K, and V values to the passed to the Multi-Head Attention of the Decoder. The Masked Multi-Head Attention makes the model Causal, which means that the model would not be allowed to see the future words. This process entails multiplying all values following the current token by -∞. When softmax is applied, these values are transformed into 0s in the resulting matrix.

Proceeding this, the output is passed to the Add and Norm which is then passed to the Feed Forward Network (Fully connected layer) and the output undergoes another Add and Norm and finally to the Linear Layer.

The LLMs considered for the study here are Llama-2 (7B) and Mistral (7B).

2.3. Large Language Models

2.3.1. *Llama-2 (7B)*

Llama-2 (7B) is an LLM from a collection of pretrained and fine-tuned large language models (LLMs) trained with 7 billion parameters released by Meta in July 2023. Figure 4 illustrates that the model performs on par with a few of the closed-source models of the time such as Vicuna, Falcon, Palm etc. on the basis of evaluations conducted by humans. This evaluation was conducted by humans on a model that accepts approximately 4000 prompts composing both single and multi-turn prompts, and vocabulary size of 32k tokens. Also note that the Confidence Interval for this human evaluation is 95% for this model. This indicates the level of confidence in the interval estimate. A 95% confidence interval means that if we were to repeat the evaluation multiple times and calculate the confidence interval each time, we would expect the true parameter value to fall within these intervals 95% of the time. The MMLU score for Llama-2 (7B) is 45.3. The Massive Multitask Language Understanding (MMLU) test covers 57 tasks including elementary mathematics, US history, computer science, law, and more. Hence the model must possess extensive world knowledge and problem-solving ability.

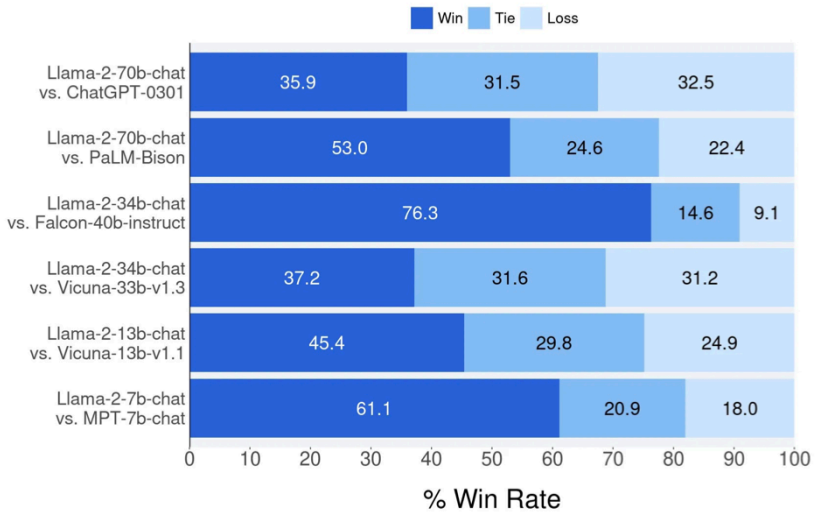

Figure 4. Helpfulness human evaluation results for Llama-2-Chat compared to other open-source and closed-source models The 95% confidence interval for this evaluation ranges from 1% to 2%, indicating a high level of precision and confidence in the estimated parameter. [source: (Touvron et al. *2023)].*

Llama-2 is trained on a variety of open source data, and was trained on data by increasing the size of the pretraining corpus by 40% and twice the context length of Llama 1 which was 2k (Touvron *et al.* 2023). One of the key aspects of Llama-2 over Llama 1 is that it adopts

grouped-query attention (GQA) for 34B and 70B models (Ainslie *et al.* 2023) mechanism which is an extension of multi-query and multi-head attention that achieves high-quality results at comparable speed to that of the multi-query attention. The model was trained using Root Mean Square Normalization (RMSN) and SwiGLU activation function and rotary positional embeddings, with AdamW optimizer and cosine learning rate schedule with a warmup of 2000 steps and reduce the learning (LR) rate down to 10 percent of the peak LR with a weight decay rate of 0.1 and gradient clipping of 1.0. The model training involved the use of Reinforcement Learning with Human Feedback (RLHF) and Supervised Fine Tuning (SFT). After pre-training the model did not show any signs of saturation. Reward modelling is used which is a form of Reinforcement Learning with Human Feedback that lets the model know about the human preference to the responses. Helpfulness and Safety are two separate rewards models that they train for, and has a trade-off between them (Bai *et al.* 2022).

2.3.1.1. Grouped-Query Attention (GQA)

Since memory access and memory transfers are expensive in terms of the computation time, In the study presented by Shazeer (2019) the number of arithmetic operations performed for a vanilla batched multi-head attention is in the order of $O(bnd^2)$, where b is the batch size, n is the sequence length, and d is the size of the embedding vector and the overall memory required for the the operations is provided by sum of all tensors involved in the calculations (including the derived ones) is $O(bnd + bhn^2 + d^2)$. The ratio of total number of times memory is used to the number of arithmetic operations which can be derived from the above is

$$O(\frac{1}{k} + \frac{1}{bn}) \tag{2.6}$$

Here the ratio would be much smaller than 1. Hence the number of memory access is much smaller than the number of arithmetic operations. Hence in vanilla batched multi-head attention the bottleneck is the number of arithmetic operations.

With the addition of KV Cache, which is described in section 2.3.2.2 in detail which helps in reducing the number of operations performed, the total number of memory access required would be reduced to $O(bnd^2 + nd^2)$, thereby changing the ratio of the total number of times the memory is accessed to the number of arithmetic operations to

$$O(\frac{n}{d} + \frac{1}{b}) \tag{2.7}$$

When n is close to d ie. when the sequence length is close to the size of the embedding vector, or batch size is 1, then the ratio becomes 1 and the memory access would be a bottleneck here. To avoid this problem, the dimension of the embedding vector must be much bigger than the sequence length. This means reducing the sequence length which means reducing the size of the text passed to the LLM. As this is not a viable solution Multi-query Attention offers an alternative.

Multi-query Attention enables the model to efficiently handle longer sequences by reducing the computational overhead associated with processing each token, thus allowing for faster and more scalable performance without compromising the quality of the results. Multi-query attention removed the h dimension from the Key (K) and Value (V) matrices while retaining it for Q, which means all different query heads will share the same keys and values. When using multi-query attention, the number of memory operations becomes $O(bnd + bn^2k + nd^2)$. This makes the ratio between the total number of memory access and the number of arithmetic operations as

$$O(\frac{1}{d} + \frac{n}{dh} + \frac{1}{b}) \tag{2.8}$$

This reduces the term $\frac{n}{d}$ from the equation (*2.7*) by a factor of h. By removing the heads from the K and V the model would have less parameters and degrees of freedom and complexity.

Table 1: WMT14 EN-DE Results.

Attention Type	h	d_k, d_v	d_{ff}	ln(PPL) (dev)	BLEU (dev)	BLEU (test) beam 1 / 4
multi-head	8	128	4096	**1.424**	**26.7**	27.7 / 28.4
multi-query	8	128	5440	1.439	26.5	27.5 / **28.5**
multi-head local	8	128	4096	1.427	26.6	27.5 / 28.3
multi-query local	8	128	5440	1.437	26.5	27.6 / 28.2
multi-head	1	128	6784	1.518	25.8	
multi-head	2	64	6784	1.480	26.2	26.8 / 27.9
multi-head	4	32	6784	1.488	26.1	
multi-head	8	16	6784	1.513	25.8	

Figure 5. BLEU score on an English to German Translation task [source: (Shazeer 2019)]

The BLEU score of multi head and multi-query attention models are compared in the Figure 5 where the reduction in BLEU score can be observed for the multi-query due to the removal of the h from K and V. However, as illustrated in the paper by Shazeer (2019), the performance gain is significant when compared to the minimal degradation of the model's response quality.

Attention Type	Training	Inference enc. + dec.	Beam-4 Search enc. + dec.
multi-head	13.2	1.7 + 46	2.0 + 203
multi-query	**13.0**	1.5 + 3.8	1.6 + 32
multi-head local	13.2	1.7 + 23	1.9 + 47
multi-query local	**13.0**	**1.5 + 3.3**	**1.6 + 16**

Figure 6. Improvement in Inference time for multi-query attention when compared to multi-head attention [source: (Shazeer 2019)]

This trade-off is further highlighted in Figure 6 which shows the remarkable improvement in the inference time from 47.7 (1.7 + 46) to 5.3 (1.5 + 3.8) microseconds for multi-query when compared to multi-head attention.

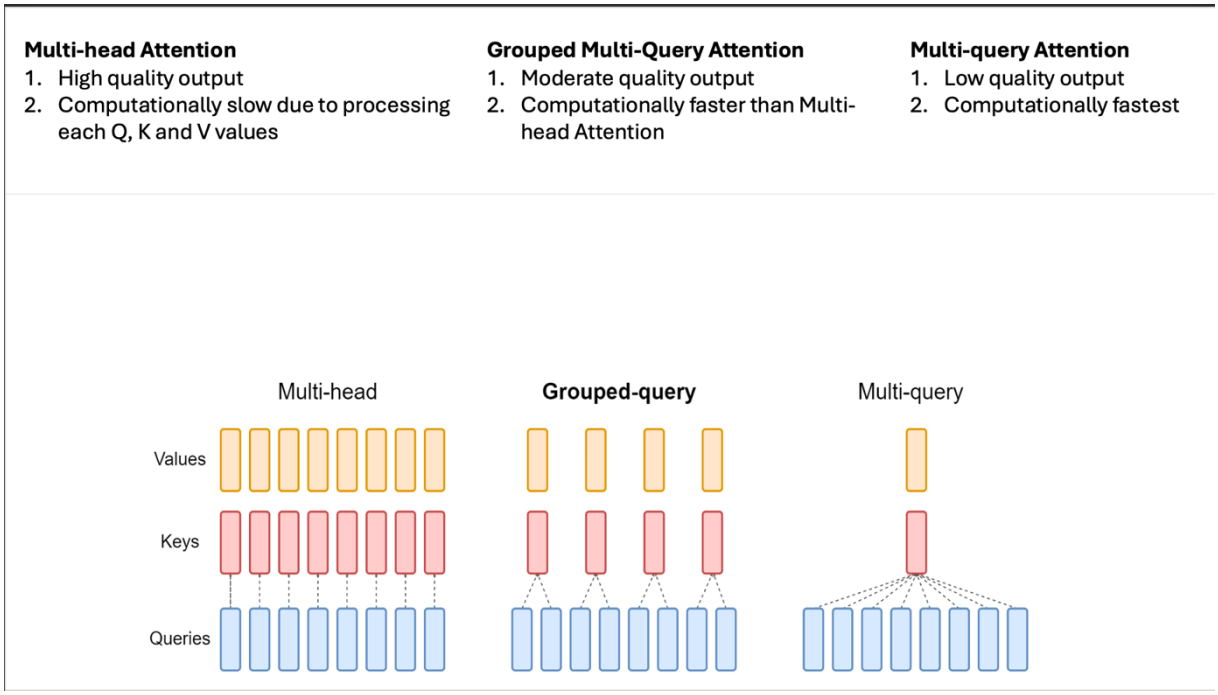

Figure 7. Architecture of grouped-query method: In multi-head attention, each of the H query heads has its own corresponding key and value heads. Multi-query attention, on the other hand, uses a single set of key and value heads that are shared across all query heads. The grouped-query attention method serves as a middle ground, where a single set of key and value heads is shared within each group of query heads, effectively bridging the gap between multi-head and multi-query attention. [source: (Ainslie et al. 2023)]

Therefore, with multi-head attention it has multiple heads for Queries (Q) and one head for K and V. As demonstrated in Figure 7, with Grouped Multi-Query Attention the queries are divided into groups and for each group, one head of K and V are assigned. This is a good compromise between multi-head where we have one-to-one correspondence and the multi-query where there is n-to-one correspondence between number of heads for queries. With Grouped-Query Attention (GQA), it has multiple heads for the Keys and Values but they are less in number when compared to the heads of the Queries. Using GQA, we benefit from the improvement in performance of the model with respect to computation because of the reduction in number of heads of K and V to an extend and not sacrificing the quality of the results as much as multi-query attention.

2.3.2. Mistral (7B)

The second open source LLM that discussed here is Mistral-7B which is trained with 7 billion parameters. This model also uses Grouped-Query Attention (GQA) as described in 2.3.1.1 along with Sliding Window Attention (SWA). Like Llama-2, Mistral also uses GQA for faster inference and reducing memory requirements while decoding which provides higher throughput due to its adoption of higher batch sizes. Unlike Llama-2, the attention mechanism used here is SWA and Rolling Buffer KV-cache, and in the Feed Forward layer instead of using the ReLU like Vanilla Transformer or SwiGLU as in Llama-2, it uses SILU. The block containing these (Transformer Block) are repeated 32 times and the final output is sent to the RMSN and then to a linear layer and finally to the Softmax layer.

These LLMs struggle with long sequences of text or data. This is due to the quadratic growth in computational and memory requirements with respect to the sequence length when using the attention mechanism. For example, processing a sequence of length n requires $O\ (n^2)$ operations and memory. SWA helps eliminate this by handling long sequences thereby reducing the computational cost, which allows the model to be applied to tasks with longer contexts without facing performance bottlenecks. Mistral is also based on the transformer architecture. Similar to Llama-2, the model has a vocabulary size of 32k (Jiang *et al.* 2023).

2.3.2.1. Sliding Window Attention

Sliding Window helps the token under process to not pay attention to tokens outside its local context. The sliding window size would determine the range of attention paid to other tokens.

This means that if the sliding window is 3, then tokens that are more that 3 tokens away from the current token would be converted to $-\infty$ to change the softmax result to 0. This enables in reduction of number of dot-products, resulting in faster inference and reduction in training time. The multiplication with the V in the attention formula would give us a matrix of the same shape as the query matrix in which each token is represented by an embedding of size 4096, where each embedding has information about other tokens according to the mask.

Since Mistral has 32 layers of encoders, the output of the previous layer is forwarded as input to the next layer. Since in each layer there is a multiplication with the V value matrix, the information (number of tokens) is accumulated for each dot product, and this enables capturing information about the tokens outside the sliding window but without the explicit multiplication. This can be seen as multiplying two vectors that already contains information about 2 distinct tokens, with another vector thereby, introducing another token into the mix.

Hence with Sliding Window attention, the 2 tokens are not connected directly with each other, rather applying multiple layers of encoders results in accumulated knowledge being passed down the layer, even if the sentence is long.

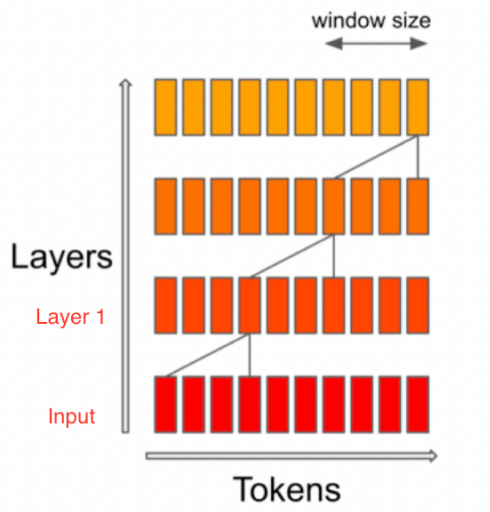

Figure 8. Effective context length where the first 4 tokens from the input is passed to the 4th token of the Layer 1 which passed to the 7th token of layer 2. Even though the 1st token is not directly related to the 7th token, its related via the layer 1 which contains the information about the 1st token. [source: (Jiang et al. 2023)].

The Figure 8 illustrates this concept with a sliding window size of 4. The 4th token of the Layer 1 depends on itself, and 3 other tokens before it. This information is passed down to the second layer (Layer 2), which takes the 4th token that contains the information about the first 4 tokens and the next 3 tokens. Hence the token 7 is not directly related to the 1st token,

however, it is indirectly related to the first one due to accumulation of information in the 4th token, which is consumed by the second layer.

Language models are trained using Next Token Prediction task, which is done via training the model with an input with [SOS] (start of sentence) followed by the sentence and a target with the sentence followed by [EOS] (end of sentence). The input is passed through the transformer model and produces an output sequence with the same length as the input sequence (provided the model is using self-attention), but the embedding is modified such that each token captures information about other tokens (Vaswani *et al.* 2023) and the loss is backpropagated to train the model.

Inference is done in multiple steps contrary to the training which happens in a single step. The inference involves providing the [SOS] token as input and the output obtained is passed as input to the model by appending it to the [SOS] input at the next time step, and this process carries on.

We only care about the most recent token that the model produces at each stage of the inference because we already possess the earlier ones. But in order for the model to choose which token to output, it needs access to every prior token because that's what makes up its context, or the "prompt". This can be computationally expensive

In the self-attention mechanism, we multiply the query Q, which are a list of tokens, where each token is an embedding of size 4096, with the transpose key K^T and the resultant matrix is multiplied by V which produces the Attention Matrix. This matrix is forwarded to the linear layer and then to the softmax layer, to predict the output. Starting with a single token the output of the softmax layer is calculated and appended to the query Q and the process repeats until [EOS] is found. This iterative process would be compute intensive and the result would not need a lot of these calculated values as we apply the Causal mask to remove the attention towards future words. And if we need to calculate say the 4th token then we do not need to iteratively do these calculations for each token starting from the beginning as we only need the 4th token.

2.3.2.2. The KV Cache

The KV cache helps solve this by not appending the output of the Attention mechanism to the Query Q, but only to the key K and values V, and use the output of the Attention mechanism

as input to the next step and not append it onto the earlier input. Since at each step we are keeping the cache of K and V and not the query, it is called as KV cache.

The KV cache along with SWA further increases the performance by not keeping all the possible K and V values in the cache and limit it to the size of the sentence window due to the same reason as why we would not want to store the KV values for the Causal masked values, as the model does not need to pay attention to the tokens outside the window. This approach of limiting the size of the KV cache to the size W of the window is known as **Rolling Buffer Cache**. The approach followed is to fill the cache until window size W and when full, the oldest value is replaced in the cache and the rest of the values are rotated in order to maintain consistency with the input.

2.3.3. Llama-3 (8B)

The Llama-3 is intended to support multilinguality, reasoning, code and tool usage. Compared to Llama and Llama-2 (Touvron *et al.* 2023) it was trained on 15T multilingual tokens that are carefully curated and pre-processed and has passed through pipelines for pre-training. In order to maintain training stability, the model is using standard dense Transformer architecture and supervised finetuning (SFT), rejection sampling (RS), and direct preference optimization (DPO) (Rafailov *et al.* 2023). The MMLU score for Llama-3 (8B) is 68.4.

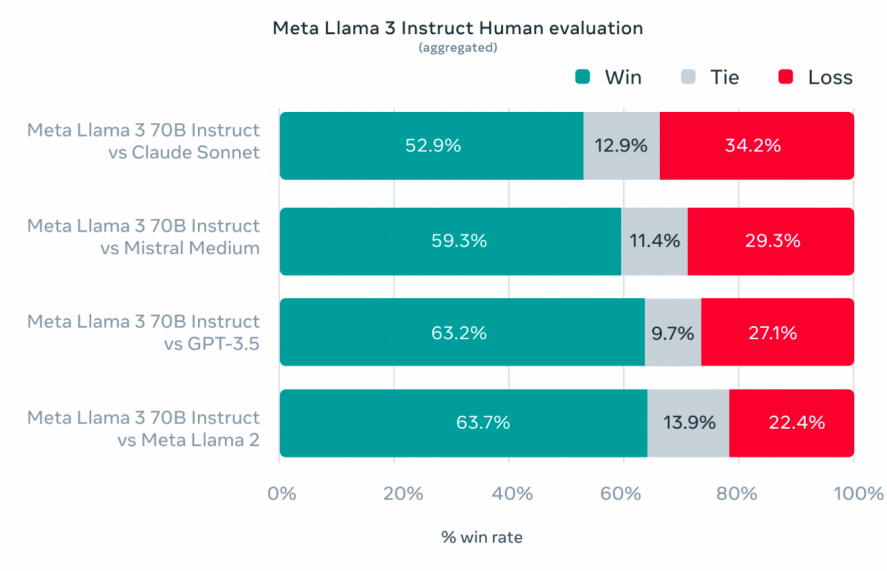

Figure 9. Helpfulness human evaluation results for Llama-3 compared to other open-source and closed-source models. [source: (Dubey et al. *2024)].*

The Figure 9 shows the human evaluation which contains 1,800 prompts that cover 12 key use cases: asking for advice, brainstorming, classification, closed question answering, coding, creative writing, extraction, inhabiting a character/persona, open question answering, reasoning, rewriting, and summarization. To prevent accidental overfitting of our models on this evaluation set, even our own modeling teams do not have access to it. The chart below shows aggregated results of our human evaluations across of these categories.

Working with LLMs means that we would need to set up prompts to instruct the LLM to take an action and produce the desired output based on the data or input queries we pass to it. Good prompting is critical in guiding the model to retrieve the correct documents and produce correct output.

Prompt engineering refers to the process of creating the prompts to optimize the output of LLMs. It enables direct interaction with the LLM using plain language, whereby clear communication of the intent and the context is mentioned with the roles that the LLM should assume along with the instruction for each role. There are various types of prompting techniques available.

2.3.4. Types of Prompting

2.3.4.1. Direct prompting (Zero-shot prompting)

The model is given a prompt without any examples or prior context. It relies solely on its pre-trained knowledge to generate a response. For example, Asking the model to "Explain the concept of machine learning.". This can also be done using a question or providing a role. For instance, “You are an expert in Artificial Intelligence and Machine Learning, and I need you to explain the concept of Machine Learning” (Schick and Schütze 2021).

2.3.4.2. Few-shot prompting

The model is provided a prompt along with a few examples of the output expected and the input provided to help it comprehend the context or format required for the response. For instance, providing the examples of question-answer pairs then asking a few questions in the similar format. (Brown *et al.* 2020)

2.3.4.3. One-shot prompting

The model is given a single example along with the prompt to provide a correct output. This is helpful in case of machine translation where interpreting one example of a translation can help it understand the context of the output expected.

2.3.4.4. Chain-of-Thought prompting

The model is prompted to output a sequence of intermediate steps which helps it reason before arriving at a final response. This helps it in complex reasoning tasks. In the work done by Wei et al. 2023, the example illustrated includes a mathematical problem which allows large language models to tackle complex arithmetic, commonsense, and symbolic reasoning tasks (Wei *et al.* 2023).

This sub-section explored the various LLMs used in the study and how they were traind and the key differences between them. The next sub-section investigates the concept of Knowledge Graphs and their need in the recommender system.

2.4. Knowledge Graph (KG)

A Knowledge Graph (KG) is a heterogeneous network where nodes represent entities and edges indicate relationships between these entities. By mapping items and their attributes into the KG, a relationship between them can be understood. Additionally, incorporating users and their associated information into the KG enables a more accurate capture of the relationships between users and items, as well as user preferences (Guo *et al.* 2022). A KG can be thought of as an approach to represent a broad dataset from numerous domains with the help of a graph structure (Ehrlinger, L. & Wöß, W.). There exist 2 types of KG namely:

1. **Item Knowledge Graph** : In this KG, item and item attributes assumes the role of nodes and edges can represent either attribute-level relationships of items, such as brand or category, or user-related relationships, such as "co-view" and "co-buy".

   For airline industry, the nodes would be users (travelers), flights, and related entities such as destinations, departure time, and seat class, pricing etc. and the edges in this graph represents relationships between these attributes, like connecting flights within the same category or with similar characteristics. It may also include user-related edges such as "co-view" (flights that are searched together) or "co-buy" (flights booked together in the past).

2. **User-Item Knowledge Graph** : In the User-Item Knowledge Graph, the nodes consist of users, items and their related entities. While it includes item-related relationships similar to those in the Item KG, it also incorporates direct relationships between users and items such as "buy" (flights the user has booked), "click" (flights the user has searched), and "mention" (flights the user has reviewed or discussed). When a flight is delayed or cancelled, the User-Item KG can offer personalized alternative recommendations by considering the user's past interactions and preferences.

The type of KG used here is the User-Item KG as user information and the flight related information are needed to provide recommendations. The knowledge graph $G = (V, E)$ is embedded into a low dimensional space (Cai *et al.* 2018) for easy representation is also known as Knowledge Graph Embedding (KGE). Once the embedding is completed each of the components of the graph, including the entity and the relation can be represented as d-dimensional vector, which preserves the side information of the graph. This can be quantified by semantic meaning or high-order proximity in the graph (Guo *et al.* 2022). These graphs can be evolved as and when required and can be tailored for specific domains. This enables better explainability as well.

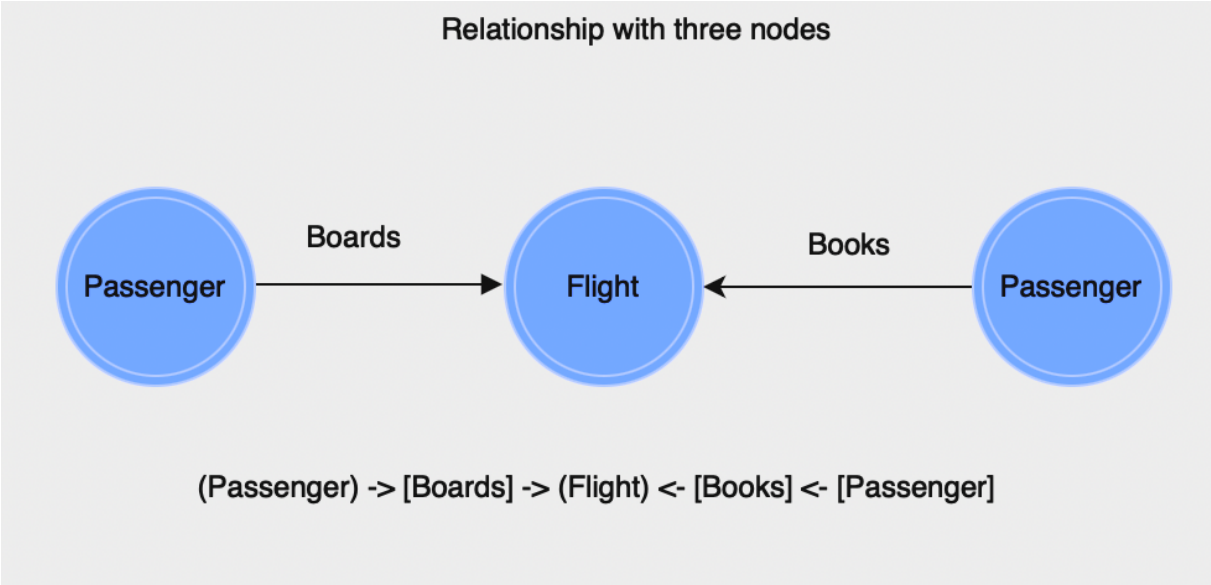

Figure 10. Example of Knowledge Graph with three nodes and two relationships between them. Passenger entity can have multiple relationships with the flight entity such as Boards and Books.

The Figure 10 shows an example of the KG with three nodes where an entity Passenger is connected to the entity Flight with a relationship called Boards and another entity passenger is connected to the flight by the relationship called Books. Here each of the entities can have

their own properties such as Flight can have attributes such as flight number, departure airport, arrival airport etc. The relationship Passenger Boards Flight can be represented as (Passenger) -> [Boards] -> (Flight) in textual format, where the nodes are described in parenthesis and the relationships are denoted in square braces.

While this may seem like a good solution, the limitation of this approach involves the application of a domain expert to identify the relationships and forming the connections (Peng *et al.* 2023). There have been approaches to form KG by using LLMs to extract keywords from a textual knowledge base and using graph Laplace learning to determine the wight of the relationship between each pair of keywords (Chen and Bertozzi 2023). LLMGraphTransformer is one way to accomplish this, where in we can extract information from textual unstructured data using an LLM to extract structured graph information. This information is then stored into a graph database which can then be used to retrieve information. Even though this approach is easy to use, this involved addition of LLMs that can hallucinate and provide inconsistent graph structure, hence the approach adopted here is by manually creating the KGs.

2.4.1. Why KG

The connection between large language models (LLMs) and knowledge bases is established through semantic similarity search traditionally. However, this falls short of capturing complex relational dynamics that are inherent in the data. This is illustrated below by the work from (Chen and Bertozzi 2023)

For instance, imagine a basic knowledge base with text blocks that describe a person named Alex's day in his or her life and related details. In the morning, Alex leaves his house, gets a coffee at Cafe A, and rides the bus to Company B to work. Many discrete bits of information, including the chats Alex had with the cafe's barista, the specifics of his coffee order, the talks he had on the bus, the conversations he had at work, and so on, are included across the knowledge base.

Here, what's interesting to consider is a model's response to the query, "Was it raining this morning when Alex left his home?", assuming that neither a direct response to this query nor any information regarding the weather in the knowledge base exist. Two bits of information about the weather are present in the knowledge base:

1. Concerning Cafe A: "A large number of people were conversing and sipping coffee in the plaza outside Cafe A."
2. Concerning Company B: "Today was a hive of activity at the car wash downstairs at Company B."

These two brief excerpts imply that it wasn't raining.

The information obtained from the knowledge base by semantic similarity vector search would only be about Alex because the inquiry is essentially about Alex and the weather (as there is no direct information on the weather). His daily movements would be the main focus of the search results. Even with more search terms, it would mainly return irrelevant information, such as his coffee order and conversations. Nevertheless, there are no cues in these information to deduce the weather for the day.

Throughout the KG generation procedure, we extract terms like Alex, Company B, and Cafe A. The input question is used in the first step of the hybrid search to retrieve the keyword Alex. Next, as similar keywords, the adjacent search finds Company B and Cafe A. Afterwards, text blocks are searched for using these keywords, which leads to the discovery of implicit weather-related data. This sample demonstrates how useful these approaches can be. Cross topic links can be absent from semantic similarity alone. It frequently pulls up a lot of small details that are relevant to the subject at hand. The depth and variety of the data received when searching with the KG approach is amplified.

Without a knowledge graph, the input can be converted into a generic feature vector, which can be fed to any supervised learning model to predict the recommendation. However, the issue here is that each interaction with the item would be seen as an independent instance of the data. However, KGs allows to extract attribute information, thereby overcoming this limitation.

The second part of the research deals with retrieval augmented generation and its optimization which is discussed in the following sections. This involves exploring how integrating retrieval mechanisms can enhance the generation process, improving the quality and relevance of the outputs. Additionally, various optimization strategies are examined to further boost the efficiency and effectiveness of RAG in different application scenarios.

2.5. Retrieval Augmented Generation (RAG)

Retrieval Augmented Generation (RAG) models were introduced by Meta AI University College London and New York University (Lewis *et al.* 2021). The reason for introducing RAGs was due to the fact that large pre-trained models like LLMs have knowledge on the trained parameters and have exhibited state-of-the-art results when fine-tuned on specific tasks. However, they lack the ability to predict something outside their knowledge domain seemed out of scope. Hence RAGs were introduced that can combine the pre-trained parametric and non-parametric memory for language generation. The parametric memory that they used is a sequence-to-sequence model and the non-parametric memory was a vector index of Wikipedia accessed using a neural retriever.

2.5.1. Working of RAG

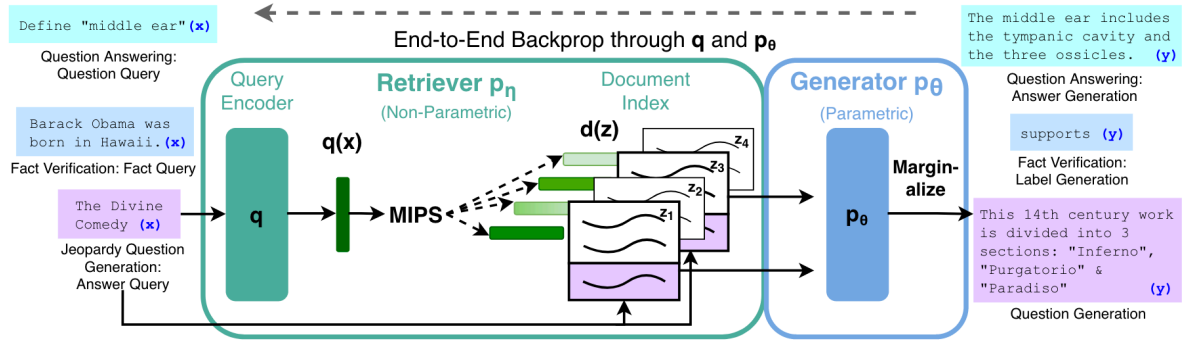

Figure 11. Working of Retrieval Augmented Generation (RAG) [source: (Lewis et al. *2021)].*

The Figure 11 shows the working of the RAG proposed by (Lewis *et al.* 2021), where they combine pre-trained retriever, which is a combination of Query Encoder and Document Index with a pretrained seq2seq model, denoted as Generator in the figure. When a query xx is received, the system uses MIPS to find the top-K most relevant documents z_i . These documents act as latent variables that provide context for making the final prediction y. The final output is determined by combining (or marginalizing over) the predictions from multiple seq2seq models, each conditioned on a different relevant document z_i . This approach helps leverage the information from the most relevant documents to make more accurate and informed predictions.

In simple words, it involves encoding all the text information from the specific domain which we would like the RAG to be answering on, and convert it to vectors and store the result into a database or a vector store. In order to understand the context and intent of the user query,

the user query is also converted to the vector space using the same encoder. Subsequently, prompt is encoded the using the same model used to encode the text. The prompt helps in providing extra context and content for the model to answer the question. For instance, (Li and Liang 2021) reported “in in-context learning, (Brown *et al.* 2020) prepend a natural language task instruction (e.g., TL;DR for summarization) and a few examples to the task input, and then generate the task output from the LM”. This is done so that the documents retrieved are in the close vicinity of the query in the vector space. The retrieved text with this information should contain the target ed documents, ideally. The next step is to enhance the language model's prompt with this information after identifying the relevant documents. This involves integrating new information while preserving the original context and query. The language model can now produce responses based on correct and current information, along with rich contextual information.

Regular updates can be made to the external data sources in order to keep the RAG system functioning well. This guarantees that throughout time, the system's responses will still be pertinent. Depending on the needs of the application and the type of data, the update procedure may be carried out automatically or in recurring batches. This RAG feature emphasizes how crucial data freshness and dynamism are to producing correct and insightful answers.

The RAG model has two parts: one that does the retrieval of the closest documents (retrieval) from the database or a corpus of documents and the second part that passes this information to the LLM to generate the textual output (generation), which involves predicting the next token given the prompt.

The retrieval models can be implemented vector embeddings and vector search, however, some of the less common methods for implementing are using BM25 (Best Match 25) and TF-IDF (Term Frequency—Inverse Document Frequency).

Generative models take over when the retrieval model has located the relevant data. These models combine the knowledge they have retrieved to create language that makes sense and is relevant to the context. Generative models, which are typically based on LLMs are capable of producing text that is coherent, grammatically accurate, and consistent with the original prompt or inquiry. By adding a narrative structure to the data chosen by the retrieval algorithms, they make the material more comprehensible, human like and useful.

The data is usually partitioned into smaller "chunks" or segments before the retrieval model can go through it. By chunking the data, you can make sure that the system can quickly retrieve relevant content and scan through the data efficiently. The speed and accuracy of the model can be significantly increased by using effective chunking strategies. A text can be divided into chapters or sections, paragraphs, sentences, or even just "chunks of words".

2.5.2. Embedding the Chunks

Once the data is chunked they are converted to a format that can be stored into a database for retrieval. In order to represent related content and feed the generator and the retriever, which is at the heart of the RAG system, depends on an effective embedding model such as those presented by (Chen *et al.* 2024). Furthermore, to improve performance in specific areas, domain-specific or task-related data can be used to fine-tune embedding models with great expressive power. Dense retrieval is a popular type of embedding-based information retrieval (IR) in which pertinent query replies are retrieved based on the embedding similarity.

Embeddings involve creating representations for multimodal and multimedia components, including calculations, tables, and figures. Usually, chunk embeddings are only made once, either when a new document is indexed or during system development. The efficiency of a RAG system is greatly impacted by query preprocessing, especially when managing negative or unclear inquiries, as reported by (Barnett *et al.* 2024). These embeddings are created via Sentence Transformers models.

In simple words, embeddings are a type of representation of words with vectors in vector space, which groups together words with similar meanings using their distance from each other. The concept of embedding is critical when working with large language models as we use this same concept in the **recommendation system** as well. There exists two different types of embeddings namely (Yang and Mao 2017):

1. **Context independent embeddings or static embeddings**: These embeddings allocate different vector representations for each of the words independent of contextual differences. For instance, the word bark in the sentences “Don’t let the dogs bark” and “The bark of this tree is strong” would have the same vector representing it regardless of the difference in the context of the word. By collecting various meanings in a fixed representation, this method produces a comprehensive map of word vectors.
   Some of the common embeddings that are frequency based are :

Bag of Words : where it creates a dictionary of the frequently occurring words in the sentences and use the frequency value to encode the sentences.

TF-IDF : where it identifies features from sentences for the most significant words in the sentence. When considering a document in a corpus, two things are taken into consideration, the first one being how important the word is in the document

$$\text{TF} = \frac{Number\ of\ occurrences\ of\ the\ word\ in\ the\ document}{Number\ of\ words\ in\ the\ document}$$

And the second, in the entire corpus how significant is the word

IDF = -log (ratio of documents that include the word)

Word2Vec : A model proposed by (Mikolov *et al.* 2013), a two-layer neural network is used in Word2Vec to learn word embeddings; however, during training, the neural network unintentionally records linguistic contexts. The main goal of the algorithm produces the embeddings as a byproduct, demonstrating the effectiveness of this method. Word2Vec offers two unique model architectures—continuous skip-gram and CBOW—that allow for customization.

Continuous Bag-of-Words (CBOW) : Selects the word of the moment from a list of words that are present in the context and highlights how context words work together to forecast the target word.

Continuous Skip-Gram : Predicts the window of context words that surrounds the current word and focuses on the target word's capacity for prediction in producing context words.

Global Vectors (GloVe) : for Word Representation, stands out due to its use of aggregated global word-word co-occurrence statistics from a corpus during training. This approach proposed by Pennington et al. 2014, not only captures semantic relationships but also uncovers interesting linear patterns within the word vector space, enhancing our understanding of word embeddings.

2. **Context dependent embeddings** : these embeddings produce different vector representation for same words depending on their context. In the sentences “Don’t let the dogs bark” and “The bark of this tree is strong”, the word bark would be encoded differently as they have different contextual meanings.

   The popular RNN based context dependent embeddings are :

ELMO : Embeddings from Language Model, is a neural language model that combines two BiLSTM layers and a character-based encoding layer to learn contextualized word representations.

CoVe (Contextualized Word Vectors) : Contextualizes word vectors by using a deep LSTM encoder using an attentional sequence-to-sequence model learned for machine translation.

To understand the sentence transformers and their usage, it is important to understand what BERT is and how the creation of SBERT mitigates a lot of the issues that BERT possessed. The transformer based embeddings are :

BERT (Bidirectional Encoder Representations from Transformers) : Developed by Google AI (Devlin *et al.* 2019), a Transformer-based language representation model that was trained on a sizable cross-domain corpus. The pre-training step involves two unsupervised tasks namely Masked Language Modeling (MLM) and Next Sentence Prediction (NSP). The MLM randomly masks some of the tokens in a sentence and then trains the model to predict the masked tokens, based on the context provided by the surrounding tokens from both directions. For example, in the sentence "The cat sat on the [MASK]", BERT will consider the context from both "The cat sat on the" and the words following the mask to predict the masked word, hence exhibiting the bidirectional nature of BERT, with the transformer architecture which provides the self-attention mechanism.

The Next Sentence Prediction (NSP): This secondary task helps BERT understand sentence relationships. The model is trained to predict whether a given pair of sentences is consecutive in the original text, further enhancing its contextual understanding.

XLM (Cross-lingual Language Model) : is a transformer that has been pre-trained with next token prediction, translation, and a BERT-like masked language modeling target.

RoBERTa (Robustly Optimized BERT Pretraining Approach) : is an approach that expands upon BERT by adjusting important hyperparameters, doing away with the next-sentence pretraining goal, and training with significantly bigger mini-batches and learning rates.

ALBERT (A Lite BERT for Self-supervised Learning of Language Representations) : It offers methods for reducing parameters, which reduce memory usage and speed up BERT training.

2.5.3. Sentence Transformer

Sentence Transformers are a framework used for the computing the sentence-level embedding. These are widely used to create embeddings when working with LLMs. In order to identify sentences with comparable meanings, the generated embeddings are compared using the cosine similarity. With the Sentence-Transformers framework, a large library of pre-trained models is available, which may be adjusted based on the use case (Ramnarain-Seetohul *et al.* 2022). The problem with BERT was that transformers work with token-level embeddings or word-level embeddings and not sentence-level embeddings.

Before introduction of the Sentence transformers, the approach towards calculating sentence similarity with close accuracy using BERT was to use a **cross-encoder structure**. This involved passing two sentences to BERT and add a classification head to top of BERT and use this to calculate the similarity. The cross-encoder architecture consists of a model which consumes sentences A and B. A [SEP] token is used to distinguish between the two, which are processed in the same order. This is followed by a feedforward NN classifier that produces a similarity score. The work done by Chiu and Shinzato 2022 illustrates the two encoder mechanisms that are commonly practiced.

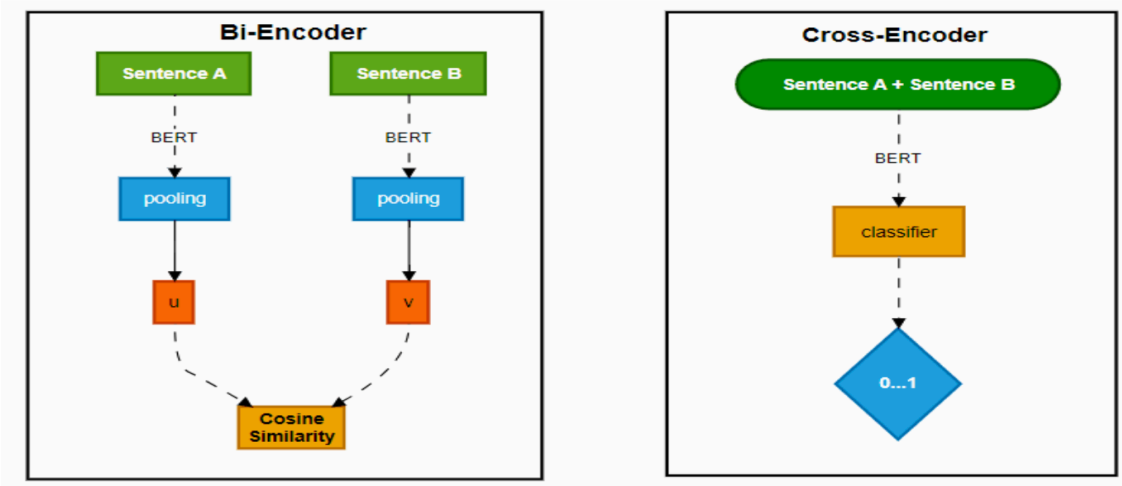

Figure 12. Bi-encoder and Cross-encoder architecture [source: (Singh 2024)].

The two encoder types are shown in Figure 12. These are employed in the later stages to optimize the implementation of RAG analysis.

Bi-Encoder: This encoder produces two unique embeddings, called u and v, by processing sentences A and B independently. A cosine similarity method is usually applied later to compare these embeddings independently, in order to establish sentence similarity. This is done for search query and the information in the vector space independently and are compared with the cosine similarity. The models that compute **dense vector embeddings for data** (which later can be used for search queries) are so-called *Bi-Encoder models.* Calculating vectors up-front coupled vector indexing (a method used to organize the embedding when stored) can lead to faster retrieval of data. As investigated by Kukreja *et al.* 2023) One of the most common vector indexing strategy is Inverted File (IVF) which involves clustering the information into different groups using techniques like K means clustering, wherein each vector of the database is assigned a specific cluster. When a user produces a new query, the system identifies nearest and most similar clusters and searches for the specific document withing those clusters, rather than traversing the whole database. There are other variants of IVF such as IVFFLAT, IVFPQ, IVFSQ. Bi-Encoders are commonly used in tasks where document retrieval or ranking is the primary goal, such as search engines or recommendation systems

Cross-Encoder: This encoder uses the Transformer network to process sentences collectively. A similarity score between 0 and 1 is the result. A single encoder processes both the query and the document in a cross-encoder model. This indicates that the model creates a combined representation using the query and the document as input. Like bi-encoders, cross-encoders are also trained to optimize the similarity between relevant query-document pairs. But because they handle the document and query concurrently, they capture interactions between the two. For every query-document pair, cross-encoders produces a single similarity score while taking the interaction between the query and document embeddings into consideration. The document with the highest score is the most relevant. But it isn't appropriate for processing individual sentences and doesn't provide separate sentence encodings. Cross-encoders are useful when capturing the interaction between the query and document is crucial, such as in tasks where understanding the context or relationship between the query and document is important.

Bi-Encoders are used when processing larger datasets and when inference must be done faster. When trying to capture complex relations between query and documents that are less

critical, Bi-Encoders are suitable. However, when trying to capture relations that are crucial for the task at hand, we use Cross-Encoders, as they are computationally expensive, and they provide better performance when comprehending context is important.

While cross-encoders provide greater accuracy with better similarity score, it is not scalable. For instance, if we want to do a similarity search through a dataset of 100k sentences, we would need to do the inference 100k times, and in order to cluster sentences we would need to compare all of the 100k sentences, which means we would need 100k (100k-1)/2=5B comparisons. One of the simplest ways to optimize this would be by averaging values across all token embeddings to create a sentence embedding, thereby if we have 512 tokens, we have 512 embeddings and obtain the average. This can considerably reduce the computation time and provide pre-computed sentence vectors that can be stored and then used whenever required. Nevertheless, the accuracy declined considerably.

Hence the introduction of SBERT by Reimers and Gurevych (2019), which outperformed the previous state-of-the-art (SOTA) models for all common semantic textual similarity (STS) tasks. Reimers and Gurevych, 2019 showed that for BERT finding the most similar sentence pair from 10K sentences took 65 hours. With SBERT, embeddings are created in approximately 5 seconds and compared with cosine similarity in about 0.01 seconds.
Some of the sentence transformers used in the study are :

1. **Sentence-transformers/all-MiniLM-L6-v2**

The project produces a trained sentence embedding model on very large sentence level datasets using a self-supervised contrastive learning objective. The model was pretrained on nreimers/MiniLM-L6-H384-uncased model and fine-tuned in on a 1B sentence pairs dataset. Contrastive learning objective: given a sentence from the pair, the model should predict which out of a set of randomly sampled other sentences, was actually paired with it in the dataset (‘sentence-transformers/all-MiniLM-L6-v2 · Hugging Face’ 2024).

The model is intended to be used as an encoder for sentences and short paragraphs. It generates a vector that contains the semantic information from an input text. Tasks involving sentence similarity, clustering, or information retrieval can all benefit from the usage of the sentence vector. Text input over 256 word segments is automatically terminated.

The batch size of 1024 (128 per TPU core) is used to train the model for 100k iterations. A warm-up learning rate of 500 is used. There is a 128 token maximum for the sequence length. A learning rate of 2e-5 is applied when using the AdamW optimizer.

2. **Int/float/multilingual-E5-small**

The work by (Wang, Yang, *et al.* 2024) provides a model that is initialized from microsoft/Multilingual-MiniLM-L12-H384 and is further trained on a blend of multilingual datasets. It supports 100 languages from xlm-roberta; however, performance may degrade for low-resource languages. This is a model that supports multilingual capabilities. The model is trained initially on contrastive pre-training and then later, on supervised fine-tuning. The dataset used for contrastive pre-training involves CC News, NLLB, Wikipedia, unsupervised SBERT data etc. and the supervised fine-tuning involves MS MARCO, NQ, Trivia QA, Quora etc. The average MRR@10 score for the model on Mr. TyDi dataset is 64.4 when compared to BM25 with a value of 33.3. Since the model was trained with a low temperature score 0.1 with InfoNCE contrastive loss it may result in higher cosine similarity scores. Since the order of the related document fetched from the database is more important, this behaviour can be overlooked.

3. **BAAI/bge-small-en-v1.5**

The model is referred to as bge. The bge is pre-trained using retromae (retromae provides a higher MRR@10 and Recall@1000 score for MS MARCO dataset and provides high-quality supervised retrieval performance in the corresponding scenario) and trained on large pairs data using contrastive learning. The model is designed to be used with fine-tuning. The model can be used for reranking which is different from typical embedding model, reranker uses question and document as input and directly output similarity instead of embedding. Cross-encoder will perform full-attention over the input pair, which is more accurate than embedding model (i.e., bi-encoder) but more time-consuming than embedding model. Therefore, it can be used to re-rank the top-k documents returned by embedding model. It provides relevance score by inputting query and passage to the re-ranker. The re-ranker is optimized based cross-entropy loss, so the relevance score is not bounded to a specific range. It accepts a dimension of 384 and sequence length of 512 and has a retriever score of 51.68 on Massive Text Embedding Benchmark (MTEB) (Muennighoff *et al.* 2023).

In the study, the embeddings were fine-tuned using the Multiple Negatives Ranking Loss as the loss function during training.

2.5.4. *Multiple Negatives Ranking Loss*

One of the key advantages of Multiple Negatives Ranking Loss is that it allows the embeddings to be trained using only the positive pairs of texts. The pairs of texts are required so that the model can learn to distinguish relevant (positive) examples from irrelevant ones. By focusing on maximizing the similarity between positive pairs, this approach helps the model more effectively capture the semantic relationships between texts. Ideally, for a search these pairs would constitute a query and the document.

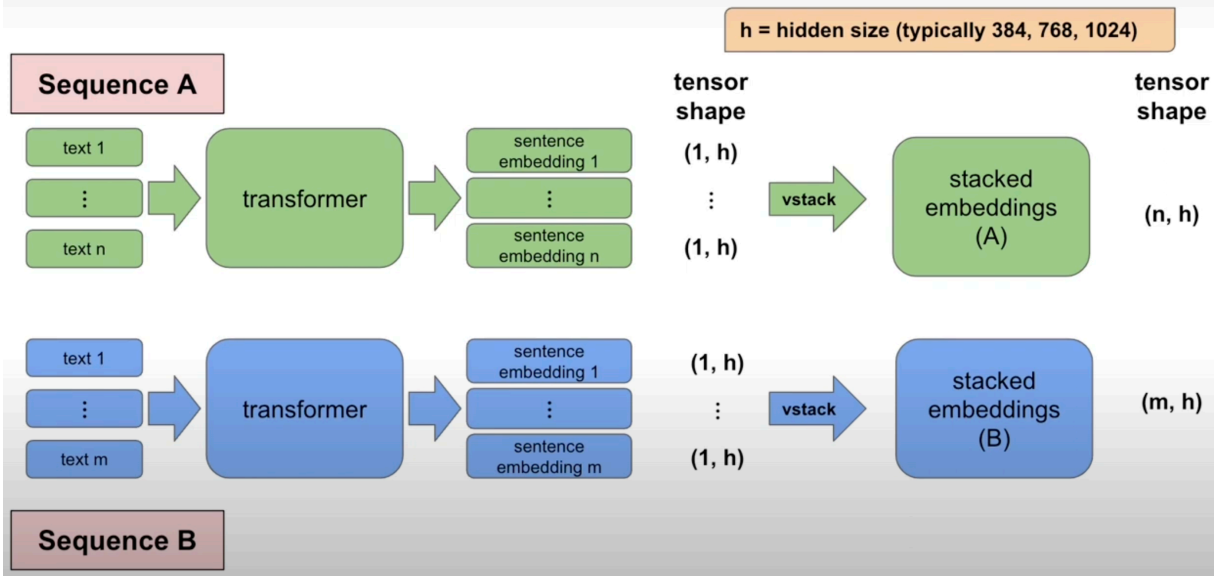

Figure 13. Data preparation for Multiple Negative Ranking Loss [source: Nicholas Broad, (2023)*].*

The illustrates the creation of data for the training using the multiple negatives ranking loss, where 'n' text are passed through a transformer to produce 'n' sentence embeddings and these embeddings are stacked vertically to from stack embeddings (A of size $n \times h$ and similarly B of size $m \times h$) where 'h' is the hidden size of the model.

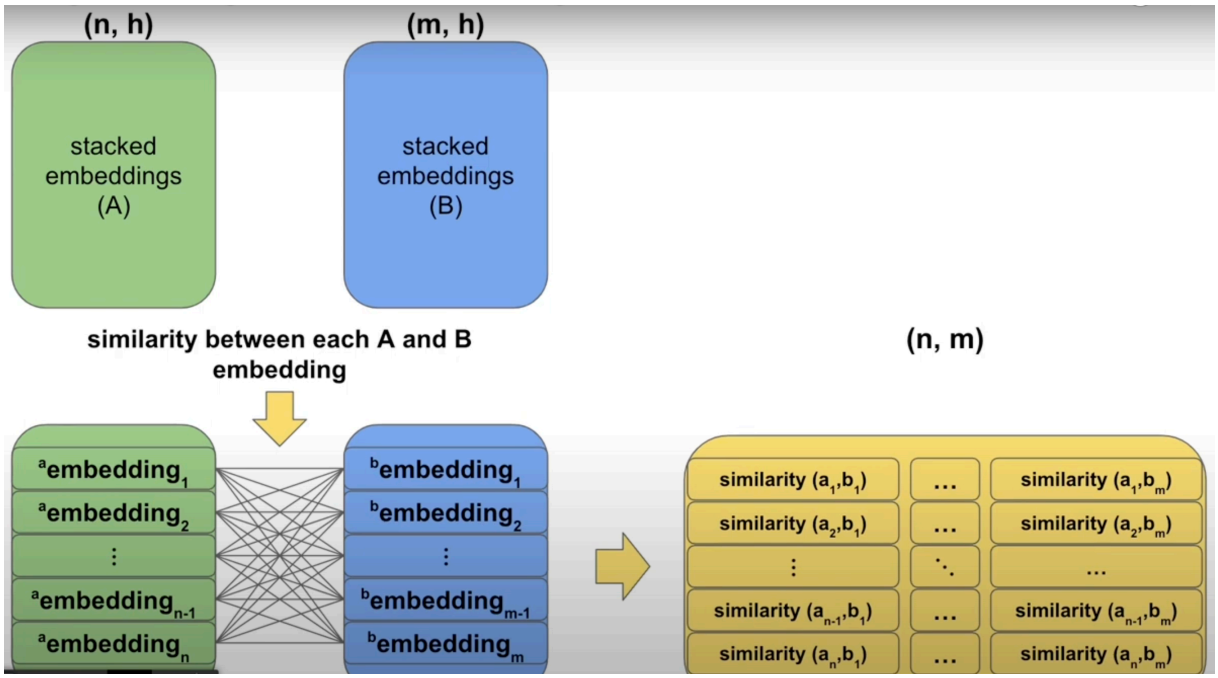

Figure 14. Formation of similarity matrix using embeddings created by stacking individual sentence embeddings [source: Nicholas Broad, (2023)].

With this data, the similarity calculation of each of the embeddings in A with that of B is done using Cosine similarity. This would result in a matrix of size $n \times m$, where the high values would correspond to high similarity between the pairs a_x *and* b_x, where x denotes the index of the matching pair. The cross entropy between the resulting $n \times m$ matrix and the ideal labels is calculated to find the loss. This can be thought of as a single label multi-class classification. The cross entropy calculated for each row of the matrix is averaged and the resultant value would be the loss value (*Nicholas Broad, 2023*).

For each embedding its assumed that there exist only one possible positive pair and all other values are considered as negatives, resulting in each record having multiple negatives, therefore, it is called as **Multiple Negatives Ranking Loss**. In essence, it encourages the model to minimize the angle (in vector space) between the embeddings of positive text pairs while maximizing the angle between embeddings that are not paired.

2.5.5. Token Optimization

All LLMs work with tokens. Tokens are the fundamental units of input and output for language models. They represent individual words, punctuation marks, or even subword units (such as prefixes or suffixes) that the model identifies and processes. When an input prompt is provided to a language model, it is tokenized, meaning it is broken down into individual

tokens, before being processed by the model. When the number of tokens that is sent to the language model for processing increases, the computational cost required increases along with the increase in response time. Many proprietary LLMs such as OpenAIs GPT-4, Google's Bard, Anthropic's Claude2 use token-based pricing models, where the charge is based on the number of tokens sent to the LLMs (input tokens) and the number of tokens received from the LLMs (output tokens). Therefore, token optimization is critical in reduction of the number of tokens and thereby reducing both the economic cost and cost in time.

Some of the widely-used techniques are :

1. **Byte Pair Encoding (BPE)**

Byte Pair Encoding is a subword tokenization technique introduced by Gage (1994), which was later modified for application into the field of NLP by (Sennrich *et al.* 2016). BPE starts with a set of characters and iteratively merges the most frequent pair of characters or sequence of characters to form subwords. The process continues till an iteration limit or token limit is reached. BPE ensures that common words are included in the vocabulary as single tokens, while less common words are split into smaller subword tokens. This approach is consistent with the goals of subword-based tokenization algorithms.

The advantage of BPE is the ability to handle out-of-vocabulary (OOV) words by breaking them into subwords already present in the training data. This makes the BPE explicitly suitable when there exists a constraint on the vocabulary size or the capacity of the model. For instance, BPE has proven effective in neural machine translation (NMT) systems, enhancing their ability to handle various languages and domains by effectively managing rare and complex words (Sennrich et al., 2016).

2. **Wordpiece**

Wordpiece is another subword tokenization technique introduced for speech recognition by Schuster and Nakajima (2012) and later adopted into NLP for neural machine translation by the work of Wu *et al. (*2016). The work done by Wu et al. (2016) creates a vocabulary of subword units by progressively breaking down words into smaller pieces based on their frequency and likelihood similar to BPE, but using a probabilistic method. The algorithm can be broken down into following steps as per the study conducted by Wu *et al.* (2016) :

1. Start by creating a vocabulary that includes all the characters found in the text
2. Train a language model using this initial vocabulary

3. Combine two existing word units to create a new one, then update the vocabulary with this new unit. Choose the new word unit out of all the possible ones, that increases the likelihood on the training data the most when added to the model
4. Repeat the process until you reach a specified number of word units or the improvement in model performance becomes minimal.

The advantage of Wordpiece is that it can also handle OOV words effectively by breaking them down into known subwords. Hence this approach can be applied to scenarios where training data is diverse. Wordpiece is an integral part of the success behind BERT (Devlin et al. 2019) which uses its ability to encode words as combination of subword units and can therefore improve the model's comprehension of complex words and phrases.

3. Sentencepiece

Sentencepiece tokenization was introduced by Kudo and Richardson (2018) and is an unsupervised text tokenizer and detokenizer (converting the label-encoded token ids back into text) that handles raw text as a series of Unicode characters. Unlike methods such as BPE and Wordpiece, Sentencepiece doesn't require pre-tokenized input, making it adaptable to various languages. It learns the most effective subword units from the data itself, providing a more flexible and language-agnostic approach. In contrast, previous tokenization methods like Wordpiece often face challenges with detokenization. These methods do not always allow for an exact conversion between raw text and tokenized sequences because they typically ignore whitespace information. Subword tokenizers, such as those used by Wordpiece, initially split the text into word segments and do not retain whitespace, leading to potential discrepancies between the tokenized output and the original text.

Sentencepiece is extensively used in leading NLP models, including T5 (Raffel et al., 2020) and XLM-R (Conneau *et al.* 2020), because it efficiently handles multiple languages and scripts without requiring language-specific preprocessing. This capability makes it especially useful for creating multilingual and cross-lingual models. Additionally, Sentencepiece supports a range of tokenization methods, such as BPE and unigram models, providing flexibility for different NLP tasks.

4. Latent Semantic Analysis (LSA)

Latent Semantic Analysis (LSA) is a technique developed by Deerwester (1990) that uses singular value decomposition (SVD) to reduce the dimensionality of a term-document matrix.

By representing documents and terms in a lower-dimensional space, LSA captures the latent semantic relationships between them. This method works well for handling synonyms and words with multiple meanings by grouping terms that occur in similar contexts.

LSA has been widely used in information retrieval, document clustering, and topic modeling. For instance, Dumais (1991) demonstrated LSA's effectiveness in improving the performance of retrieval systems by addressing the issues of term variability (different words or phrases used to express similar concepts such as car, automobile, vehicle etc.) and the sparseness of term-document matrices (most entries in the matrix are zero values because documents contain only a small subset of possible terms). Additionally, LSA has found applications in automatic essay scoring and educational assessment, where it helps in understanding and grading the content of student essays based on their semantic content.

2.5.6. Evaluation of RAG

While ROUGE and BLEU are commonly used metrics for evaluating text quality, they mainly measure how much the generated text overlaps with a ground truth text, focusing on n-gram similarity. These metrics work well for tasks like machine translation and summarization, where the goal is to closely match human-written text. However, they might not fully capture the effectiveness of more complex tasks, such as retrieval-augmented generation (RAG) systems, where the relevance and accuracy of the retrieved information are crucial. In these cases, metrics like RAGAS (Retrieval Augmented Generation Assessment) offer a more complete evaluation by assessing both the retrieval and generation aspects, making them a better fit for such systems(Es *et al.* 2023).

2.5.6.1. RAGAS (Retrieval Augmented Generation Assessment)

In this research, RAGAS is used as a framework to evaluate the Retrieval Augmented Generation pipeline as it provides a suite of metrics to evaluate the pipeline (Es *et al.* 2023). The metrics provided by RAGAS as illustrated by (‘Metrics | Ragas’ 2024) include :

1. Faithfulness
2. Answer Relevance
3. Context Precision
4. Context Relevance
5. Context Recall

6. Context entities Recall

It also provides metrics to reevaluate the LLM such as Answer semantic similarity, and Answer correctness.

1. Faithfulness

Faithfulness measures the factual consistency of the generated answer with respect to the context provided. This value is obtained using the answer and the retrieved context. The result is scaled in between 0 and 1, where 1 denotes the highest faithfulness.

$$Faithfulness\ score = \frac{|Number\ of\ claims\ in\ the\ generated\ answer\ that\ can\ be\ inferred\ from\ the\ given\ context|}{|Total\ number\ of\ claims\ in\ the\ generated\ answer|}$$

The generated answer is considered to be faithful if all the claims made in the answer can be inferred from the context provided.

2. Answer Relevance

Answer relevancy is focused on how relevant the answer generated is to the prompt provided. Lower scores are provided to answers which are incomplete or contain repeating information. The value for this metric is calculated using the question, context and the answer. Ideally, the score is obtained by asking the LLM to generate an appropriate question for the generated answer multiple times. The mean cosine similarity is measured between the generated question and the original question.

$$Answer\ relevancy = \frac{1}{N}\sum_{i=1}^{N} \cos\ (E_{gi}, E_o)$$

The idea here is that, if the generated question accurately addresses the initial question, then the LLM should be able to generate questions from the answer that align with the original question.

3. Context Precision

Context Precision is a metric that evaluates whether all of ground-truth relevant items present in the context are ranked higher or not. Ideally, all the relevant chunks must appear at the top ranks. The value for this metric is calculated using the question, context and the ground-truth.

$$Precision@k = \frac{True\ Positive@k}{True\ Positive@k + False\ Positive@k}$$

4. Context Relevancy

The metric calculates the relevancy of the retrieved context and is calculated based on both contexts and the question. The retrieved contexts should contain only essential information to

address the query. The estimation of |S| is done by identifying sentences within the retrieved context that are relevant for answering the query.

$$Relevancy = \frac{|S|}{|Total\ number\ of\ sentences\ in\ retrieved\ context|}$$

5. Context Recall

Context recall measures the extent to which the retrieved context matches with the answer, treated as ground truth. This score is computed with the help of ground truth and context retrieved.

$$Context\ Recall = \frac{|Ground\ Truth\ sentences\ that\ can\ be\ attributed\ to\ the\ context|}{|Number\ of\ Senetences\ in\ Ground\ Truth|}$$

Each sentence in the ground truth is analyzed to determine if the sentence can be attributed to the retrieved context.

2.5.6.2. Normalized Discounted Cumulative Gain (NDCG) Metric

NDCG metric is used to evaluate how well a recommender system works. For a simple classification or regression, the closeness of the predicted value to the actual value can be measured. However, to evaluate a recommender system it is necessary to measure both the relevance of the results and the quality of the ordering. The MAP@K metric is widely used to evaluate recommender systems. It measures how many of the recommended items are relevant and appear towards the top of the list. However, the MAP@K metric has some limitations. It mainly focuses on whether recommended items are relevant or not, ie. it focuses on precision, but it doesn't measure how relevant the recommendations actually are (Dhinakaran 2023).

When users provide explicit ratings for recommendations or when expert evaluators give scores, we have a clearer picture of the relevance. In such cases, we can better assess how relevant the recommendations are. Essentially, the difference between MAP and NDCG is like comparing binary classification, which deals with yes/no decisions, to regression, which involves predicting scores.

NDCG (Normalized Discounted Cumulative Gain) assesses each retrieved document with varying levels of relevance, unlike traditional ranking measures that only consider documents as either relevant or not relevant. This means NDCG can capture degrees of relevance for documents. Furthermore, NDCG uses a discount function that places more importance on higher-ranked documents, unlike many other measures that treat all positions equally. This is

particularly important for search engines since users tend to value the top-ranked documents much more than the lower-ranked one (Wang et al. 2013).

1. Cumulative Gain (CG)

To fully understand NDCG, it's important to first grasp the concept of Cumulative Gain (CG). Cumulative Gain is a sum of gains associated for items within a search query. Cumulative Gain (CG) is calculated by summing the relevance scores of the items retrieved by the recommendation system. Mathematically,

$$CG_{@k} = \sum_{i=1}^{k} G_i$$

2. Discounted Cumulative Gain (DCG)

However, since this does not take into account the ordering, Discounted Cumulative Gain (DCG) is considered. Discounted Cumulative Gain (DCG) adjusts the relevance scores based on their position, giving more weight to recommendations at the top and less weight to those further down the list. This can be mathematically represented using,

$$DCG_{@k} = \sum_{i=1}^{k} \frac{G_i}{log_2(i+1)}$$

However, DCG also has a limitation that the DCG score is dependent on the number of items recommended. A recommender with more items to be recommended would have a higher score when compared to a recommender with less number of expected recommendations ie. a top 10 recommender would have a higher DCG value when compared to a top 3 recommender.

The DCG with a normalization factor is known as Normalized Discounted Cumulative Gain (NDCG). The denominator is the ideal DCG, when we recommend the most relevant items first.

3. Ideal Discounted Cumulative Gain (IDCG) and NDCG

$$IDCG_{@k} = \sum_{i=1}^{k^{ideal}} \frac{G_i^{ideal}}{log_2(i+1)}$$

$$NDCG_{@k} = \frac{DCG_{@k}}{IDCG_{@k}}$$

Where IDCG calculates the DCG of the ideal order based on the gains. It answers the question: what is the best possible DCG for a group? The most relevant information would always be at the top with IDCG.

Comparison of NDCG vs. other ranking metrics

Metric	Accounts for Position	Accounts for Relevance	Accommodates Graded Relevance	Holistic Measure
NDCG	Yes	Yes	Yes	Yes
Precision@k	Partially (up to k)	Yes	No (Binary)	No
Recall@k	Partially (up to k)	Yes	No (Binary)	No
Mean Avg Precision (MAP)	No (only order)	Yes	Sometimes (depends on implementation)	No

Figure 15. Comparison of NDCG with other metrics [source: Aporia, 2023].

The Figure 15 shows the comparison of NDCG with other metrics which are typically used for measuring the recommender systems, and it shows that NDCG takes into consideration the position, relevance and graded relevance and can be used as a holistic measure.

2.5.6.3. Information Retrieval Evaluator

When looking at the optimization of RAG, one of the optimizations involves fine-tuning embeddings. To evaluate the fine-tuned embeddings, the study uses InformationRetrievalEvaluator of the Sentence Transformer. Evaluating retriever models is a bit different from evaluating most language models. Normally, for language models, we input some text and then calculate the error between the predicted values and the true values.

A metric that measures the rate of successful retrievals is required to evaluate the information retrievals. One of the popular metrics that does this is MAP@K, which measure the fraction of retrieved documents that are relevant (average precision value that takes into consideration the top K retrieved documents). The InformationRetrievalEvaluator of Sentence Transformer needs three inputs :

- Corpus – maps the context ids to the context text
- Queries – a dictionary of mappings that maps the question IDs to the question text

- Relevant documents – maps the question IDs to the relevant context IDs.

To create the dictionary objects required, assignment of unique IDs to questions and contexts need to be done without duplication (‘Retriever Models for Open Domain Question-Answering | Pinecone’ 2024). The dataset created for evaluation of the RAG would need these columns in a CSV format.

This section covers various types of recommenders, including content-based and collaborative filtering methods. It delves into the two LLMs considered for this research: Llama-2 (7B) and Mistral (7B). The discussion highlights their training processes and distinctive features, such as Mistral’s Sliding Window Attention.

The exploration then shifts to Knowledge Graphs, examining their role in enhancing recommender systems. Following this, RAGs and their functionality, along with different types of embeddings, are discussed. The history and development of Sentence Transformers are traced back to BERT, and the methods used before SBERT, including Bi-Encoders and Cross-Encoders, are reviewed. These methods are then applied in the next chapter for reranking documents retrieved by the RAG system to improve its performance.
Finally, the study examines some of the sentence transformers used and reviews the RAGAS evaluation metrics, including the NDCG metrics and their derivation.

3. Methodology

The previous section introduced the concepts used in the following discussions. This chapter discusses the dataset acquired for conducting the experiments and the various experiments attempted to gather the results for the recommender system using both Llama-2 and Mistral, and for the optimization of RAG using different concepts such as reranking using cross-encoders, and fine-tuning embeddings, and context enhancements.

All the experiments are run on an Nvidia A 100 GPU with Ollama as the platform for running the LLMs locally due to the ease of usage and it being open-source. The models described below ie. Llama-2, Mistral, Llama-3 are run locally on the GPU server by installing Ollama on the server and downloading and loading the model on Ollama.

3.1. Dataset

3.1.1. Dataset for Recommender System

Since the aim of the research is to provide a recommender system for the airline industry using the comments from the passengers from various platforms like TrustPilot, Airline Ratings etc., the data that would be required to provide the same, can be scrapped from these websites. However, because of ethical guidelines for this research and personal identifiability issue, scrapping was not preferred, and the data was obtained from Kaggle (‘Airline Reviews Dataset’ 2024).

The obtained data contains information from different airlines Air France, Emirates, Japan Airline, Korean Air, Qatar Airways, Singapore Airlines etc. However, since the work done here is meant for any one company and not an analysis of multiple companies, the observations taken into consideration for this research is of Qatar Airways. Based on this we have 1949 distinct records. The data is present in a csv format with fields :

Airline_Verified, Reviews, Type_of_Traveler, Month_Flown, Route, Class, Seat, Comfort, Staff_Service, Food_&_Beverages, Inflight_Entertainment, Value_For_Money, Overall_Rating, is_Recommended.

However, since most of these fields are not relevant for recommendation of alternatives at the airport, a few other fields were considered for the work and only the Reviews are extracted from the provided dataset. The fields considered here are :

1. Booking_ID – The booking Id of the passenger
2. Passenger_Name – The name of the passenger
3. Review – The review obtained from the Kaggle Dataset concerning both positive and negative reviews
4. Membership_level – The membership level of the passenger, here 4 membership levels are considered namely, Bronze, Silver, Gold, Platinum, where priority is given in order.
5. Rating – The rating provided by the passenger for the journey undertaken.

3.1.2. Reasonable Assumptions made

The user details are created using a Faker library (Holland 2023), as attributes needed are Booking_ID, Passenger_Name, Passenger_ID, Flight_Number, Departure_Airport, Arrival_Airport, Departure_Date_and_Time, Arrival_Date_and_Time, Seat_Number, Class, Booking_Date Contact_Information, Payment_Status, Price, Loyalty_Program_Level.
Since these values are the items user interact with in order to provide the user query/complaint, these are created randomly. The number of distinct Departure_Airports and Arrival_Airpor considered here is 10.

In order to obtain a recommendation for each user's query/complaint, some of the assumed recommendations are "Provide Accommodation", "Upgrade", "Refund full amount", "Refund 25%", "Refund 50%", "Provide Meals", "Provide Coupon", "No action needed". This is stored in a column called **Acceptable_Recommendation** in the dataset. Since this data was not available in the initial dataset, Llama-3 is used for creating the Acceptable_Recommendation value using the category of the issue.

The **category** of the issue is obtained by passing the user review to the Llama-3 and are classified into 'delay 1-3 hours', 'delay 3-7 hours', 'delay 7 - 12 hours', 'delay over 12 hours', 'delay over 24 hours', 'customer service issues', 'food issue', 'seating issue', 'entertainment issue', 'baggage handling issue', 'cleanliness issue', 'booking issue', 'flight cancellation issue', 'in-flight service issue', 'other issue'. These are some of the reasonable categories assumed for classifying the issues/queries into. Based on these categories, the LLM provides an

Acceptable recommendation, which will be taken as the ground truth. Since the LLMs are prone to hallucination, manual editing of some of the 1948 records are done. In order to ascertain the decision making of the LLM, an **Explanation** column is also included in the data to make the manual analysis easier.

The resultant dataset is illustrated below:

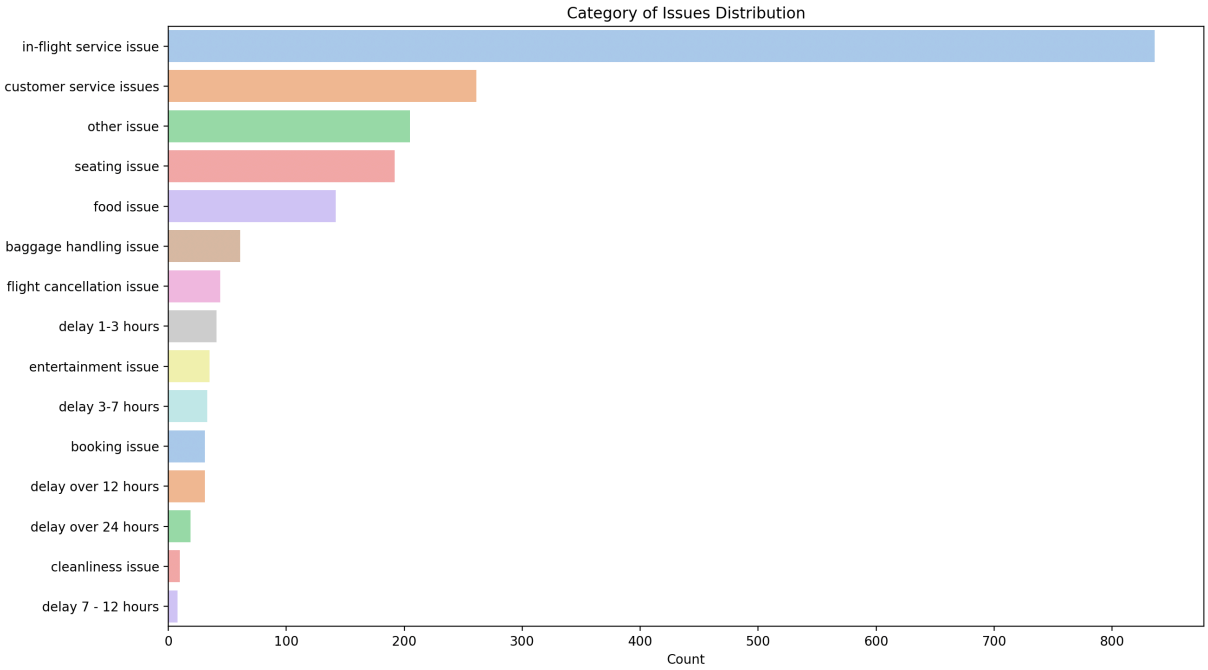

Figure 16. Category of Issues suggested by LLM on the dataset created.

The Figure 16 shows the category identified by Llama-3 for each of the query/issues faced by each of the users. These estimate a realistic data from the issues reported by passengers as most of the passengers faced an in-flight service issue and not delays over 7 hours.

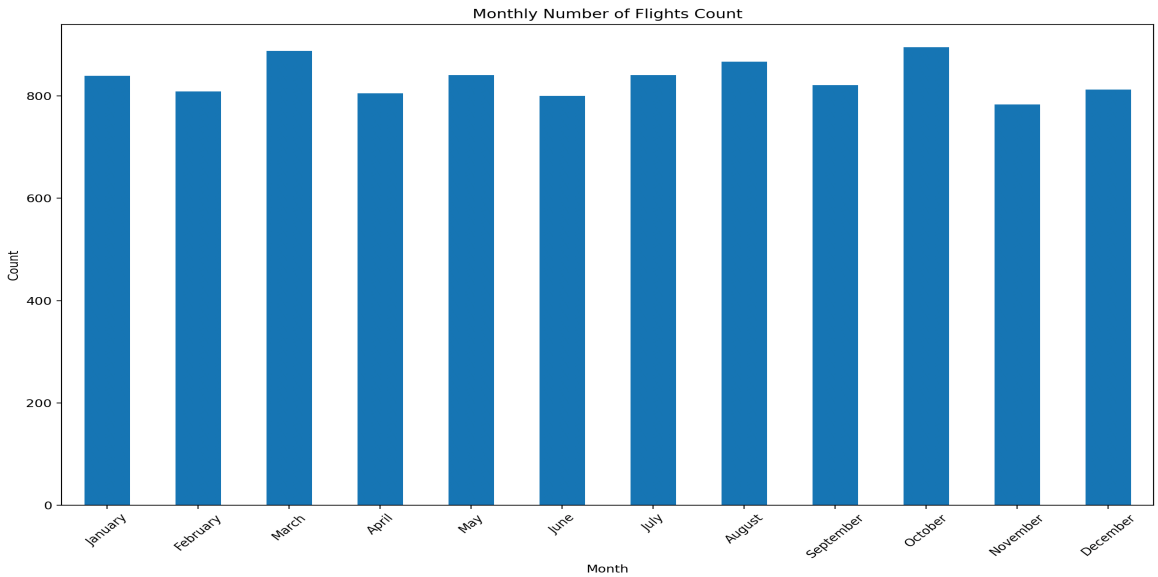

Figure 17. Monthly number of flights taking off in the dataset.

The issues are analyzed for flights that are consistently uniform throughout the year as illustrated by Figure 17.

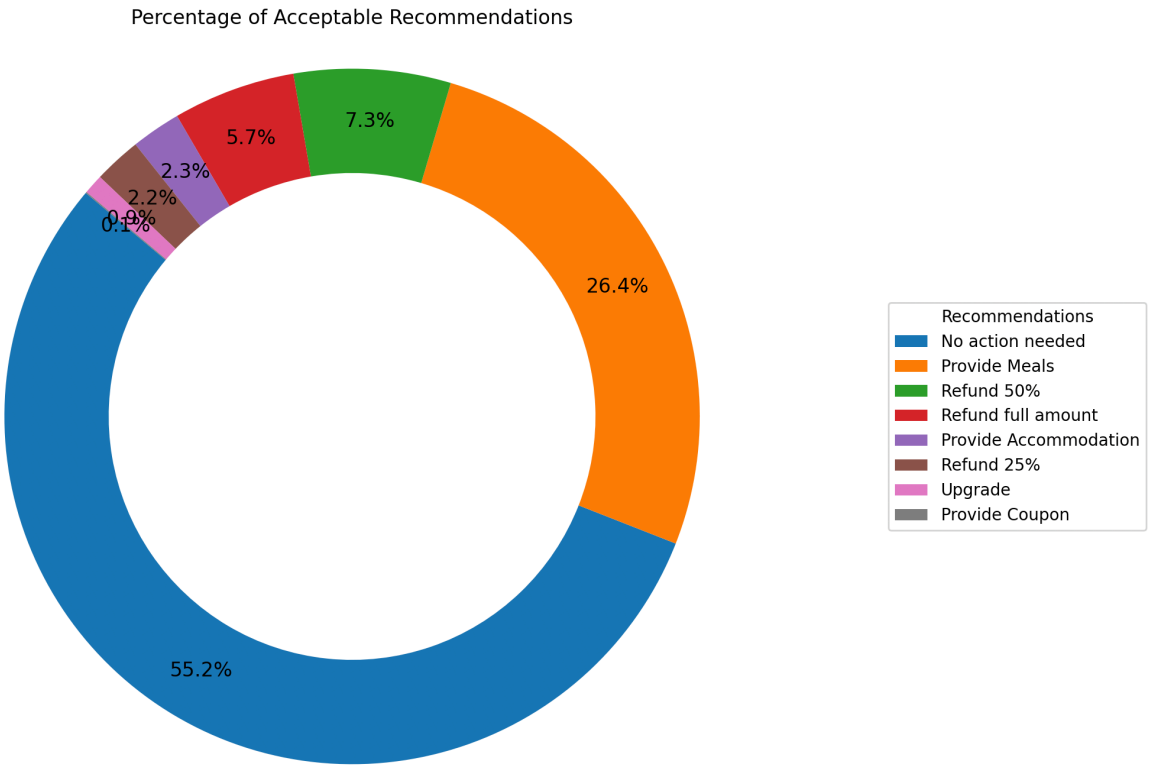

Figure 18. Percentage of Acceptable Recommendations created by LLM that is coherent with real world scenarios.

The generated Acceptable Recommendations are illustrated in Figure 18 as a percentage highlighting the distribution of recommendations deemed acceptable by users. This data is generated by the Llama-3 LLM based on the categories identified which is illustrated in Figure 16. This also mirrors the real-world scenario, where for over 55% of the customers, action recommended is to take 'no action'. The second most popular recommendation created was to provide meals, which was recommended to only for 26.4% of the users in the whole database. From the pie chart it can be observed that the refund and providing accommodation is recommended comparatively lesser due to these items being costly for the airline company. Hence, the probability for the LLM of recommending these would decrease in the future as well. The data is altered like this to mirror the real-world conditions. This way, the models and analysis are more accurate and relevant. By including realistic variations and unexpected factors, complexities faced in real life can be better simulated. This approach helps develop solutions and strategies that are practical and effective in real-world situations.

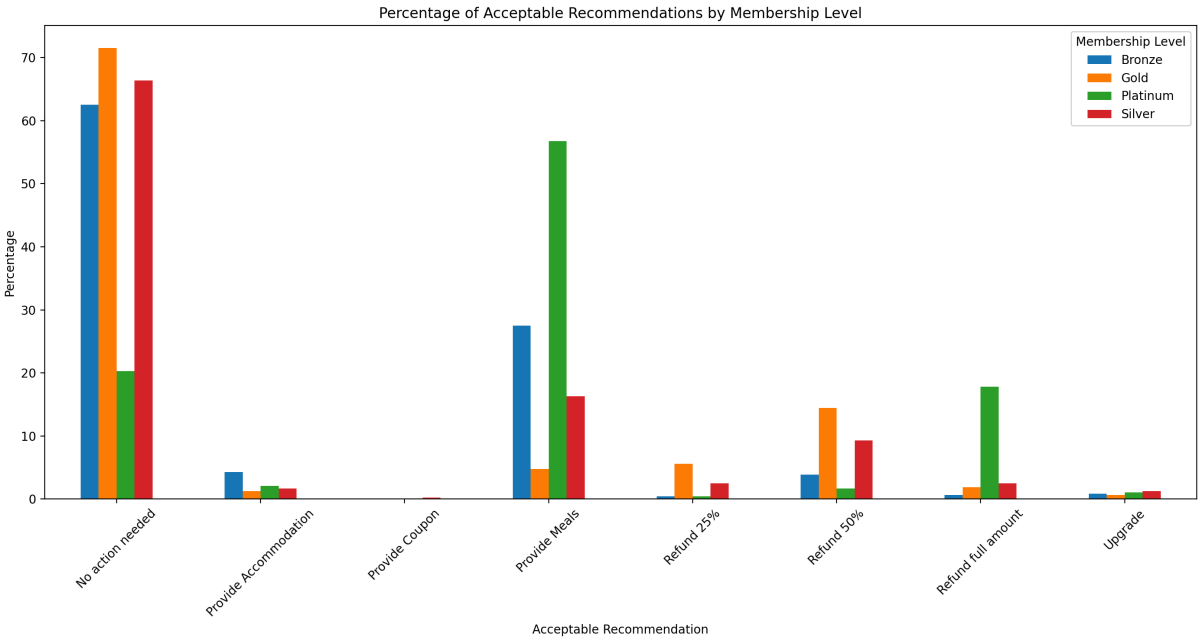

Figure 19. Acceptable Recommendation based on membership level.

The Figure 19 illustrates the percentage of recommendations provided for each class. As shown in the plot, the recommendation 'No action needed' is the most common overall but is the least frequently suggested for passengers in the Platinum class, highly suggested for the other classes of passengers. The next most frequently suggested option is Provide Meals. Once again, the Platinum members have the highest count in this recommendation even. In order to add some irregularity as seen in the real world, the data is augmented to suggest that the recommendation for a 50% refund is more frequent for Gold Members than for Platinum or any other category. However, to make it consistent, the percentage of Refund full amount, is prescribed to Platinum members over 17% higher than that of any other class.

This is how the dataset is constructed, with real world scenarios in mind for the Zero-shot Recommender.

3.1.3. Dataset for RAG optimization

Retrieval Augmented Generation (RAG) combines the strengths of retrieval-based and generation-based approaches, enabling the model to leverage external knowledge effectively while generating coherent and relevant responses with respect to the context. The quality and characteristics of the dataset are crucial to the performance of the RAG model, as it directly influences the model's ability to retrieve pertinent information and generate accurate responses.

In an effort to analyze the optimization for RAG, the dataset used here is the data obtained from Wikipedia using the Wikipedia-api. In Wikipedia, a "category" is a classification system used to group related articles. Categories help organize content and make it easier for users to find information on similar topics. Categories categorize articles into groups based on their subject matter, making it simpler to navigate related topics. For instance, articles about various types of animals might be grouped under the category “Animals”. In the study the categories searched for are:

"Sherlock_Holmes", "Arthur_Conan_Doyle", "A_Scandal_in_Bohemia", "The_Adventures_of_Sherlock_Holmes", "A_Study_in_Scarlet", "The_Sign_of_the_Four", "The_Memoirs_of_Sherlock_Holmes", "The_Hound_of_the_Baskervilles", "The_Return_of_Sherlock_Holmes", "The_Valley_of_Fear", "His_Last_Bow", "The_Case-Book_of_Sherlock_Holmes", "Canon_of_Sherlock_Holmes", "Dr._Watson", "221B_Baker_Street", "Mrs._Hudson", "Professor_Moriarty", "The_Strand_Magazine".

The dataset used in the study is with the category related to ‘Sherlock Holmes’. Since evaluation is done with RAGAS framework besides the NDCG and Recall scores, ground truth and question dataset needed to be generated for the evaluation. For the purpose of creating the ground truth and questions we use Llama-3 as it has relatively good performance when compared with any other open-source LLM till date, and supports multilingual aspect of data generation (‘Introducing Meta Llama-3: The most capable openly available LLM to date’ 2024). This result is manually validated for all the 300 queries generated and the resultant context and the ground truth has been adjusted where the result was incorrect. The occurrences of incorrect results were infinitesimally small when compared to the correct ones. The resultant data is converted into French to test the multilingual aspect of the LLM by using a Llama-3 as a translator with the prompt :

> “You are a French Language Expert converter, convert the text passed as {context_str} to French language, put all the necessary information into the converted text (do not assume the reader knows the text), and return the converted text exclusively in JSON format in the format {{"output":"..."}}
> Provide only the converted text without any text like "Here is the converted text in French, with necessary information added and returned exclusively in JSON format"
> Here is the piece of information: {context_str}
> OUTPUT JSON:”

3.2. Zero-shot Recommender System

3.2.1. Simple Recommender System

A simple Recommender System architecture is provided in the Figure 20.

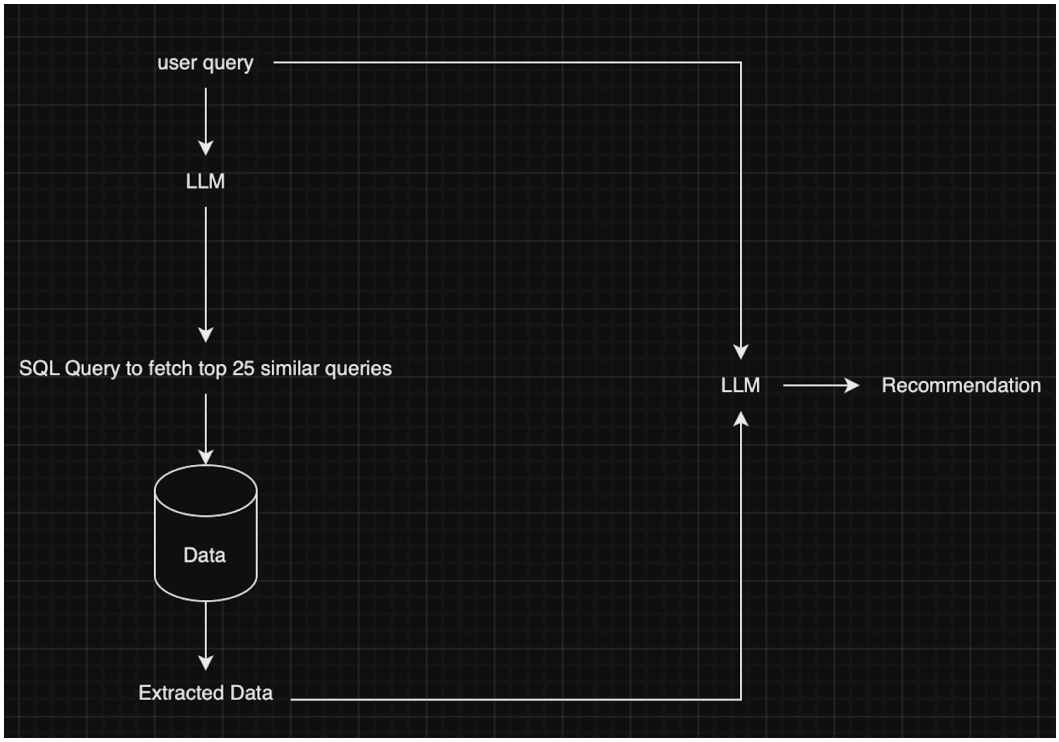

Figure 20. Architecture of a simple recommender system using LLM.

In this architecture, a user (an employee of the airline) provides a query for the passenger addressing the concern of the passenger along with his Booking ID. The Booking ID is a critical aspect of this architecture, as any passenger is assured to have a Booking ID. The working of the recommender is as follows :

1. The provided query from the employee with the concern of the passenger and the Booking ID are passed to the LLM to classify the user concern or query into one of the following categories namely, 'delay 1-3 hours', 'delay 3-7 hours', 'delay 7 - 12 hours', 'delay over 12 hours', 'delay over 24 hours', 'customer service issues', 'food issue', 'seating issue', 'entertainment issue', 'baggage handling issue', 'cleanliness issue', 'booking issue', 'flight cancellation issue', 'in-flight service issue', 'other issue'.
2. Based on the Booking ID, the membership level is identified for the passenger.
3. Taking into account the identified category, LLM creates a query to be executed on the database, which fetches the top 25 records that are belonging to the same category and the same membership level as that of the passenger. The database referred to here,

contains the information about the passengers and the review related information such as Acceptable_Recommendation and Category of the issues identified by the airline.

4. The top 25 records that are fetched is passed to the LLM again to fetch the top k recommendations based on the membership and rating, taking the rating into consideration as this rating denotes the satisfaction of the passenger.

This is the simplest implementation of the Zero-shot Recommender System using LLM. The implementation does not have any training of the model or any other prompting methods. The implementation was done using Llama-2 and Mistral and the observations are recorded and discussed later.

3.2.2. *Recommender System with Knowledge Graph*

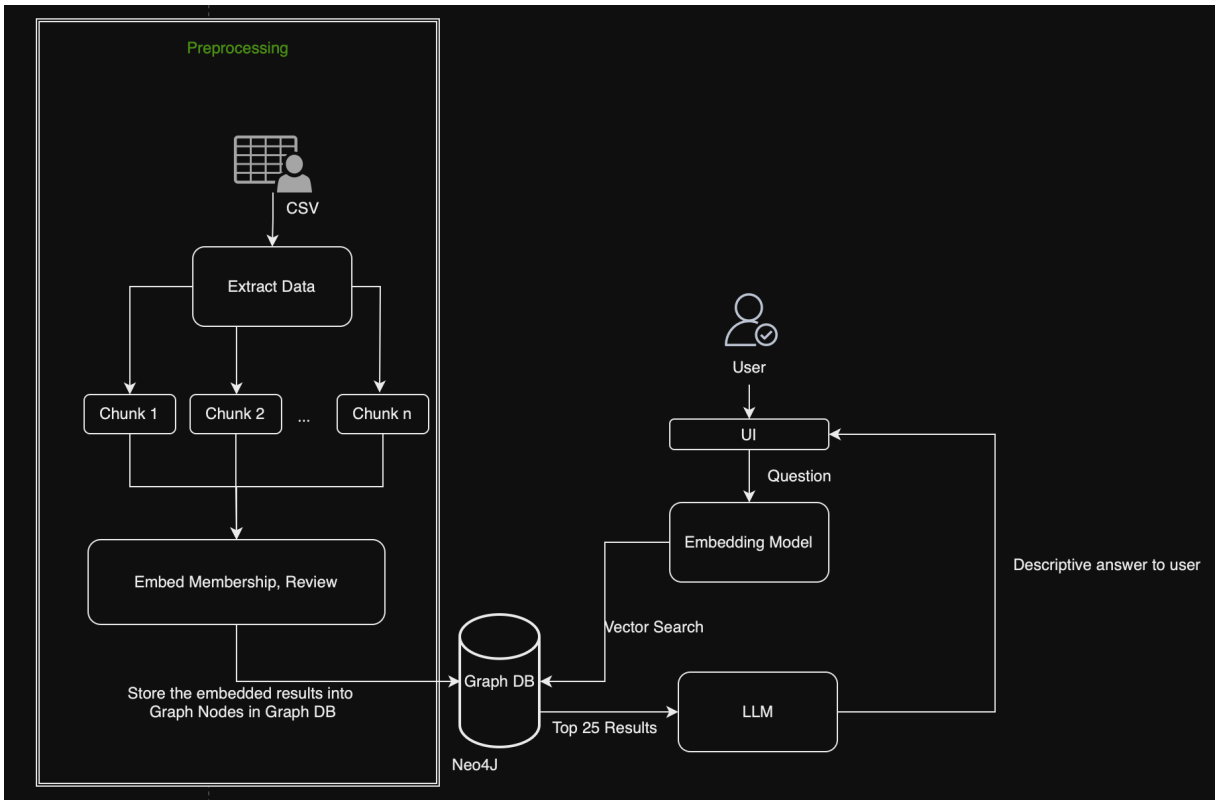

Figure 21. Recommender with Knowledge Graph (KG).

The second architecture considered here is a recommender system with a Knowledge Graph (KG) implementation. Preprocessing step involves the following:

1. Extract the data from the CSV that contains all the fields described in Dataset.
2. Split the data into smaller chunks for efficient memory management

3. Embed the data obtained using a Sentence Transformer (sentence-transformers/all-MiniLM-L6-v2) and store the result in the Graph Database (Neo4J)

In the Literature Review, we explored the significance and applications of knowledge graphs in various domains, highlighting their ability to represent complex relationships between entities and improve data integration. Building on this foundation, our methodology leverages a knowledge graph to enhance the accuracy and comprehensiveness of our data analysis. The knowledge graph for this study will be constructed manually using Cypher queries and using the platform provided by Neo4j, incorporating data from the data created in Dataset. This graph will be used to retrieve the relevant data based on the user query.

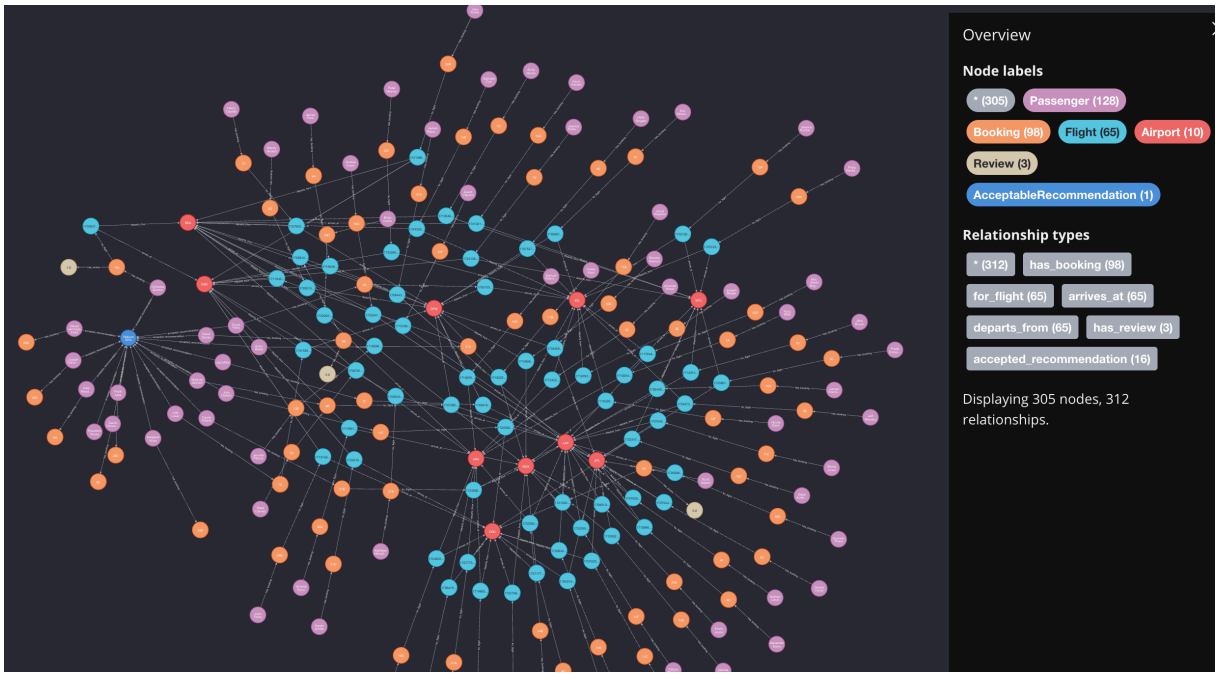

Figure 22. A sub-section of the Knowledge Graph (KG).

The Figure 22 shows a part of the whole knowledge graph under consideration. The figure shows the intricate web of connections between different entities, highlighting how they interact and influence one another within the knowledge graph. The number of nodes can be viewed on the right hand-side of the figure. The KG constructed composes of nodes Passenger, Booking, Flight, Airport, Review and Acceptable_Recommendation.

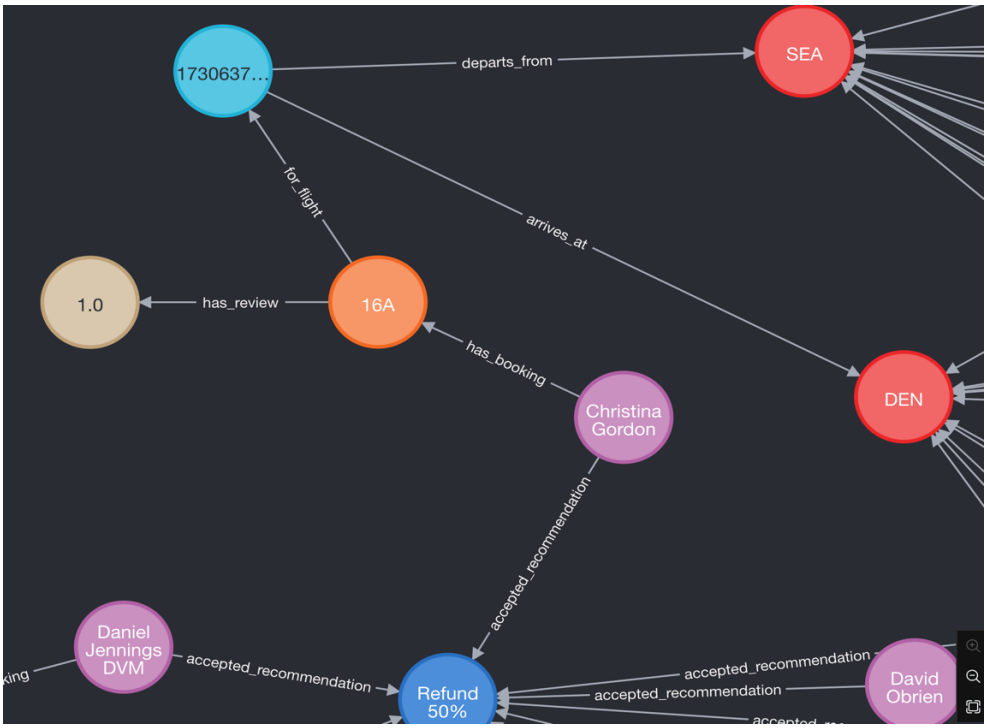

Figure 23. An Enlarged view of the relationships between the nodes in the KG.

The Figure 23 demonstrates the nodes and the relationships between the nodes in finer detail. The figure shows the connections between the entities considered for the problem at hand. The connections can be explained as follows:

The **Passenger** 'Christina Gordon' has-a-booking on **Seat number** '16A' for the **Flight** that **departs at** '17:30' **from** the airport with code 'SEA' and **arrives at** the airport with code 'DEN' and has a **review** with rating '1.0'. This passenger has accepted 'Refund 50%' as the recommendation from the system (Airline company).

Many other passengers who accepted the same recommendation can be viewed from the Figure 23 on the right and the left bottom of the image. The target of using such a system is that this information can be embedded easily and be interpreted even more easily by the LLMs.

The attributes of the review node can be viewed on the Figure 24. The review node has attributes that denotes the membership level of the passenger who reported this review, the rating of the review, the Booking ID, the reasoning behind accepting the recommendation (this field is generated by LLM, not required in the KG).

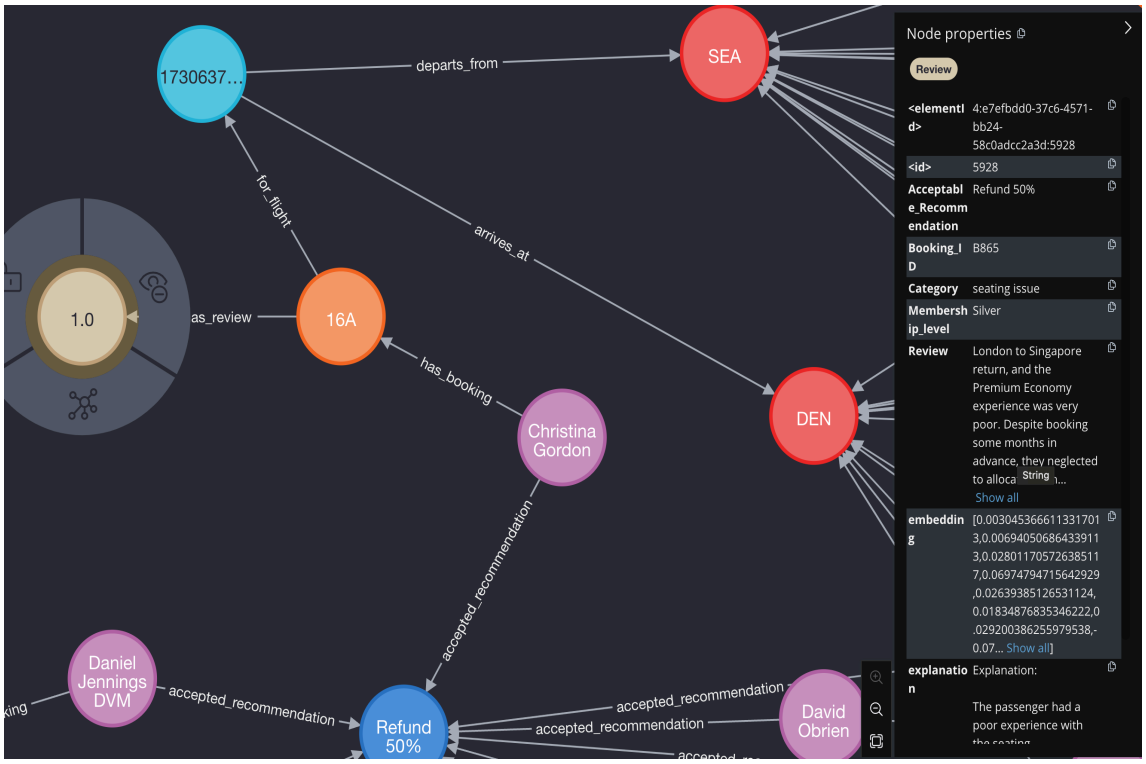

Figure 24. Attributes of Review Node such as Membership_levelm embedding of the concatenated membership, review, and category, the acceptable recommendation.

The working of the recommender system with the Knowledge graph is as follows:

1. The employee provides the passenger concern along with the Booking ID to the system through the UI.
2. Using the Booking ID supplied to the system, it gets the membership level associated with the passenger from the KG created earlier. Along with obtaining the membership level, the system uses the LLM initially for classifying the issue into one of the categories as done for the simple recommender system.
3. While the KG contains this information from the dataset formed as described in Dataset, an additional column called embedding is introduced in the KG, which stores the embedding for the concatenated value of membership, review and the category from the dataset. The embedding is using a Sentence Transformer (sentence-transformers/all-MiniLM-L6-v2) and applied to each of the rows in the dataset, and is stored as an attribute in the review node as illustrated in Figure 24. A vector index is created to store the embeddings for the concatenated value of the membership level, user query, and category using cypher query.
4. Similar to the simple recommender, the next step involves extracting the top 25 records out of the KG database. In order to accomplish this, the information retrieved in the second step namely the membership level and the category is concatenated to

the query from the user and this text data is embedded using the same Sentence Transformer (sentence-transformers/all-MiniLM-L6-v2). This step converts the text into a lower-dimensional space or a vector space.

5. Now the user query can be compared against the dataset using cosine similarity to identify the top 25 records.
6. With the top 25 records identified, the recommendations that are deemed acceptable from the history (recommendation_history) is passed to the LLM along with the query. The recommendation_history comprises of information such as the Acceptable_recommendation, Memebership_level of the passenger, rating associated with the flight by the passenger, the category of the issue identified, and the cosine similarity score derived from matching the input with the dataset. The LLM takes all of these into consideration and outputs a reasonable recommendation along with the explanation of why these recommendations are suggested.

To evaluate effectiveness with the metrics like Precision@K, Recall@K, NDCG as described in section 2.5.6.2, relevance score is required which denotes how relevant each item in a list of recommendations is to a given query. These scores typically range from 0 (not relevant) to higher values, with higher numbers indicating greater relevance (e.g., 1 for binary relevance or higher for more nuanced scales). The evaluation requires two relevance scores namely,

1. **Relevance Score**

   Ground truth relevance: These scores come from the Rating column in the input DataFrame. Each rating reflects how relevant or useful a particular recommendation is expected to be.

   Relevance scores: For the top 3 recommendations generated by the LLM, their relevance scores are determined by matching each recommendation with its corresponding rating in the ground_truth_dict. If a recommendation isn't found in the ground truth, it receives a relevance score of 0.

2. **Ideal Relevance Score**

   Ideal relevance scores: These represent the best possible relevance scores. To find them, all the relevance scores from the Rating column are sorted in descending order, and the top 3 are selected as the ideal scores.

The steps involved in the evaluation are :

1. **Fetch Ground Truth Relevance Scores**:

The ground truth recommendation list is pulled from the Acceptable_Recoomendation column in the recommendations dataframe. Their corresponding relevance score is taken from the Rating column. A dictionary, ground_truth_dict, is created to map each recommendation to its relevance score.

2. **Calculate Relevance Scores for Top 3 Recommendations**:

The top 3 recommendations generated by the LLM are compared against the ground_truth_dict.

For each of these recommendations, the corresponding relevance score is retrieved. If a recommendation doesn't exist in the ground truth, it's assigned a score of 0.

3. **Calculate Ideal Relevance Scores**:

All relevance scores from the ground truth are sorted in descending order.

The top 3 highest scores are then chosen as the ideal relevance scores.

Evaluation Metrics:

The actual relevance scores and the ideal relevance scores are used to calculate various evaluation metrics, such as NDCG, Recall@k, Precision@k, MAP_score (Mean Average Precision), mean reciprocal rank (MRR), hit rate (HR) and F1 score for Llama-2 and Mistral. These metrics help measure how well the top 3 recommendations align with what would be considered ideal recommendations.

3.3. Retrieval Augmented Generation optimization

The extracted textual information from Wikipedia related to Sherlock Holmes is saved into a text file for further processing. Since the RAG has to be evaluated using RAGAS metrics, it requires fields such as question, context, answer and ground truth as mentioned in the Evaluation of RAG section. For the purpose of generating these fields from the extracted texts, an LLM is used, Llama-3 is used due to its ability to comprehend languages and it having the highest scores across the board when compared to any other open-source models. After generation of these fields the resultant data is stored into a CSV file and the results are evaluated using RAGAS framework.

A typical RAG implementation is as illustrated in Figure 25. The Data Preprocessing unit has 3 steps :

1. Extract the source data from Wikipedia using a web crawler or API.
2. Split the document into smaller chunks (chunk size of 1000 and a chunk overlap of 50 is used here) using RecursiveCharacterTextSplitter.

3. Embed the split chunks using an embedding model (all-MiniLM-L6-v2) to store the results in a vector database (ChromaDB).

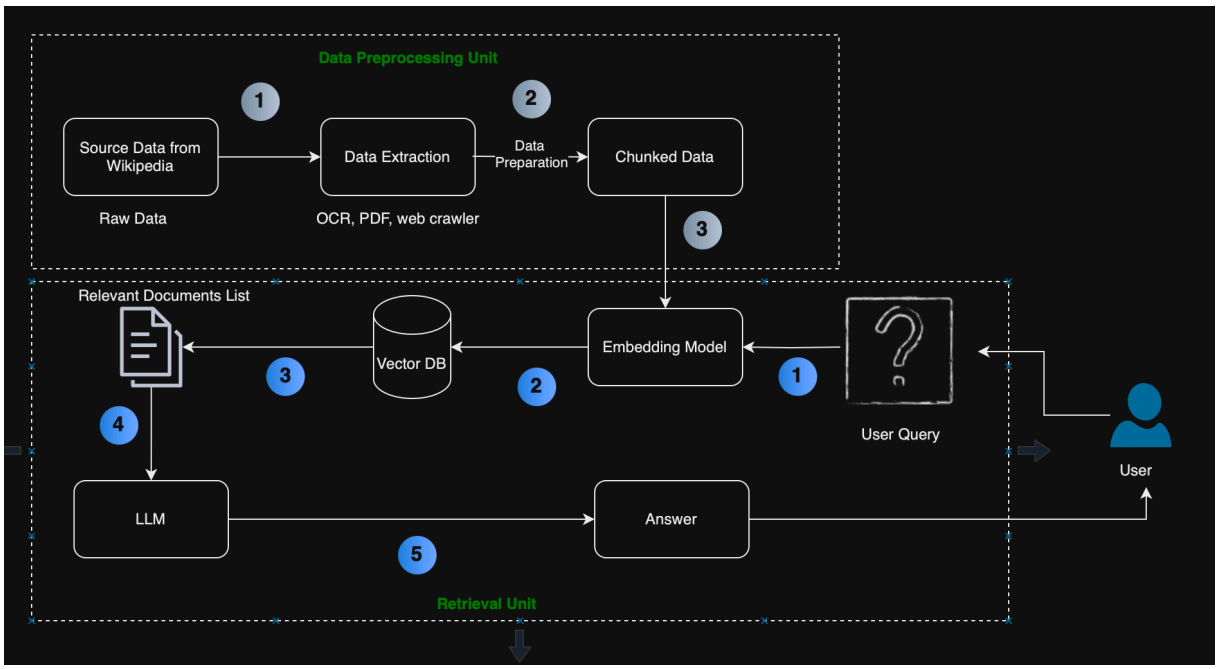

Figure 25. RAG Pipeline showing the different steps in the retrieval augmented generation and the way the result is returned to the user provided a query from the user is consumed by the system [source: (Özker 2024)].

These steps are required for each of the records in the extracted texts so that any user query can be answered based on this data. The Retrieval Unit comprises of 5 steps :

1. When the user asks a query through a user interface, the query is embedded using the same embedding model used to embed the documents in the Data Preprocessing Unit.
2. The embedding of the user query is passed to the Vector database and the system then searches for the relevant documents from the vector database that already has the embedding of every document. The search is typically conducted using cosine similarity between the user query embedding and the documents in the database, to retrieve the most similar top k documents.
3. The data retrieved as a result of the search is combined with the user query of the prompt (the prompt can be modified using prompt engineering to produce better results). The formed result is converted to a format that can be ingested by the LLM.
4. This result is passed to the LLM.
5. Using the data passed to the LLM, an answer is produced by the LLM which is returned to the user.

3.3.1. *Evaluating the LLMs*

The framework used in the study to evaluate RAG is RAGAS framework. In order to run the evaluation of the RAG, the dataset has to be in a format that has 4 different columns namely:

1. Question – The question posed to the RAG for retrieval
2. Answer – The answer from the RAG after passing the retrieved information to the LLM
3. Ground Truth – The expected answer that must be retrieved from the RAG system.
4. Context – The context from which the answer must be generated.

To form the dataset that looks like this from the extracted text from Wikipedia, question and answer needs to be generated from the extracted data. This is done by using Llama-3 and the result is manually verified. The prompt used to generate the question and answer is :

> "Create a question and its answer from the following piece of information,
> put all the necessary information into the question (do not assume the reader knows the text), and return it exclusively in JSON format in the format {{"question": "...", "answer": "..."}}
> Here is the piece of information to elaborate:
> {context_str}
> OUTPUT JSON:"

The answer generated is considered as Ground Truth. This answer is manually verified in order to make this a **Golden Dataset**. With this data, the Ground Truth, Context, and Question is available for RAGAS evaluation. The Answer data is created using another call to the LLM after each of the optimizations applied to the RAG and are evaluated accordingly.

The working of RAG described above can be optimized using techniques that are described below:

3.3.2. *Fine-tuning the embedding*

The above-described model provides a typical RAG with a low score for all of the RAGAS metrics and even a lower score for the NDCG, MRR, MAP etc. This score can be improved by optimizing the relevant documents obtained from the vector database.

The documents are stored in the vector database as embeddings after chunking the documents and passing them through an embedding model. However, there exist several embedding

models each one having unique applications and strengths. While Word2Vec and GloVe concentrate on capturing semantic relationships, models such as BERT, RoBERTa, and GPT-3 are better suited for capturing contextual relationships in text. The accuracy and relevance of the returned documents are influenced by how well the document representations match the query representations, which is why the choice of embedding model can have a substantial effect on the RAG system's performance. Consequently, choosing the right embedding model would need carefully weighing the requirements of the task at hand.

In the implementation at hand, the examination is made on a French Dataset which may not be ideal with the BAAI/bge-small-en-v1.5 embedding model as it is designed for the English Language. However, with fine-tuning this embedding model, the result can be improved. To accomplish this, the dataset is divided into train and test sets with 80:20 ratio. The pictorial representation of embeddings of the Train dataset when projected onto a 2-dimensional vector space is illustrated in the Figure 26.

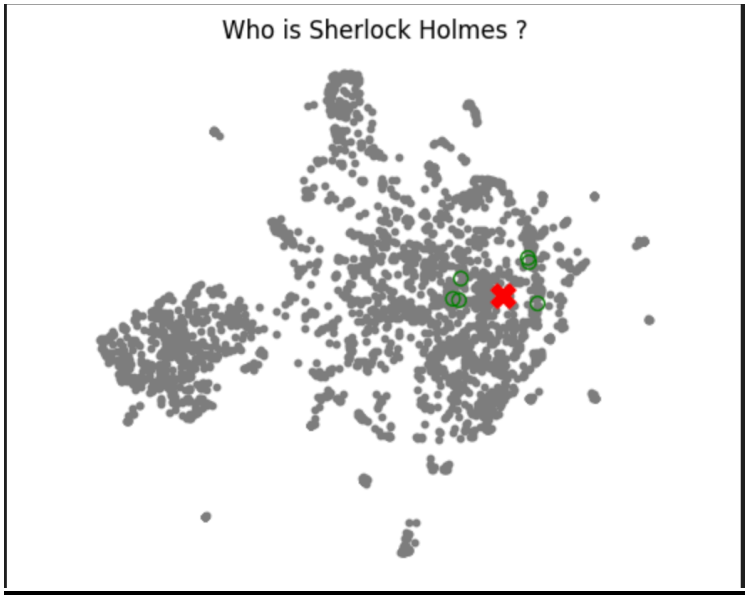

Figure 26. The embeddings of the Train Dataset projected onto a 2-dimensional vector space with the query (red X) and the retrieved documents (green circle) as highlighted.

The figure also shows the query marked with a red X and the retrieved relevant documents which are highlighted in the green circles for a query “Who is Sherlock Holmes?”. From the figure, it is observable that the documents retrieved are in close proximity to the embedding of the query. This can be optimized further by using fine-tuning of the embedding. The training of the embedding model is done on two of the embeddings described in the Sentence Transformer section in literature review, namely BAAI/bge-small-en-v1.5 and

int/float/multilingual-E5-small. This allows for the comparison of performance of multilingual embeddings.

The dataset comprises of documents related to Sherlock Holmes retrieved from Wikipedia, which is in English Language. However, to measure the optimization achieved for multilingual embeddings the data is converted to French language. This is done with the help of Llama-3 model with the prompt:

> "Background information is below.
>
> ---
> {context_str}
> ---
>
> Given contextual information and not prior knowledge.
> Only generate questions based on the query below.
>
> You are a teacher/professor. Your task is to configure \
> {num_questions_per_chunk} questions for next \
> quiz/exam in French. The questions must be of a diverse nature\
> through the document. Provide only the question in French. Limit questions to \
> contextual information provided."

This prompt is looped for each chunks of text extracted from the Wikipedia page related to Sherlock Holmes. This prompt takes in two inputs into consideration :

1. context_str – The text extracted from Wikipedia pages is divided into chunks using RecursiveCharacterTextSplitter, and each resulting chunk is passed as context_str.
2. num_questions_per_chunk – The number of questions that need to be generated for each of the chunks.

The resultant questions and corpus and relevant documents are stored into a CSV file.
The training of the embedding model involves the following steps :

1. Setting training parameters
   - The number of training epochs was set to 2
   - Warmup steps were calculated as 10% of the total training steps, using the formula:

$$warmup\_steps = int(len(loader) * EPOCHS * 0.1) \quad (3.1)$$

2. Training embedding model

- The embedding model was trained using the 80% train data and optimized using the MultipleNegativesRankingLoss function.
- The training was conducted for 2 epochs as increasing the epochs lead to overfitting of the model to the training data.
- The learning rate included a warm up phase calculated using the formula (*3.1*
- The model is evaluated every 50 steps to monitor the progress
- The call to the model.fit method completes the training with an evaluation frequency of 50 steps and the output is moved into a directory.
- This output is loaded onto personal huggingface library for easy access later.

The evaluation of the fine-tuned model is done with the InformationRetrievalEvaluator library of Sentence Transformers.

3.3.3. Multi-query context Expansion

Another way to obtain more relevant documents is using multi-query context expansion, resulting in expanding the context of the search thereby retrieving documents that are related to the original query by considering multiple queries or variations of the query.

Queries similar to the user query can be generated that can fetch related information by embedding all the queries generated and searching the vector database for all related embeddings. This helps in broadening the scope of the search increasing the chances of retrieving more comprehensive and relevant information. By using embeddings, the system can capture the nuances and semantic similarities between queries and documents, in higher quality and more relevant of the search results. This approach not only improves the chances of finding potential answers but also provides a more effective way to retrieve information from complex and diverse dataset such as the Sherlock dataset. To follow this approach, an LLM is required with the prompt :

> "*You are an AI language model assistant. Your task is to generate five different versions of the given user question to retrieve relevant documents from a vector database. By generating multiple perspectives on the user question, your goal is to help the user overcome some of the limitations of the distance-based similarity search. Provide these alternative questions separated by newlines. Only provide the query, no numbering. Original question: {question}"*

The output of the LLM for a question like "Who is Sherlock Holmes?" would be :

1. "Who was the famous detective created by Arthur Conan Doyle?"
2. "Can you find information about the character Sherlock Holmes from the stories of Sir Arthur Conan Doyle?"

3. "Identify the protagonist of the Arthur Conan Doyle's detective stories."

4. "Provide details on the character known as Sherlock Holmes in the works of Sir Arthur Conan Doyle."

5. "Describe the main character in the detective series written by Sir Arthur Conan Doyle, focusing on his name."

This generates a result that has a higher score on RAGAS metrics and the traditional evaluation metrics.

3.3.4. Re-ranking with Cross Encoders

Using method like multi-query expansion and fine-tuning embeddings the relevant documents can be obtained with better quality and and increased accuracy, leading to more precise and contextually appropriate search results. This result can be further optimized by using a specialized model called cross-encoders which evaluates the relevance of documents in relation to the entire query, offering even finer-grained matching. The working of cross-encoders is mentioned in the literature review under the title Sentence Transformer. The cross-encoders are computationally expensive and are slower to from the results. Because accuracy is important for any RAG system, using a two-stage retrieval and re-ranking system would be beneficial (‘Retriever Models for Open Domain Question-Answering | Pinecone’ 2024).

The steps involved here are :

1. Retrieve the most relevant documents using the custom fine-tuned embedding models and the multi-query expansion applied to it.
2. Re-rank the retrieved documents using a more accurate model such as cross-encoders

This is a practice that can be beneficial to follow ie. use a bi-encoder to retrieve many documents, thereby ensuring recall and then use a cross-encoder to re-rank the documents and retrieve the top results with higher precision than that of using only the bi-encoders. The Figure 27 shows the working of combination of bi-encoders with cross-encoder at the time of inference.

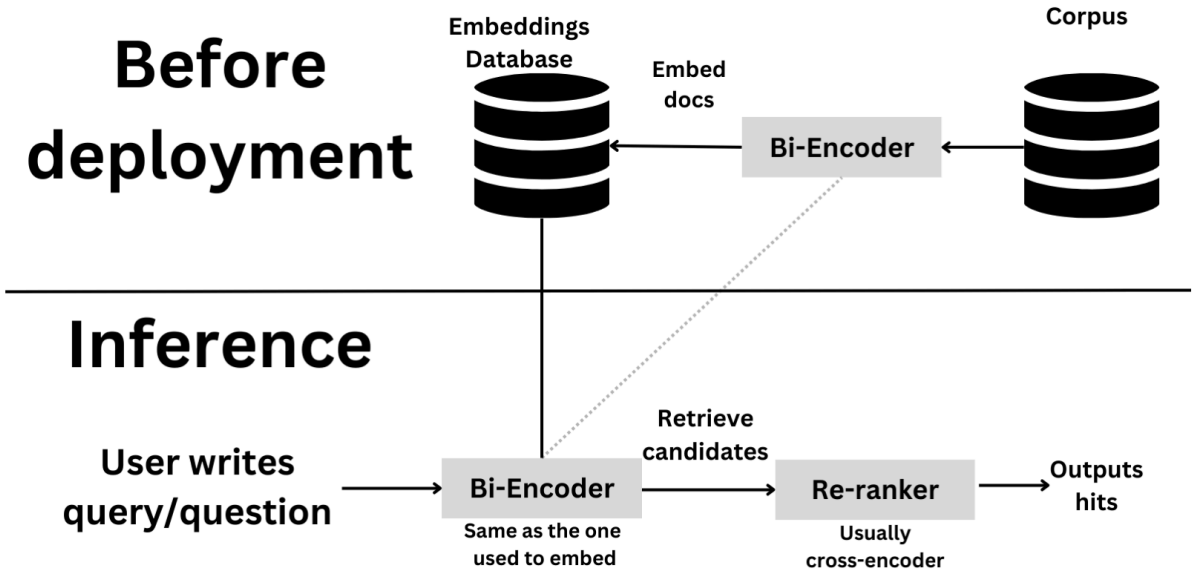

Figure 27. Illustration of how bi-encoders and cross-encoders work together to improve result retrieval [source: Osanseviero, O., 2023].

The data from the corpus is embedded using a bi-encoder and stored in an embedding database or a vector database (if the resultant embedding can be converted to a vector). At inference, the data from the embedding database is matched with the user query that is embedded using the same bi-encoder as that used to embed the corpus and store the result to the embedding database. The retrieved results are passed to the cross-encoder for re-reranking the documents retrieved, and the output retrieved is evaluated using the score provided by the cross-encoder as the relevance of the document to calculate the NDCG, precision, recall etc.

3.4. Token Optimization

3.4.1. Byte-Pair Encoding (BPE)

The observations are made in this part of the study are with the Sherlock dataset created. The steps followed to create the BPE are as follows:

1. **Initialize Vocabulary:**

Create Initial Vocabulary: construct an initial vocabulary from the input data. Each word is broken down into its individual characters, with a special end-of-word token (</w>) appended to each. This results in a dictionary where the keys are these character-based tokens, and the values represent their frequencies in the text.

2. **Count Pairs:**

Calculate Pair Frequencies: calculate the frequency of adjacent token pairs in the vocabulary. This involves:

I. Breaking each token into its constituent characters.
II. Counting how often each adjacent pair of characters appears together.

3. **Merge Most Frequent Pair:**
    1. **Identify the Best Pair:** Determine the most frequent adjacent pair from the counted pairs.
    2. **Merge Tokens:** replace occurrences of this frequent pair with a new token that combines the pair. This step includes:
        I. Creating a regular expression to locate the pair in the vocabulary.
        II. Substituting the pair with the new merged token throughout the vocabulary.

4. **Repeat Merges:**

Perform Iterative Merges: Continue the process of counting pairs and merging the most frequent pair for a specified number of iterations. Each iteration refines the vocabulary by reducing the number of tokens and updating their frequencies.

5. **Tokenize Text with Updated Vocabulary:**

Apply BPE to Text: Use the apply_bpe_to_text function to tokenize new text data based on the final BPE vocabulary. This involves:

1. Breaking each word in the text into its individual characters.
2. Replacing these characters with their corresponding BPE-encoded tokens from the updated vocabulary.

3.4.2. Sentencepiece Tokenization and Wordpiece Tokenization and LSA summarizer

The following are the steps taken to apply Sentencepiece tokenization on the Sherlock dataset.

1. **Load the Sentencepiece Model**:

Initialize Model: Begin by loading a pre-trained Sentencepiece model from Sentencepiece. This model is used for tokenizing text according to the Sentencepiece algorithm.

2. **Define Tokenization Function:**

Apply Sentencepiece Tokenization: Create a function that takes text input and tokenizes it using the Sentencepiece model. This function converts the text into a sequence of tokens according to the Sentencepiece encoding scheme.

3. **Preprocess and Simplify Text:**

Text Simplification: Implement a function to simplify the text by summarizing it. This involves:

I. Parsing the text and tokenizing it.
II. Using a summarization algorithm (e.g., LSA Summarizer) to reduce the text to a specified number of sentences. The sentence count specified here is 2.
III. Joining the summarized sentences back into a simplified version of the original text.

4. **Count Tokens:**

Token Counting: count the number of tokens in a given text by splitting the text into tokens based on whitespace.

5. **Process Dataset:**

Load Data: Read the dataset from a CSV file into a DataFrame.

Iterate Over Rows: For each row in the DataFrame

I. Extract the question, ground_truth, and context from the dataset.
II. Simplify the context using the summarization function.

6. **Tokenization and Analysis:**

Prepare Messages: Construct two sets of messages: one using the original context and another using the simplified context.

Count Original Tokens: Compute the total number of tokens in the original messages.

7. **Apply Sentencepiece:**

Tokenize the simplified messages using Sentencepiece and count the number of tokens in the tokenized messages.

Tokenize the original messages with Sentencepiece and count the number of tokens in these tokenized forms.

8. **Wordpiece Tokenization:**

Tokenize with Wordpiece: Use a separate Wordpiece tokenizer to tokenize both the original and simplified contexts. Convert token IDs to tokens and join them back into text for analysis.

9. **Calculate Token Reduction:**

Compare Token Counts: Measure the difference in token counts before and after applying Sentencepiece tokenization to determine the reduction in token count.

Calculate Reduction Percentage: Compute the percentage reduction in tokens to assess the efficiency of Sentencepiece tokenization compared to the original tokenization.

This chapter delved into the two datasets created from the available data and the exploratory data analysis of the dataset to verify that the created data is relevant and comparable with the real-world scenarios. This chapter also examines the implementation details of the recommender systems, including an exploration of the two different architectures (with and without the knowledge graph) and the construction of the knowledge graph. The chapter also discussed the architecture of a simple RAG system and various optimization techniques for the RAG in order to get better results.

4. Results

The results obtained are illustrated below for a simple recommender with the KG and the recommender with the KG and then deduce the best open-source LLM for application into the airlines industry. Subsequently the discussion follows the optimization of RAG and its results. The experiments were carried out on an Nvidia A100 GPU using Ollama as the platform for running the LLMs locally. Ollama was selected for its ease of use and open-source availability. The models, such as Llama-2, Mistral, and Llama-3, were run locally by installing Ollama on the GPU server and loading the models directly through the platform. Ollama uses GGUFs (GPT-Generated Unified Format) of the large language models for optimum performance.

Since the results derived here are from a novel dataset which was created for testing the usability of open-source LLM for airline industry and the optimization of RAG using data that are related but not from a single document (the process involved extracting documents from Wikipedia related to Sherlock Holmes by providing a set of categories such as "Sherlock_Holmes", "Arthur_Conan_Doyle", "A_Scandal_in_Bohemia" etc.), the result is not directly comparable to existing results. Most of the existing research like the work Adomavicius and Tuzhilin (2005), Potter *et al.* (2022), Wang, Liu, *et al.* (2024) are related to movies dataset, which makes direct comparison challenging.

The target of the recommender system is to recommend top 3 recommendations to the airline employee based on the concern raised by the passenger by looking at the travel history of the passenger. As all the recommender systems are evaluated using metrics such as NDCG, Precision@K, Recall@K, Hit Rate, Mean Average Precision score (MAP), and Mean Reciprocal Rank score (MRR), the evaluation aims to assess the effectiveness of each approach in providing relevant recommendations.

1. NDCG (Normalized Discounted Cumulative Gain): This metric assesses the quality of the recommendations by considering the position of the correct recommendations in the list. Higher NDCG values indicate that relevant recommendations are ranked higher, reflecting better performance in terms of relevance.
2. Precision@K: This measures the proportion of relevant recommendations among the top K recommendations. It provides insight into how often the system's top recommendations are relevant, with higher precision values indicating better performance.

3. Recall@K: This metric evaluates the proportion of relevant items that are recommended out of all possible relevant items. It highlights the system's ability to identify relevant recommendations but does not account for their rank.
4. Hit Rate (HR): This metric indicates whether at least one relevant item is present in the top K recommendations. It provides a straightforward measure of whether relevant recommendations are included but does not capture their ranking.
5. Mean Average Precision (MAP): MAP calculates the average precision across all queries, considering the ranking of relevant items. It provides a summary measure of the system's precision performance across multiple queries.
6. Mean Reciprocal Rank (MRR): This metric assesses the rank of the first relevant recommendation. It is particularly useful for evaluating systems where the position of the first relevant recommendation is critical.

Since the target here is to provide top 3 recommendations, the evaluation is done for top 3 recommendations only, hence the K considered here is 3.

4.1. Simple Recommender without Knowledge Graph

The observations reported in the study are by taking the mean of the above-mentioned scores.

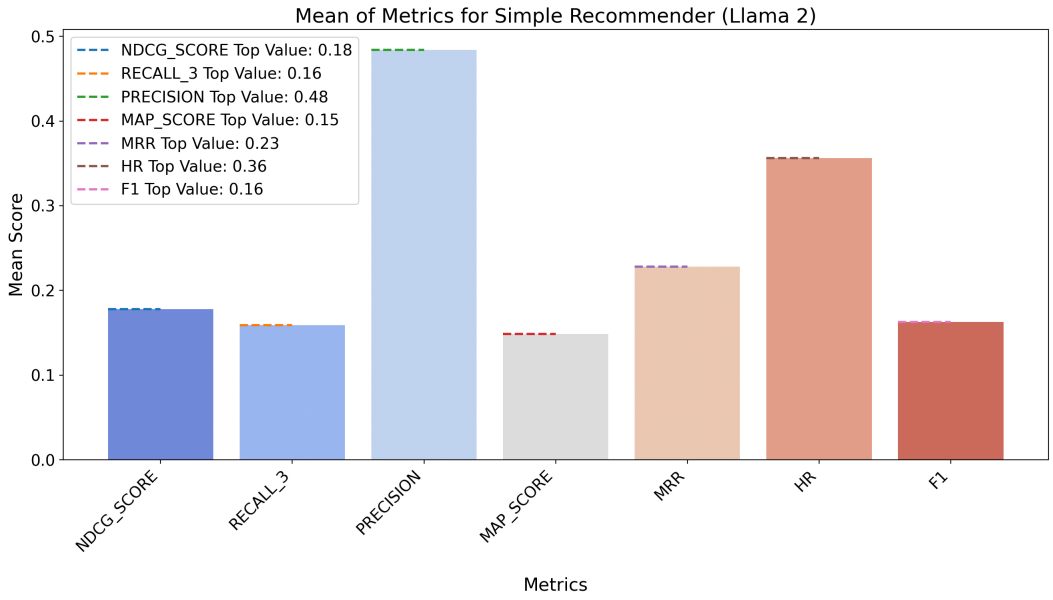

Figure 28. Mean value of the different metrics using Llama-2 as the LLM for the Simple Recommender without knowledge graph.

The Figure 28 illustrates the mean of all the values obtained for a simple recommender created using the Llama-2 as the LLM, plotted using a bar chart.

Using these prompts the results are generated using Llama-2. The NDCG score for Llama-2 is 0.18, the recall score is 0.16, the precision 0.48, the MAP score is 0.15, MRR values is 0.23 and the hit rate is 0.36, with the F1 score as 0.16. These scores are relatively low when compared to the standards.

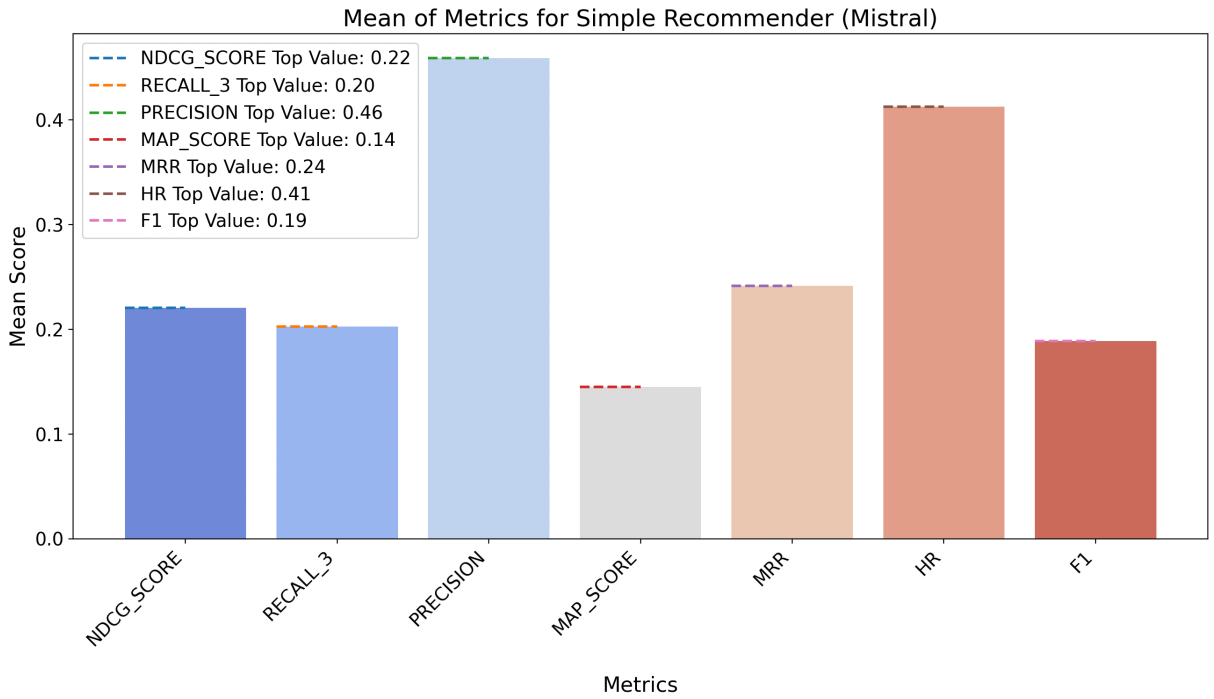

Figure 29. Mean value of the different metrics using Mistral as the LLM for the Simple Recommender.

The Figure 29 shows the mean of all the values for metrics considered for the simple recommender with Mistral as the LLM. The results show a considerable improvement over the results from Llama-2, with an increase of 22% in the NDCG score, which indicated a rise to 0.22 from 0.18. Another considerable increase of 25% was observed for the recall which showed a rise from 0.16 to 0.20 with Mistral. However, the precision value showed a slight decrease from 0.48 to 0.46. Nevertheless, the F1 score, and the Hit Rate has shown an increase of 57% and 18% respectively. Hence for a simple recommender, the Mistral outperforms Llama-2 for the airline data created.

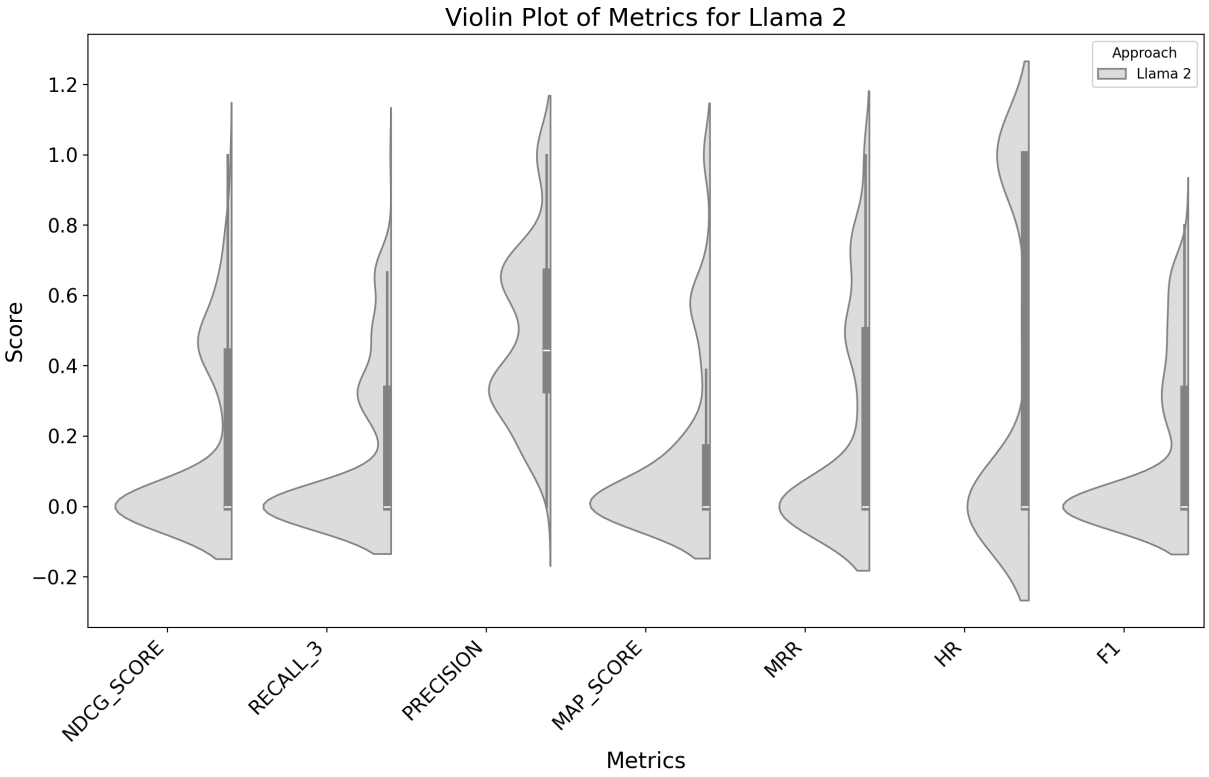

Figure 30. Violin plot of the data distribution for Llama-2 results.

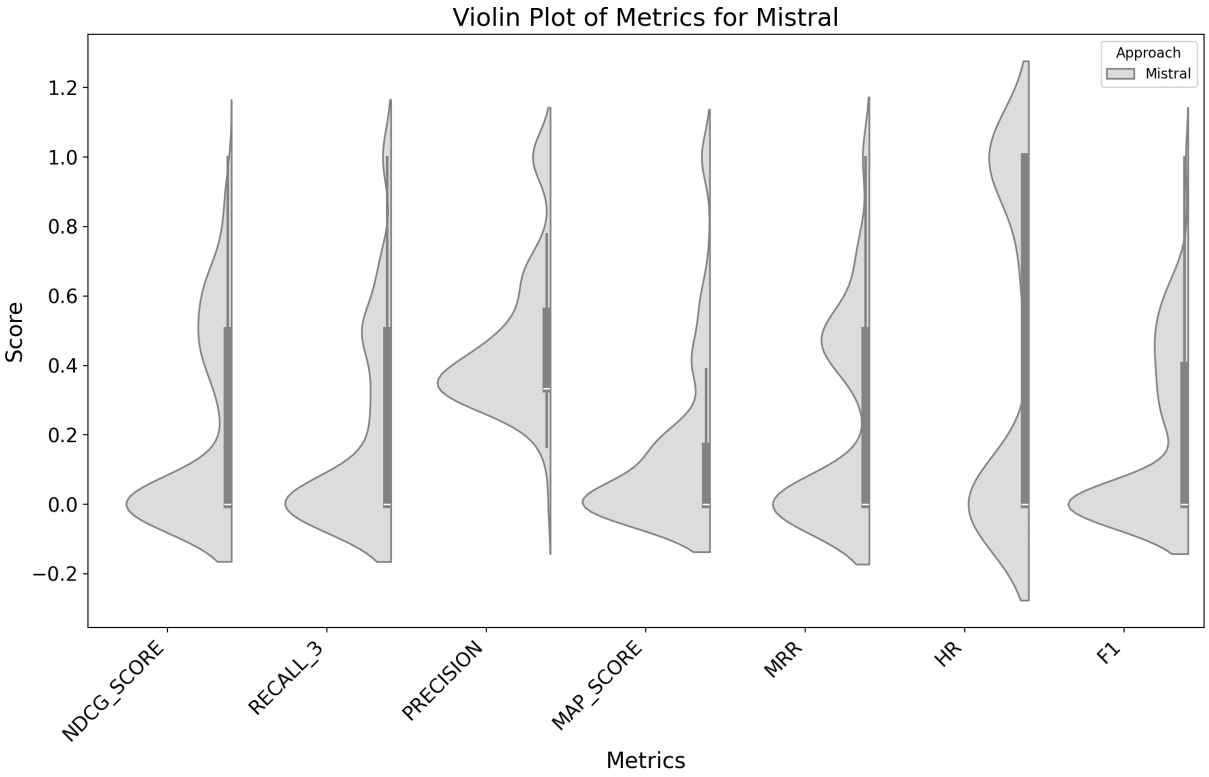

Figure 31. Violin plot of the data distribution for Mistral results.

Looking at the Violin plots (Figure 30 and Figure 31) suggest that the Mistral the distribution of the results are similar for Llama-2 and Mistral indicating that their models are comparable in nature ie. one does not outperform the other by a huge amount. Hence, we introduce the

KG into the architecture to see if we can introduce a huge variation in the metrics considered when comparing these two LLMs.

4.2. Zero-Shot Recommender with KG

In this section, we present the results obtained from implementing a knowledge graph within the recommender system. The data analyzed represents the mean values derived from 400 distinct records, providing a comprehensive overview of the system's performance. Since KGs can capture complex relations that need not be explicitly specified, they should ideally improve the accuracy metrics for the recommender.

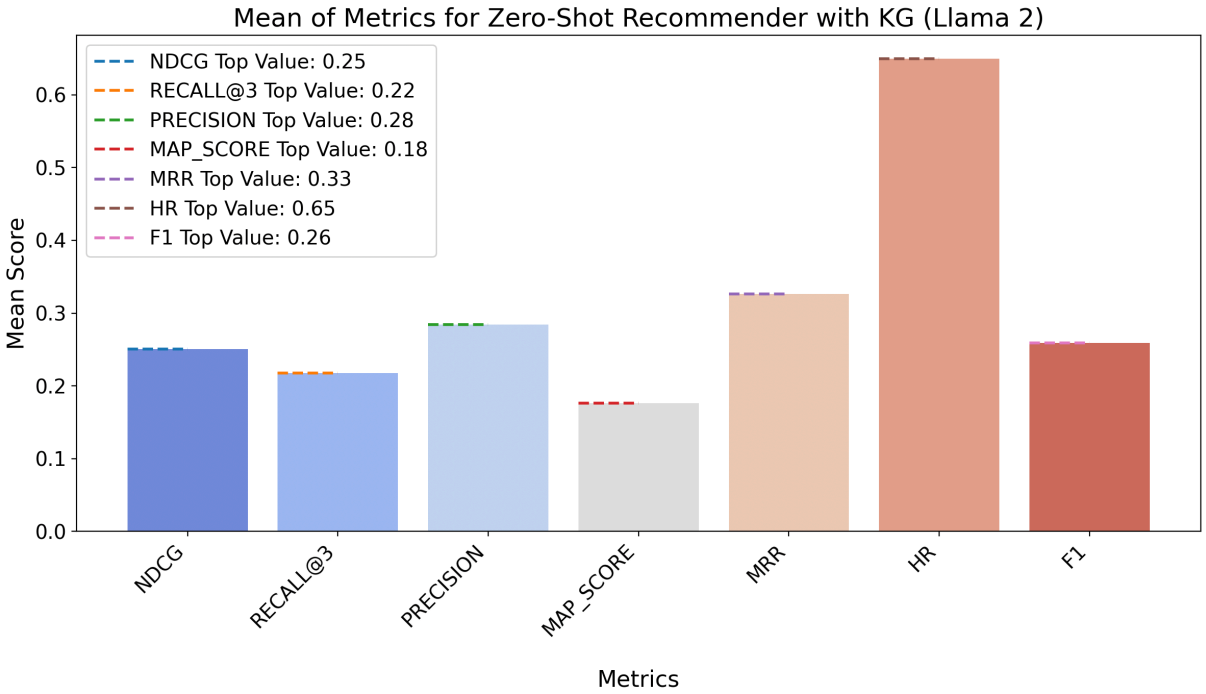

Figure 32. Mean value of metrics illustrated for a Zero-shot Recommender with KG implementation with Llama-2 as the LLM. The results have an improved score when compared to the one without KG.

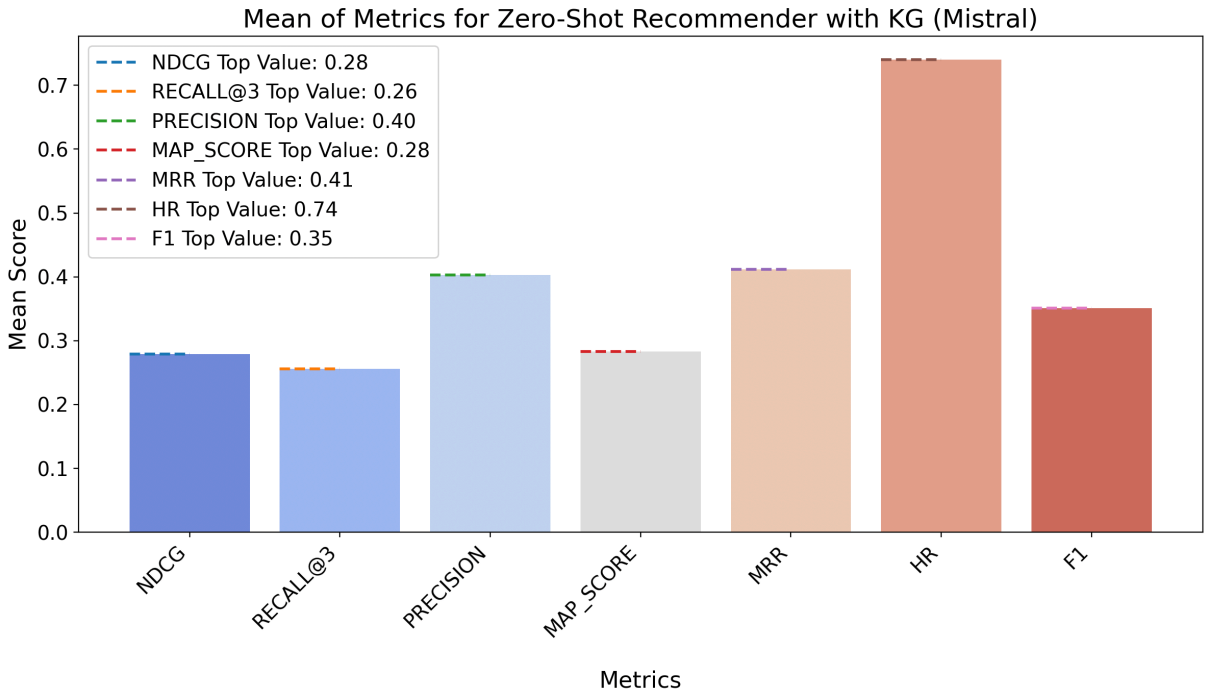

Figure 33. Mean value of metrics illustrated for a Zero-shot Recommender with KG implementation with Mistral as the LLM. The results have an improved score when compared to the one without KG.

The figures Figure 32 and Figure 33 shows the increase in the metrics values when KG is included in the architecture compared to the architecture without the KG. Taking Llama-2 into consideration, the improvement in the NDCG score is over 38% (from 0.18 to 0.25) and the increase in the Hit Rate is over 80% (from 0.36 to 0.65). These metrics themselves show that KGs can enhance the recommender in an efficient way. The other values that showed increase with respect to Llama-2 are MRR value increased from 0.23 to 0.33, F1 score showed an increase from 0.19 to 0.26 which is a rise of 36%. Similarly, with Mistral, the improvement in performance when KG is added can be seen when comparing Figure 33 to Figure 29. It is evident from the comparison that NDCG score exhibited a climb from 0.22 to 0.28 (27% growth), Recall@3 presented a gain from 0.20 to 0.26. Similar to Llama-2, the hit rate and the F1 Score showed a considerable increase of 105% and 84% respectively. These results can confirm that the KG helps improve the recommender system considerably.

In both these observations Mistral outperforms Llama-2 on almost every one of the performance metrics considered for the study. The improvement identified on addition of KG to the model architecture also shows a considerable improvement for Mistral (compare Figure 29 and Figure 33) when compared to Llama-2 (compare Figure 28 and Figure 32).

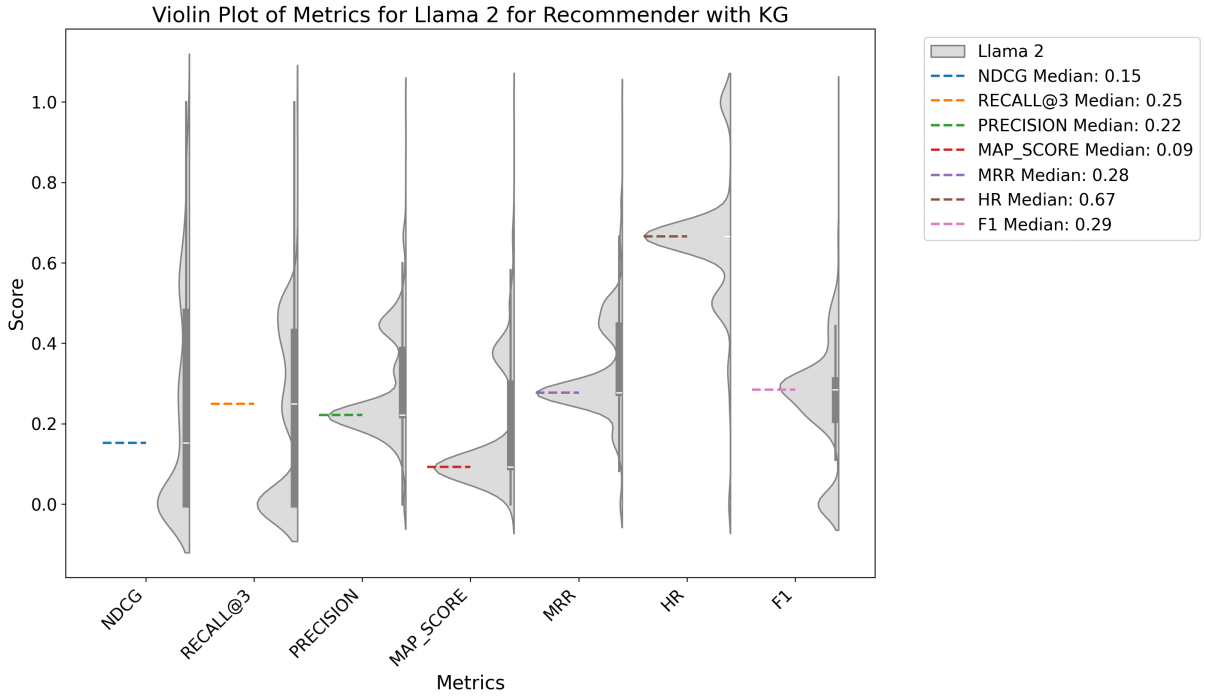

Figure 34. Violin Plot showing the distribution of results for Llama-2 for Recommender with KG the median score highlighted.

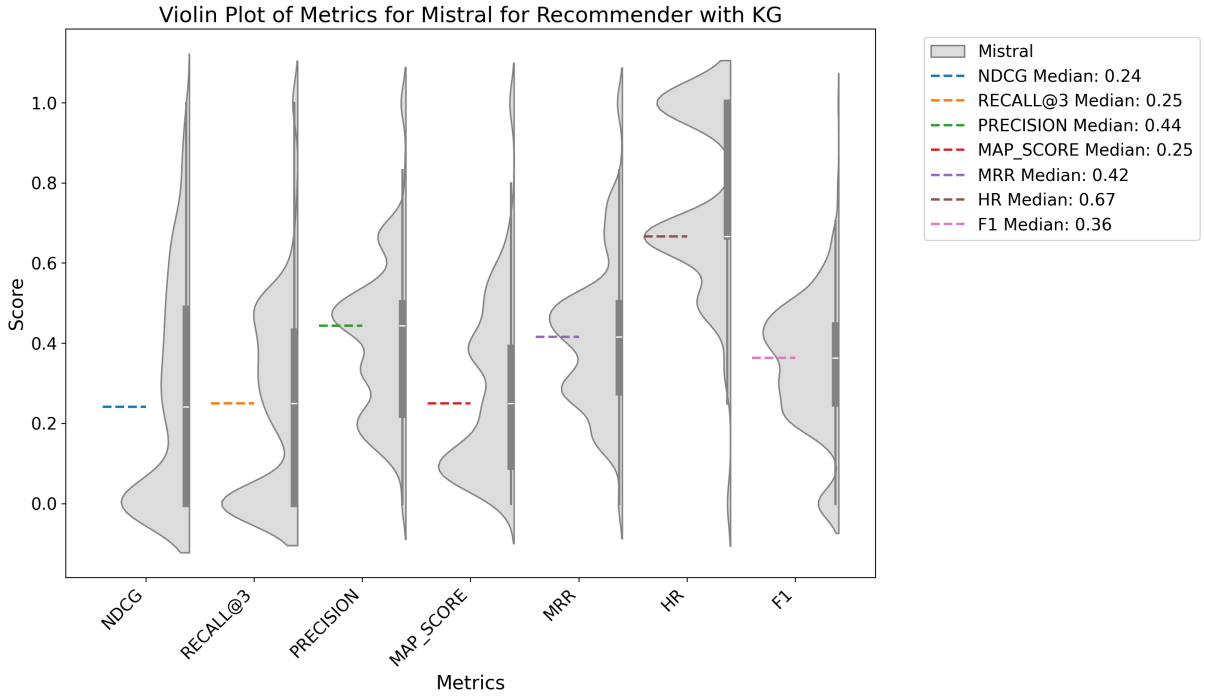

Figure 35. Violin Plot showing the distribution of results for Mistral for Recommender with KG with the median score highlighted.

The Figure 34 and Figure 35 shows the violin plots illustrating the variation in distribution of the results for Llama-2 and Mistral for the recommender with KG implementation respectively. The plot highlights the median scores of the results, demonstrating that the median scores for NDCG, F1, and MAP are higher for Mistral compared to Llama-2. This

indicates that Mistral not only provides more accurate and consistent recommendations but also shows a more reliable performance improvement across these metrics, as reflected by the higher median values.

Hence from the above data we can clearly see that the Mistral (7B) LLM outperforms Llama-2 (7B) model for the airline industry dataset.

4.3. RAG Optimization

The results evaluated in this section are related to the RAG optimization methodologies discussed in the Literature review (Section 2.5) and the methodology (Section 3.3) section of the study. The LLM used here is Llama-3 as it outperforms both Llama-2 and Mistral as per the Figure 9 obtained from the Llama-3 research. As a baseline the study created answers generated using the prompt

> "*Answer the following question: {question} from the context: {context}*
> *put all the necessary information into the question (do not assume the reader knows the text), and return it exclusively in JSON format in the format {{"question": "...", "answer": "..."}},*
> *Do not provide your own hallucinated results, only take the answer from the context and question.*
> *If you cannot find the information, then mark it as 'not mentioned'*
> *Here is the piece of information to elaborate: {context}*
> *OUTPUT JSON:*"

The answers are evaluated using RAGAS metrics for each record in the csv. The average of all the records is presented in the study as the baseline. The baseline results are illustrated in table

context_precision	faithfulness	answer_relevancy	context_recall
0.12749004	0.42184874	0.51365509	0.86475

Table 1. RAGAS metrics for a simple RAG.

4.3.1. Fine-tuning Embeddings

The embeddings considered in the study are detailed in Sentence Transformer section of the Literature Review. These are :

1. **BAAI/bge-small-en-v1.5**

2. **Sentence-transformers/all-MiniLM-L6-v2**
3. **Int/float/multilingual-E5-small**

The initial embedding used is **Sentence-transformers/all-MiniLM-L6-v2** due to its wide applicability in different domains and the MTEB score of the model.

The result obtained on the training set using this embedding model is illustrated in Figure 36

Figure 36. The pictorial representation of vector embedding for Sentence-tranformers/all-MiniLM-L6-v2 for the question "Who is Sherlock Holmes ?" presented on a 2-dimensional plane with the query (marked with red X) and the retrieved relevant documents (marked with green circles).

The embedding model that is commonly used for English language is BAAI/bge-small-en-v1.5. The dataset that was split into training and testing with a 80:20 ratio is used to train the embedding BAAI/bge-small-en-v1.5 with the MultipleNegativesRankingLoss function as the loss function from the sentence_transformers library. The Figure 37 illustrates the performance metrics for BAAI/bge-small-en-v1.5 embedding before fine-tuning the model with the training data. This table offers a comprehensive comparison of Cosine and Dot product similarities across several evaluation metrics, including Accuracy, Precision, Recall, MRR, NDCG, and MAP, at various k values. The metrics are displayed for k=1,3,5,10, 100, highlighting the performance differences between the two approaches. All the embeddings are evaluated using this approach.

Metric	Measure	@1	@3	@5	@10	@100
Cosine	Accuracy	0.3576	0.4735	0.5430	0.5894	-
	Precision	0.3576	0.1578	0.1086	0.0589	-
	Recall	0.3576	0.4735	0.5430	0.5894	-
	MRR	-	-	-	0.4306	-
	NDCG	-	-	-	0.4689	-
	MAP	-	-	-	-	0.4439
Dot	Accuracy	0.3576	0.4735	0.5430	0.5894	-
	Precision	0.3576	0.1578	0.1086	0.0589	-
	Recall	0.3576	0.4735	0.5430	0.5894	-
	MRR	-	-	-	0.4306	-
	NDCG	-	-	-	0.4689	-
	MAP	-	-	-	-	0.4439

Figure 37. The performance metrics of BAAI/bge-small-en-v1.5 embedding on the test data before fine-tuning. This table provides a detailed comparison of the Cosine and Dot product similarities across various evaluation metrics such as Accuracy, Precision, Recall, MRR, NDCG, and MAP at different k values. The metrics are presented for k=1,3,5,10,100, where k is the number of recommendations, showcasing the performance variations between the two methods.

Metric	Measure	@1	@3	@5	@10	@100
Cosine	Accuracy	0.3742	0.5298	0.5828	0.6358	-
	Precision	0.3742	0.1766	0.1166	0.0636	-
	Recall	0.3742	0.5298	0.5828	0.6358	-
	NDCG	-	-	-	0.5074	-
	MRR	-	-	-	0.4660	-
	MAP	-	-	-	-	0.4768
Dot	Accuracy	0.3742	0.5298	0.5828	0.6358	-
	Precision	0.3742	0.1766	0.1166	0.0636	-
	Recall	0.3742	0.5298	0.5828	0.6358	-
	NDCG	-	-	-	0.5074	-
	MRR	-	-	-	0.4660	-
	MAP	-	-	-	-	0.4768

Figure 38. The performance metrics of BAAI/bge-small-en-v1.5 embedding on the test data after fine-tuning. The improved figures show the improvement in performance of the results when using the fine-tuned embedding model.

The Figure 38 shows the improvement in the performance metrics after fine-tuning the model with the training dataset. All the metrics in the results show a considerable increase in the scores. For both Cosine and Dot metrics, accuracy at k=10k=10 showed a significant increase from 0.5894 to 0.6358, indicating a marked improvement in the model's ability to correctly

identify relevant recommendations when more options are available. Recall at k=3k=3 for both metrics also saw a substantial rise from 0.4735 to 0.5298, suggesting that the model is retrieving a larger proportion of relevant items at this level, thereby enhancing the overall effectiveness of the recommendation system. Additionally, the NDCG score at k=10 improved from 0.4689 to 0.5074, reflecting better ranking of relevant items, which is crucial for user satisfaction.

Figure 39. The vector embeddings for the fine-tuned BAAI/bge-small-en-v1.5 model, depicting the question "Who is Sherlock Holmes?" on a 2-dimensional plane. The query is represented by a red X, while the retrieved relevant documents are shown as green circles.

The Figure 39 illustrates the pictorial representation of fine-tuned BAAI/bge-small-en-v1.5 embedding on the test data for the question “Who is Sherlock Holmes” is displayed on a 2-dimensional plane with the query marked with a red X and the retrieved relevant document marked with green circles. The relevant documents are near the question as per the Figure 39.

4.3.1.1. Evaluation of fine-tuned BAAI/bge-small-en-v1.5 and comparison with Sentence-transformers/all-MiniLM-L6-v2

The result shows the comparison of the fine-tuned BAAI/bge-small-en-v1.5 and Sentence-transformers/all-MiniLM-L6-v2.

Figure 40. The pictorial representation of the vectors on a 2-dimensional plane with the query highlighted with a red X and the relevant documents retrieved highlighted with green circles for Sentence-transformers/all-MiniLM-L6-v2 for a question "Who is Sherlock Holmes ?".

The Figure 40 illustrates the data retrieved are even closer with the embedding for Sentence-transformers/all-MiniLM-L6-v2 when compared to Figure 39 which was for the fine-tuned BAAI/bge-small-en-v1.5.

However, since this result can be for this particular question, the performance metrics are evaluated.

Metric	Measure	@1	@3	@5	@10	@100
Cosine	Accuracy	0.2947	0.4205	0.4735	0.5430	-
	Precision	0.2947	0.1402	0.0947	0.0543	-
	Recall	0.2947	0.4205	0.4735	0.5430	-
	NDCG	-	-	-	0.4134	-
	MRR	-	-	-	0.3725	-
	MAP	-	-	-	-	0.3865
Dot	Accuracy	0.2947	0.4205	0.4735	0.5430	-
	Precision	0.2947	0.1402	0.0947	0.0543	-
	Recall	0.2947	0.4205	0.4735	0.5430	-
	NDCG	-	-	-	0.4134	-
	MRR	-	-	-	0.3725	-
	MAP	-	-	-	-	0.3865

Figure 41. The performance metrics of Sentence-transformers/all-MiniLM-L6-v2 for the Sherlock Holmes Dataset on the test data. All of the metrics values shows a decrease when compared to the fine-tuned BAAI/bge-small-en-v1.5.

The Figure 41 depicts the performance metrics model when using the Sentence-transformers/all-MiniLM-L6-v2 embedding on the test data. It underperforms when compared

to the data on the fine-tuned BAAI/bge-small-en-v1.5 as illustrated in Figure 38. The most significant improvement is seen at @1, where the accuracy of the fine-tuned BAAI/bge-small-en-v1.5 model increases from 0.2947 to 0.3742, indicating a substantial enhancement in the accuracy of the top recommendation. The NDCG@10 for BAAI/bge-small-en-v1.5 also shows a notable rise from 0.4134 to 0.5074, suggesting better ranking of relevant items. Additionally, the MAP@100 score improves meaningfully, with BAAI/bge-small-en-v1.5 achieving 0.4768 compared to 0.3865 for MiniLM, reflecting greater precision across a broader set of recommendations.

Given that the dataset is in French and the fine-tuned embedding model is designed for English, the performance of the int/float/multilingual-E5-small model was evaluated as this model was designed for multilingual interpretation.

4.3.1.2. Performance metric of int/float/multilingual-E5-small embedding

Metric	Measure	@1	@3	@5	@10	@100
Cosine	Accuracy	0.3742	0.5232	0.5563	0.6192	-
	Precision	0.3742	0.1744	0.1113	0.0619	-
	Recall	0.3742	0.5232	0.5563	0.6192	-
	NDCG	-	-	-	0.4971	-
	MRR	-	-	-	0.4583	-
	MAP	-	-	-	-	0.4700
Dot	Accuracy	0.3742	0.5232	0.5563	0.6192	-
	Precision	0.3742	0.1744	0.1113	0.0619	-
	Recall	0.3742	0.5232	0.5563	0.6192	-
	NDCG	-	-	-	0.4971	-
	MRR	-	-	-	0.4583	-
	MAP	-	-	-	-	0.4700

Figure 42. The performance metrics of int/float/multilingual-E5-small embedding for the Sherlock Holmes Dataset on the test data at different values of k. The metrics demonstrate a notable similarity in results when compared to the fine-tuned BAAI/bge-small-en-v1.5 model.

The metric from Figure 42 indicate that an untrained int/float/multilingual-E5-small embedding can perform comparably to a fine-tuned BAAI/bge-small-en-v1.5 embedding. The accuracy score, precision, NDCG values at different values of k approaches results obtained with the fine-tuned BAAI/bge-small-en-v1.5 embedding. Hence, the investigation delved into the performance of a fine-tuned int/float/multilingual-E5-small model.

Metric	Measure	@1	@3	@5	@10	@100
Cosine	Accuracy	0.3940	0.5464	0.6026	0.6457	-
	Precision	0.3940	0.1821	0.1205	0.0646	-
	Recall	0.3940	0.5464	0.6026	0.6457	-
	NDCG	-	-	-	0.5227	-
	MRR	-	-	-	0.4831	-
	MAP	-	-	-	-	0.4935
Dot	Accuracy	0.3940	0.5464	0.6026	0.6457	-
	Precision	0.3940	0.1821	0.1205	0.0646	-
	Recall	0.3940	0.5464	0.6026	0.6457	-
	NDCG	-	-	-	0.5227	-
	MRR	-	-	-	0.4831	-
	MAP	-	-	-	-	0.4935

Figure 43. The performance metrics of fine-tuned int/float/multilingual-E5-small embedding for the Sherlock Holmes Dataset on the test data.

The Figure 43 demonstrates that the fine-tuned int/float/multilingual-E5-small outperforms all the models considered until now, including the fine-tuned BAAI/bge-small-en-v1.5. The NDCG score where K = 10 has increased 3% from 0.5074 to 0.5227 when compared to the result of the fine-tuned BAAI/bge-small-en-v1.5 as illustrated in Figure 38. The accuracy, precision and recall values also showed an increase from 1 - 5 % when considering the K values from 1 to 10.

Figure 44. The pictorial representation of the vectors represented on 2-dimensional plane for the fine-tuned int/float/multilingual-E5-small.

Considering the improvement in performance the Figure 44 portrays the pictorial representation of this. Here as can be observed the query embedding and the result are considerably close when compared to the initial fine-tuned BAAI/bge-small-en-v1.5 illustrated in Figure 39.

Figure 45. Comparison of vectors of BAAI/bge-small-en-v1.5 and int/float/multilingual-E5-small embeddings fine-tuned when plotted on a 2-dimensional plane with the query "Who is Sherlock Holmes ?" highlighted in red X and the retrieved documents highlighted using green circles.

The Figure 45 clarifies that selecting the right embedding for the domain is critical, as evident from the figure that the fine-tuned multilingual model int/float/multilingual-E5-small has better accuracy in determining the correct/relevant documents as a result than that of the BAAI/bge-small-en-v1.5.

4.3.2. Cross-Encoder evaluation with and without Multi-query expansion

The multi-query expansion as illustrated in the section 3.3.3 expands the search space by creating multiple queries similar to the original user query to fetch documents that can be related to the question but may be phrased differently in the documents and thereby provide the related documents in the retrieval phase of the RAG by checking the embeddings that are close enough to the altered queries. Along with the cross-encoders as described in the section 3.3.4 helps achieve even finer results from the database even though a bit slower, but with high accuracy.

context_precision	faithfulness	answer_relevancy	context_recall
0.15079365	0.42357456	0.5287953	0.86728088

Table 2. RAGAS metrics with cross-encoder re-ranking and without multi-query expansion and int/float/multilingual-E5-small.

context_precision	faithfulness	answer_relevancy	context_recall
0.15873016	0.44818182	0.53343058	0.87026892

Table 3. RAGAS metrics with cross-encoder re-ranking and with multi-query expansion and int/float/multilingual-E5-small.

The tables Table 2 and Table 3 illustrates that the result can be improved considerably with the application of multi-query expansion and the application of cross-encoder after the bi-encoder application once the fine-tuned int/float/multilingual-E5-small embedding is used. When comparing with Table 1, the improvement in the results for all the RAGAS metrics can be observed.

To ensure the robustness of our results and confirm they are not due to chance, we evaluate the following metrics: NDCG@K, MAP, Precision@K, and Recall@K where K = 3. The tables

NDCG	MAP	Precision@K	Recall@K
0.43041579	0.45047393	0.26530612	0.42531915

Table 4. Performance metrics with cross-encoder re-ranking and without multi-query expansion and int/float/multilingual-E5-small embedding. Here K = 3

NDCG	MAP	Precision@K	Recall@K
0.46337701	0.47730937	0.27536232	0.46677632

Table 5. Performance metrics with cross-encoder re-ranking and with multi-query expansion and the fine-tuned int/float/multilingual-E5-small embedding. Here K = 3

The results are as shown Table 4 and Table 5 which illustrate the performance metrics of cross-encoder re-ranking and multi-query expansion verifying the findings that the cross-encoder and the multi-query expansion with the application of int/float/multilingual-E5-small embedding improves the NDCG, MAP, Precision@K and Recall@K where K = 3 metrics.

4.4. Token optimization techniques

As described in the Literature Review section 2.5.5, the token optimization techniques applied here are Byte-Pair Encoding (BPE), Sentencepiece, Wordpiece and each of their combination with Latent Semantic Analysis (LSA) for summarizing the text by highlighting important words and sentences and eliminating unnecessary ones. These results are evaluated using RAGAS metrics as well as the time taken for each of these operations along with the average number of tokens generated for each question answer pair in the Sherlock dataset. The results presented here are an average of all the individual records.

4.4.1. Byte-Pair Encoding (BPE)

The results are evaluated counting the number of tokens until the number of merges reaches 20, which would in turn reduce the number of tokens considerably. This is reflected in its strong performance across most metrics.

The RAGAS evaluation results of the application of BPE on the dataset is demonstrated in the Table 6.

context_precision	faithfulness	answer_relevancy	context_recall	Token count before Optimization	Token count after Optimization	Time taken for optimization
0.13492063	0.40212766	0.62119522	0.86628486	1397	157	0.000725259

Table 6. Byte Pair Encoding evaluated with RAGAS metrics.

Byte Pair Encoding (BPE) achieved the highest overall performance in terms of answer relevancy and faithfulness compared to the other methods. With a context precision score of 0.1349, BPE does a decent job of ensuring that the most relevant information appears near the top of the retrieved context, which is important for generating accurate answers.

The faithfulness score of 0.4021 suggests that BPE is quite reliable when it comes to producing answers that are factually consistent with the provided context. This means the

answers generated using BPE generally stick to the facts and don't stray too far from the original content.

The standout feature of BPE is its answer relevancy score of 0.6212, which is the highest among all the methods. This high score indicates that BPE is particularly good at generating answers that directly address the prompt, without missing key details or repeating unnecessary information. This makes BPE a strong choice for tasks where the accuracy and completeness of responses are critical.

The optimization process significantly cut the token count from 1,397 to around 158 in under 0.000725259 seconds, showing a major boost in efficiency. This reduction indicates that BPE can compress information without losing too much of the key content. Even with fewer tokens, the context recall score of 0.8663 shows that BPE still captures most of the important information from the original context. This balance between cutting down on tokens and retaining crucial details is important, as it means BPE doesn't just focus on being concise but also ensures the responses remain accurate and relevant. The time taken for BPE optimization is also the least among all the optimization mechanisms. This makes BPE a strong choice, particularly in situations where token limits are tight, like in real-time applications or when working with models that have strict token constraints.

4.4.2. BPE with Latent Semantic Analysis Summarizer

The Table 7 shows the RAGAS evaluation metrics results for BPE with LSA summarizer

context_precision	faithfulness	answer_relevancy	context_recall	Token count before Optimization	Token count after Optimization	Time taken for optimization
0.13095238	0.34854167	0.2159347	0.86628486	741	97	0.000505909

Table 7. Byte-Pair Encoding with LSA evaluated with RAGAS metrics

Adding Latent Semantic Analysis (LSA) to BPE results in some noticeable changes. LSA helps simplify text by reducing its complexity and capturing underlying semantic relationships, but it can also lead to a loss of important details.

The slight drop in context precision to 0.1309 suggests that LSA might make it harder to prioritize the most relevant information, possibly because it abstracts some of the details that were easier to rank higher with BPE alone.

The faithfulness score of 0.3485 is a bit lower than BPE alone, indicating that LSA might introduce some inconsistencies, making the generated answers less aligned with the original facts. This could be because LSA simplifies the text too much, leading to a slight loss of factual accuracy.

The most significant impact of adding LSA to BPE is seen in the answer relevancy score, which drops to 0.2159. This suggests that while LSA helps in generalizing the content, it can also dilute the relevance of the answers, making them less specific and less directly related to the prompts.

Despite these changes, the token count dropped significantly from 741 to just 97 in under 0.000505909 seconds after adding LSA to BPE. This is even faster than the BPE. This reduction highlights how efficiently LSA and BPE work together to compress text. However, this efficiency comes with a trade-off, as seen in the slight dip in context precision and answer relevancy. While fewer tokens mean a more concise output, it also suggests that some important details might be getting lost in the process, leading to lower faithfulness and relevancy scores.

4.4.3. Sentencepiece

Sentencepiece which is a more flexible subword encoding method, showed lower faithfulness and answer relevancy compared to BPE. However, its performance in answer relevancy was still notably better than BPE combined with LSA. The drop in faithfulness compared to BPE suggests that Sentencepiece may struggle with maintaining factual consistency in the generated answers as illustrated in Table 8.

context_precision	faithfulness	answer_relevancy	context_recall	Token count before Optimization	Token count after Optimization	Time taken for optimization
0.13095238	0.29413717	0.49645385	0.86628486	3497	2328	0.002064672

Table 8. Sentencepiece evaluated with RAGAS metrics

The context precision score of 0.1309, similar to BPE with LSA, indicates that Sentencepiece is capable of identifying and prioritizing relevant items in the context, but it doesn't outperform BPE in this area.

With a faithfulness score of 0.2941, Sentencepiece shows a moderate ability to keep answers consistent with the context. This lower score compared to BPE suggests that Sentencepiece might introduce some ambiguity, making it a bit harder to maintain the exact details needed for factual accuracy.

The answer relevancy score of 0.4965, while lower than BPE's, is still respectable and suggests that Sentencepiece can produce relevant answers, though they might not be as precise or detailed. This makes Sentencepiece a good middle-ground option for generating answers that balance generalization and specificity.

Despite starting with a higher token count of 3,497, Sentencepiece successfully reduced this to 2,328 after optimization in under 0.002064672 seconds. While this isn't as dramatic a cut as BPE's, it still shows a notable improvement in efficiency. Importantly, Sentencepiece keeps a high context recall score of 0.8663, meaning it retains a lot of the key information even with fewer tokens. This suggests that although Sentencepiece doesn't compress as aggressively as BPE, it manages to strike a good balance between reducing token count and keeping the important details intact. This makes it a solid option when a more moderate approach to token reduction is needed.

4.4.1. Sentencepiece with LSA

context_precision	faithfulness	answer_relevancy	context_recall	Token count before Optimization	Token count after Optimization	Time taken for optimization
0.13095238	0.30331858	0.4241718	0.86628486	2265	1232	0.001143125

Table 9. Sentencepiece with LSA evaluated with RAGAS metrics

The Table 9 shows applying LSA to Sentencepiece shows some interesting effects. While the context precision remains the same as with Sentencepiece alone, LSA slightly improves faithfulness, raising the score to 0.3033. This suggests that LSA might help in filtering out some noise, leading to answers that are more factually consistent.

However, the answer relevancy score of 0.4242 indicates that LSA can also reduce the relevance of the answers. This is likely due to the abstraction process, which, while improving consistency, might also strip away some of the specific details that make answers directly relevant to the prompt.

Applying LSA reduced the token count from 2,265 to 1,232 in under 0.001143125 seconds. This drop suggests that LSA effectively compresses the information, potentially filtering out less relevant content. Despite this reduction, the high context recall score of 0.8663 indicates that LSA still preserves most of the crucial details from the original context. So while LSA improves efficiency in token usage and maintains good context coverage, there's a trade-off with answer relevancy, likely due to the abstraction process that can strip away specific details. While the LSA offers speed in optimization it gives away a lot of the relevant information.

4.4.2. Wordpiece

Wordpiece, similar to Sentencepiece, is another subword tokenization method that showed relatively balanced performance. Its faithfulness and answer relevancy scores were lower than BPE but higher than Sentencepiece. The moderate performance in these metrics suggests that Wordpiece provides a good trade-off between maintaining context and generating relevant answers.

context_precision	faithfulness	answer_relevancy	context_recall	Token count before Optimization	Token count after Optimization	Time taken for optimization
0.13095238	0.32599119	0.55996955	0.86628486	1067	608	0.00531563

Table 10. Wordpiece evaluated with RAGAS metrics

As illustrated in Table 10 the context precision score of 0.1309 is consistent with the other methods that don't use LSA, showing that Wordpiece is similarly competent in ranking relevant context items but doesn't offer an edge over BPE.

With a faithfulness score of 0.3260, Wordpiece does a fairly good job of maintaining the factual consistency of the answers. This makes it a reliable option when you need a balance between keeping context integrity and allowing for flexibility in handling different linguistic patterns.

The answer relevancy score of 0.5600 is second only to BPE, making Wordpiece an effective choice for generating relevant and coherent answers. This high score suggests that Wordpiece is well-suited for tasks where producing detailed and pertinent responses is important.

The token count for Wordpiece dropped from 1,067 to 608 after optimization completed under 0.00531563 seconds, showing a clear boost in efficiency. Even with this reduction, Wordpiece still keeps a strong context recall score of 0.8663, much like BPE. This means that Wordpiece is good at holding onto the important information from the original context, even with fewer tokens. Being able to reduce token usage while still maintaining context is key, especially in scenarios with tight token limits. This makes Wordpiece a solid option for generating precise and relevant responses.

4.4.3. Wordpiece with LSA

The addition of LSA to Wordpiece improved faithfulness to a level close to BPE, indicating better factual consistency. However, similar to BPE + LSA, the answer relevancy was significantly reduced, suggesting a potential trade-off between maintaining consistency in context and generating relevant answers.

context_precision	faithfulness	answer_relevancy	context_recall	Token count before Optimization	Token count after Optimization	Time taken for optimization
0.13095238	0.38459916	0.21075163	0.86628486	1067	268	0.00278383

Table 11. Wordpiece with LSA evaluated with RAGAS metrics

As demonstrated in Table 11 combining Wordpiece with LSA significantly boosts the faithfulness score to 0.3846, which is close to BPE's level. This suggests that LSA helps Wordpiece in creating answers that are more consistent with the original context, making it a strong option when factual accuracy is a priority.

However, the trade-off is seen in the answer relevancy score, which drops to 0.2108. This decrease indicates that while LSA enhances the factual consistency, it also reduces the relevance of the answers, possibly due to the abstraction process that loses some of the details needed for specificity.

The optimization process also cut the token count from 1,067 to 268 accomplished under 0.002783883 seconds, showing a big jump in efficiency. Even with this reduction, the context recall score stays high at 0.8663, meaning that Wordpiece combined with LSA still keeps a lot of the key information from the original context. This balance between fewer tokens and strong context recall shows that while this approach manages to be efficient, it still maintains important details, even if it does affect answer relevancy.

Overall, Byte Pair Encoding (BPE) stands out as the most effective method for generating answers that are both relevant and factually consistent, and extremely fast making it the best choice for tasks where accuracy and detail are crucial. Adding Latent Semantic Analysis (LSA) generally improves faithfulness but often at the expense of answer relevancy, which could be beneficial in scenarios where maintaining context integrity is more important than the exact relevance of the answers. Sentencepiece and Wordpiece provide balanced alternatives, with Wordpiece showing particular strength in answer relevancy, making it a solid choice when generating coherent and relevant responses is essential. This effectiveness is why advanced models like Llama-2, Llama-3 and Mistral rely on BPE, ensuring their outputs are both accurate and fast, aligning with the demands of high-performance natural language processing tasks.

This chapter presented the results of the experiments conducted as a part of this study

5. Conclusion

This dissertation explored the potential of two open-source large language models, Mistral (7B) and Llama-2 (7B), to determine which would be best suited for a zero-shot recommender system intended for use by airline employees dealing with passenger delays and other issues at the airport. The research also examined how the integration of a knowledge graph (KG) could enhance the system's performance, demonstrating significant improvements in recommendation accuracy. Additionally, the study investigated Retrieval Augmented Generation (RAG) using Llama-3, which outperforms both Llama-2 and Mistral, and looked into various optimization techniques to further better the system's effectiveness. The dissertation also assessed common token optimization methods such as Byte Pair Encoding (BPE), Sentencepiece, and Wordpiece, along with the use of Latent Semantic Analysis (LSA) for text summarization. The findings highlight the importance of choosing the right model, incorporating architectural enhancements, and applying effective optimizations in building a robust zero-shot recommender system for practical use.

The study found that Mistral (7B) is the best open-source LLM to be used as a zero-shot recommender for airlines in both the cases ie. without KG and with KG, when compared to Llama-2 (7B), from the NDCG@3, Recall@3, Precision @3 scores. Since the target for number of generated outputs is 3, we need to look at the top 3 outputs produced. Hence the value 3 was selected as K in NDCG, Recall, Precision etc.

With the adoption of the knowledge graph (KG) into the architecture, there was a considerable improvement in the values of these metrics for both Llama-2 and Mistral. It helped in gaining insight into some of the internal information that was not explicitly specified. In light of this, the performance scores of Mistral (7B) consistently outperformed those of Llama-2 (7B), indicating that Mistral (7B) is more effective in leveraging the KG to enhance the performance of the zero-shot recommender system. This suggests that Mistral (7B) is the superior choice for scenarios where maximizing accuracy and relevancy in recommendations is critical.

This could be largely attributed to the Sliding Window Attention (SWA) mechanism, rolling buffer cache and Grouped-Query Attention (GQA) as described in Section 2.3.2.1 and 2.3.1.1 due to which Mistral can retain information from tokens that are not in its local context, enabling it to process the data faster. The use of the KV cache, in combination with SWA,

further enhances performance by avoiding the storage of all potential K and V values, instead limiting the cache size to match the sentence window. As SWA is one of the key difference in architecture between the two models and given that GQA is utilized only in the Llama-2 34B and 70B models but not in the 7B variant provides the upper hand for Mistral (7B) to be considered the best for Zero-shot recommender for airline industry.

The study also investigated a few of the embeddings commonly used and produced the results with respect to the retrieval metrics. Since this part of the study was using a dataset created and converted to French language, the results showed fine-tuning of a multilingual embedding model such as int/float/multilingual-E5-small with high retrieval MTEB score would perform better than a generic embedding model like Sentence-transformers/all-MiniLM-L6-v2. Also, the application of cross-encoder re-ranking along with bi-encoders, and multi-query expansion of the context helped improve the accuracy of the retrieved results thereby ensuring a better performing RAG system.

Among the token optimization techniques we examined, byte-pair encoding (BPE) stands out as the most effective for improving the language model's performance. Unlike BPE combined with a Latent Semantic Analysis (LSA) summarizer - which can significantly alter the context and query—BPE maintains the integrity of the input, keeping the context and queries more consistent.

This chapter presented the conclusions of the experiments conducted as a part of this study. In the next chapter we present the future works that can be done on top of the existing work. The next chapter will explore potential future directions for research, discussing how to build on this work, highlighting areas that need further investigation, and suggesting new approaches to advance the field and tackle any remaining challenges

6. Future work

This chapter seeks to present opportunities, emphasizing areas for future development and improvement of the current work. Important areas that require further research comprise. For future research, several areas could be explored if additional time were available.

1. Exploring Other Large Language Models: The experiments in this dissertation could be extended by applying the same methods to other large language models (LLMs). Testing different LLMs might reveal models that are better suited for zero-shot recommender systems, offering a broader perspective on model performance in this context.

2. Integrating Knowledge Graphs into RAG Systems: While this study showed the benefits of using a knowledge graph (KG) in the zero-shot recommender system, future research could explore adding KGs to Retrieval Augmented Generation (RAG) systems. This would help determine how such integration impacts performance, particularly in terms of accuracy and speed.

3. Fine-Tuning and Re-Ranking with Cross-Encoders: Future studies could look into fine-tuning the embeddings used in the recommender system and incorporating cross-encoder re-ranking to further improve results. These methods should be evaluated for both their accuracy and their impact on response times.

4. Applying Byte Pair Encoding (BPE): Another area for future research is applying Byte Pair Encoding (BPE) to the recommender system. Experimenting with BPE could provide insights into how this optimization technique affects the system's efficiency and accuracy in generating recommendations.

7. References

Achiam, J., Adler, S., Agarwal, S., Ahmad, L., Akkaya, I., Aleman, F.L., Almeida, D., Altenschmidt, J., Altman, S., Anadkat, S., Avila, R., Babuschkin, I., Balaji, S., Balcom, V., Baltescu, P., Bao, H., Bavarian, M., Belgum, J., Bello, I., Berdine, J., Bernadett-Shapiro, G., Berner, C., Bogdonoff, L., Boiko, O., Boyd, M., Brakman, A.-L., Brockman, G., Brooks, T., Brundage, M., Button, K., Cai, T., Campbell, R., Cann, A., Carey, B., Carlson, C., Carmichael, R., Chan, B., Chang, C., Chantzis, F., Chen, D., Chen, S., Chen, R., Chen, J., Chen, M., Chess, B., Cho, C., Chu, C., Chung, H.W., Cummings, D., Currier, J., Dai, Y., Decareaux, C., Degry, T., Deutsch, N., Deville, D., Dhar, A., Dohan, D., Dowling, S., Dunning, S., Ecoffet, A., Eleti, A., Eloundou, T., Farhi, D., Fedus, L., Felix, N., Fishman, S.P., Forte, J., Fulford, I., Gao, L., Georges, E., Gibson, C., Goel, V., Gogineni, T., Goh, G., Gontijo-Lopes, R., Gordon, J., Grafstein, M., Gray, S., Greene, R., Gross,. (2023) 'GPT-4 Technical Report', *arXiv e-prints*, available at: https://doi.org/10.48550/arXiv.2303.08774(Accessed: [date you accessed]).

Adomavicius, G. and Tuzhilin, A. (2005) 'Toward the next generation of recommender systems: a survey of the state-of-the-art and possible extensions', *IEEE Transactions on Knowledge and Data Engineering*, 17(6), 734–749, available: https://doi.org/10.1109/TKDE.2005.99.

Ainslie, J., Lee-Thorp, J., de Jong, M., Zemlyanskiy, Y., Lebrón, F., and Sanghai, S. (2023) 'GQA: Training Generalized Multi-Query Transformer Models from Multi-Head Checkpoints', available: http://arxiv.org/abs/2305.13245 [accessed 25 Jul 2024].

Airline Reviews Dataset [online] (2024) available: https://www.kaggle.com/datasets/sujalsuthar/airlines-reviews [accessed 31 Jul 2024].

Ba, J.L., Kiros, J.R., and Hinton, G.E. (2016) 'Layer Normalization', available: http://arxiv.org/abs/1607.06450 [accessed 5 Aug 2024].

Bai, Y., Jones, A., Ndousse, K., Askell, A., Chen, A., DasSarma, N., Drain, D., Fort, S., Ganguli, D., Henighan, T., Joseph, N., Kadavath, S., Kernion, J., Conerly, T., El-Showk, S., Elhage, N., Hatfield-Dodds, Z., Hernandez, D., Hume, T., Johnston, S., Kravec, S., Lovitt, L., Nanda, N., Olsson, C., Amodei, D., Brown, T., Clark, J., McCandlish, S., Olah, C., Mann, B., and Kaplan, J. (2022) 'Training a Helpful and Harmless Assistant with Reinforcement Learning from Human Feedback', available: http://arxiv.org/abs/2204.05862 [accessed 29 Jul 2024].

Bao, K., Zhang, J., Zhang, Y., Wenjie, W., Feng, F., and He, X. (2023) 'Large Language Models for Recommendation: Progresses and Future Directions', in *Proceedings of the Annual International ACM SIGIR Conference on Research and Development in Information Retrieval in the Asia Pacific Region*, Presented at the SIGIR-AP '23: Annual International ACM SIGIR Conference on Research and Development in Information Retrieval in the Asia Pacific Region, Beijing China: ACM, 306–309, available: https://doi.org/10.1145/3624918.3629550.

Barnett, S., Kurniawan, S., Thudumu, S., Brannelly, Z., and Abdelrazek, M. (2024) 'Seven Failure Points When Engineering a Retrieval Augmented Generation System', in *2024 IEEE/ACM 3rd International Conference on AI Engineering – Software Engineering for AI (CAIN)*, Presented at the 2024 IEEE/ACM 3rd International Conference on AI Engineering – Software Engineering for AI (CAIN), 194–199, available: https://ieeexplore.ieee.org/document/10556182/ [accessed 28 Jul 2024].

Brown, T., Mann, B., Ryder, N., Subbiah, M., Kaplan, J.D., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020) 'Language Models are Few-Shot Learners', in *Advances in Neural Information Processing Systems*, Curran Associates, Inc., 1877–1901, available: https://papers.nips.cc/paper_files/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html [accessed 25 Jul 2024].

Cai, H., Zheng, V.W., and Chang, K.C.-C. (2018) 'A Comprehensive Survey of Graph Embedding: Problems, Techniques, and Applications', *IEEE Transactions on Knowledge and Data Engineering*, 30(9), 1616–1637, available: https://doi.org/10.1109/TKDE.2018.2807452.

Chen, B. and Bertozzi, A.L. (2023) 'AutoKG: Efficient Automated Knowledge Graph Generation for Language Models', in *2023 IEEE International Conference on Big Data (BigData)*, Presented at the 2023 IEEE International Conference on Big Data (BigData), Sorrento, Italy: IEEE, 3117–3126, available: https://doi.org/10.1109/BigData59044.2023.10386454.

Chen, J., Xiao, S., Zhang, P., Luo, K., Lian, D., and Liu, Z. (2024) 'BGE M3-Embedding: Multi-Lingual, Multi-Functionality, Multi-Granularity Text Embeddings Through Self-Knowledge Distillation', available: http://arxiv.org/abs/2402.03216 [accessed 28 Jul 2024].

Chiu, J. and Shinzato, K. (2022) 'Cross-Encoder Data Annotation for Bi-Encoder Based Product Matching', in Li, Y. and Lazaridou, A., eds., *Proceedings of the 2022 Conference on Empirical Methods in Natural Language Processing: Industry Track*, Presented at the EMNLP 2022, Abu Dhabi, UAE: Association for Computational Linguistics, 161–168, available: https://doi.org/10.18653/v1/2022.emnlp-industry.16.

Conneau, A., Khandelwal, K., Goyal, N., Chaudhary, V., Wenzek, G., Guzmán, F., Grave, E., Ott, M., Zettlemoyer, L., and Stoyanov, V. (2020) 'Unsupervised Cross-lingual Representation Learning at Scale', in Jurafsky, D., Chai, J., Schluter, N. and Tetreault, J., eds., *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, Presented at the ACL 2020, Online: Association for Computational Linguistics, 8440–8451, available: https://doi.org/10.18653/v1/2020.acl-main.747.

Deerwester, S., Dumais, S.T., Furnas, G.W., Landauer, T.K., and Harshman, R. (1990) 'Indexing by latent semantic analysis', *Journal of the American Society for Information Science*, 41(6), 391–407, available: https://doi.org/10.1002/(SICI)1097-4571(199009)41:6<391::AID-ASI1>3.0.CO;2-9.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019a) 'BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding', available: http://arxiv.org/abs/1810.04805 [accessed 4 Aug 2024].

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2019b) 'BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding', available: http://arxiv.org/abs/1810.04805 [accessed 4 Aug 2024].

Dhinakaran, A. (2023) Demystifying NDCG [online], *Medium*, available: https://towardsdatascience.com/demystifying-ndcg-bee3be58cfe0 [accessed 13 Aug 2024].

Dubey, A., Jauhri, A., Pandey, A., Kadian, A., Al-Dahle, A., Letman, A., Mathur, A., Schelten, A., Yang, A., Fan, A., Goyal, A., Hartshorn, A., Yang, A., Mitra, A., Sravankumar, A., Korenev,

A., Hinsvark, A., Rao, A., Zhang, A., Rodriguez, A., Gregerson, A., Spataru, A., Roziere, B., Biron, B., Tang, B., Chern, B., Caucheteux, C., Nayak, C., Bi, C., Marra, C., McConnell, C., Keller, C., Touret, C., Wu, C., Wong, C., Ferrer, C.C., Nikolaidis, C., Allonsius, D., Song, D., Pintz, D., Livshits, D., Esiobu, D., Choudhary, D., Mahajan, D., Garcia-Olano, D., Perino, D., Hupkes, D., Lakomkin, E., AlBadawy, E., Lobanova, E., Dinan, E., Smith, E.M., Radenovic, F., Zhang, F., Synnaeve, G., Lee, G., Anderson, G.L., Nail, G., Mialon, G., Pang, G., Cucurell, G., Nguyen, H., Korevaar, H., Xu, H., Touvron, H., Zarov, I., Ibarra, I.A., Kloumann, I., Misra, I., Evtimov, I., Copet, J., Lee, J., Geffert, J., Vranes, J., Park, J., Mahadeokar, J., Shah, J., van der Linde, J., Billock, J., Hong, J., Lee, J., Fu, J., Chi, J., Huang, J., Liu, J., Wang, J., Yu, J., Bitton, J., Spisak, J., Park, J., Rocca, J., Johnstun, J., Saxe, J., Jia, J., Alwala, K.V., Upasani, K., Plawiak, K., Li, K., Heafield, K., Stone, K., El-Arini, K., Iyer, K., Malik, K., Chiu, K., Bhalla, K., Rantala-Yeary, L., van der Maaten, L., Chen, L., Tan, L., Jenkins, L., Martin, L., Madaan, L., Malo, L., Blecher, L., Landzaat, L., de Oliveira, L., Muzzi, M., Pasupuleti, M., Singh, M., Paluri, M., Kardas, M., Oldham, M., Rita, M., Pavlova, M., Kambadur, M., Lewis, M., Si, M., Singh, M.K., Hassan, M., Goyal, N., Torabi, N., Bashlykov, N., Bogoychev, N., Chatterji, N., Duchenne, O., Çelebi, O., Alrassy, P., Zhang, P., Li, P., Vasic, P., Weng, P., Bhargava, P., Dubal, P., Krishnan, P., Koura, P.S., Xu, P., He, Q., Dong, Q., Srinivasan, R., Ganapathy, R., Calderer, R., Cabral, R.S., Stojnic, R., Raileanu, R., Girdhar, R., Patel, R., Sauvestre, R., Polidoro, R., Sumbaly, R., Taylor, R., Silva, R., Hou, R., Wang, R., Hosseini, S., Chennabasappa, S., Singh, S., Bell, S., Kim, S.S., Edunov, S., Nie, S., Narang, S., Raparthy, S., Shen, S., Wan, S., Bhosale, S., Zhang, S., Vandenhende, S., Batra, S., Whitman, S., Sootla, S., Collot, S., Gururangan, S., Borodinsky, S., Herman, T., Fowler, T., Sheasha, T., Georgiou, T., Scialom, T., Speckbacher, T., Mihaylov, T., Xiao, T., Karn, U., Goswami, V., Gupta, V., Ramanathan, V., Kerkez, V., Gonguet, V., Do, V., Vogeti, V., Petrovic, V., Chu, W., Xiong, W., Fu, W., Meers, W., Martinet, X., Wang, X., Tan, X.E., Xie, X., Jia, X., Wang, X., Goldschlag, Y., Gaur, Y., Babaei, Y., Wen, Y., Song, Y., Zhang, Y., Li, Y., Mao, Y., Coudert, Z.D., Yan, Z., Chen, Z., Papakipos, Z., Singh, A., Grattafiori, A., Jain, A., Kelsey, A., Shajnfeld, A., Gangidi, A., Victoria, A., Goldstand, A., Menon, A., Sharma, A., Boesenberg, A., Vaughan, A., Baevski, A., Feinstein, A., Kallet, A., Sangani, A., Yunus, A., Lupu, A., Alvarado, A., Caples, A., Gu, A., Ho, A., Poulton, A., Ryan, A., Ramchandani, A., Franco, A., Saraf, A., Chowdhury, A., Gabriel, A., Bharambe, A., Eisenman, A., Yazdan, A., James, B., Maurer, B., Leonhardi, B., Huang, B., Loyd, B., De Paola, B., Paranjape, B., Liu, B., Wu, B., Ni, B., Hancock, B., Wasti, B., Spence, B., Stojkovic, B., Gamido, B., Montalvo, B., Parker, C., Burton, C., Mejia, C., Wang, C., Kim, C., Zhou, C., Hu, C., Chu, C.-H., Cai, C., Tindal, C., Feichtenhofer, C., Civin, D., Beaty, D., Kreymer, D., Li, D., Wyatt, D., Adkins, D., Xu, D., Testuggine, D., David, D., Parikh, D., Liskovich, D., Foss, D., Wang, D., Le, D., Holland, D., Dowling, E., Jamil, E., Montgomery, E., Presani, E., Hahn, E., Wood, E., Brinkman, E., Arcaute, E., Dunbar, E., Smothers, E., Sun, F., Kreuk, F., Tian, F., Ozgenel, F., Caggioni, F., Guzmán, F., Kanayet, F., Seide, F., Florez, G.M., Schwarz, G., Badeer, G., Swee, G., Halpern, G., Thattai, G., Herman, G., Sizov, G., Guangyi, Zhang, Lakshminarayanan, G., Shojanazeri, H., Zou, H., Wang, H., Zha, H., Habeeb, H., Rudolph, H., Suk, H., Aspegren, H., Goldman, H., Molybog, I., Tufanov, I., Veliche, I.-E., Gat, I., Weissman, J., Geboski, J., Kohli, J., Asher, J., Gaya, J.-B., Marcus, J., Tang, J., Chan, J., Zhen, J., Reizenstein, J., Teboul, J., Zhong, J., Jin, J., Yang, J., Cummings, J., Carvill, J., Shepard, J., McPhie, J., Torres, J., Ginsburg, J., Wang, J., Wu, K., U, K.H., Saxena, K., Prasad, K., Khandelwal, K., Zand, K., Matosich, K., Veeraraghavan, K., Michelena, K., Li, K., Huang, K., Chawla, K., Lakhotia, K., Huang, K., Chen, L., Garg, L., A, L., Silva, L., Bell, L., Zhang, L., Guo, L., Yu, L., Moshkovich, L., Wehrstedt, L., Khabsa, M., Avalani, M., Bhatt, M., Tsimpoukelli, M., Mankus, M., Hasson, M., Lennie, M., Reso, M., Groshev, M., Naumov, M., Lathi, M., Keneally, M., Seltzer, M.L., Valko, M., Restrepo, M.,

Patel, M., Vyatskov, M., Samvelyan, M., Clark, M., Macey, M., Wang, M., Hermoso, M.J., Metanat, M., Rastegari, M., Bansal, M., Santhanam, N., Parks, N., White, N., Bawa, N., Singhal, N., Egebo, N., Usunier, N., Laptev, N.P., Dong, N., Zhang, N., Cheng, N., Chernoguz, O., Hart, O., Salpekar, O., Kalinli, O., Kent, P., Parekh, P., Saab, P., Balaji, P., Rittner, P., Bontrager, P., Roux, P., Dollar, P., Zvyagina, P., Ratanchandani, P., Yuvraj, P., Liang, Q., Alao, R., Rodriguez, R., Ayub, R., Murthy, R., Nayani, R., Mitra, R., Li, R., Hogan, R., Battey, R., Wang, R., Maheswari, R., Howes, R., Rinott, R., Bondu, S.J., Datta, S., Chugh, S., Hunt, S., Dhillon, S., Sidorov, S., Pan, S., Verma, S., Yamamoto, S., Ramaswamy, S., Lindsay, S., Lindsay, S., Feng, S., Lin, S., Zha, S.C., Shankar, S., Zhang, S., Zhang, S., Wang, S., Agarwal, S., Sajuyigbe, S., Chintala, S., Max, S., Chen, S., Kehoe, S., Satterfield, S., Govindaprasad, S., Gupta, S., Cho, S., Virk, S., Subramanian, S., Choudhury, S., Goldman, S., Remez, T., Glaser, T., Best, T., Kohler, T., Robinson, T., Li, T., Zhang, T., Matthews, T., Chou, T., Shaked, T., Vontimitta, V., Ajayi, V., Montanez, V., Mohan, V., Kumar, V.S., Mangla, V., Ionescu, V., Poenaru, V., Mihailescu, V.T., Ivanov, V., Li, W., Wang, W., Jiang, W., Bouaziz, W., Constable, W., Tang, X., Wang, X., Wu, X., Wang, X., Xia, X., Wu, X., Gao, X., Chen, Y., Hu, Y., Jia, Y., Qi, Y., Li, Y., Zhang, Y., Zhang, Y., Adi, Y., Nam, Y., Yu, Wang, Hao, Y., Qian, Y., He, Y., Rait, Z., DeVito, Z., Rosnbrick, Z., Wen, Z., Yang, Z., and Zhao, Z. (2024) 'The Llama 3 Herd of Models', available: http://arxiv.org/abs/2407.21783 [accessed 11 Aug 2024].

Dumais, S.T. (1991) 'Improving the retrieval of information from external sources', *Behavior Research Methods, Instruments, & Computers*, 23(2), 229–236, available: https://doi.org/10.3758/BF03203370.

Es, S., James, J., Espinosa-Anke, L., and Schockaert, S. (2023) 'RAGAS: Automated Evaluation of Retrieval Augmented Generation', available: http://arxiv.org/abs/2309.15217 [accessed 13 Aug 2024].

Fan, W., Ma, Y., Yin, D., Wang, J., Tang, J., and Li, Q. (2019) 'Deep social collaborative filtering', in *Proceedings of the 13th ACM Conference on Recommender Systems*, Presented at the RecSys '19: Thirteenth ACM Conference on Recommender Systems, Copenhagen Denmark: ACM, 305–313, available: https://doi.org/10.1145/3298689.3347011.

Gao, M., Li, J.-Y., Chen, C.-H., Li, Y., Zhang, J., and Zhan, Z.-H. (2023) 'Enhanced Multi-Task Learning and Knowledge Graph-Based Recommender System', *IEEE Transactions on Knowledge and Data Engineering*, 35(10), 10281–10294, available: https://doi.org/10.1109/TKDE.2023.3251897.

Guo, Q., Zhuang, F., Qin, C., Zhu, H., Xie, X., Xiong, H., and He, Q. (2022) 'A Survey on Knowledge Graph-Based Recommender Systems', *IEEE Transactions on Knowledge and Data Engineering*, 34(8), 3549–3568, available: https://doi.org/10.1109/TKDE.2020.3028705.

He, K., Zhang, X., Ren, S., and Sun, J. (2016) 'Deep Residual Learning for Image Recognition', in *2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR)*, Presented at the 2016 IEEE Conference on Computer Vision and Pattern Recognition (CVPR), Las Vegas, NV, USA: IEEE, 770–778, available: https://doi.org/10.1109/CVPR.2016.90.

Hochreiter, S. (1998) 'The Vanishing Gradient Problem During Learning Recurrent Neural Nets and Problem Solutions', *International Journal of Uncertainty, Fuzziness and Knowledge-Based Systems*, 06(02), 107–116, available: https://doi.org/10.1142/S0218488598000094.

Holland, G. (2023) ‘How to Create Dataset with Python Faker’, *AbsentData*, available: https://absentdata.com/python/how-to-create-dataset-with-python-faker/ [accessed 31 Jul 2024].

Introducing Meta Llama 3: The Most Capable Openly Available LLM to Date [online] (2024) *Meta AI*, available: https://ai.meta.com/blog/meta-llama-3/ [accessed 1 Aug 2024].

Jiang, A.Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D.S., Casas, D. de las, Bressand, F., Lengyel, G., Lample, G., Saulnier, L., Lavaud, L.R., Lachaux, M.-A., Stock, P., Scao, T.L., Lavril, T., Wang, T., Lacroix, T., and Sayed, W.E. (2023) ‘Mistral 7B’, available: http://arxiv.org/abs/2310.06825 [accessed 26 Jul 2024].

Koren, Y., Bell, R., and Volinsky, C. (2009) ‘Matrix Factorization Techniques for Recommender Systems’, *Computer*, 42(8), 30–37, available: https://doi.org/10.1109/MC.2009.263.

Kudo, T. and Richardson, J. (2018) ‘SentencePiece: A simple and language independent subword tokenizer and detokenizer for Neural Text Processing’, in Blanco, E. and Lu, W., eds., *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, Brussels, Belgium: Association for Computational Linguistics, 66–71, available: https://doi.org/10.18653/v1/D18-2012.

Kukreja, S., Kumar, T., Bharate, V., Purohit, A., Dasgupta, A., and Guha, D. (2023) ‘Vector Databases and Vector Embeddings-Review’, in *2023 International Workshop on Artificial Intelligence and Image Processing (IWAIIP)*, Presented at the 2023 International Workshop on Artificial Intelligence and Image Processing (IWAIIP), 231–236, available: https://doi.org/10.1109/IWAIIP58158.2023.10462847.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W., Rocktäschel, T., Riedel, S., and Kiela, D. (2021) ‘Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks’, available: http://arxiv.org/abs/2005.11401 [accessed 27 Jul 2024].

Li, X.L. and Liang, P. (2021) ‘Prefix-Tuning: Optimizing Continuous Prompts for Generation’, in Zong, C., Xia, F., Li, W. and Navigli, R., eds., *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, Presented at the ACL-IJCNLP 2021, Online: Association for Computational Linguistics, 4582–4597, available: https://doi.org/10.18653/v1/2021.acl-long.353.

Lipton, Z.C. (n.d.) ‘A Critical Review of Recurrent Neural Networks for Sequence Learning’.

Luo, X., Zhou, M., Li, S., You, Z., Xia, Y., and Zhu, Q. (2016) ‘A Nonnegative Latent Factor Model for Large-Scale Sparse Matrices in Recommender Systems via Alternating Direction Method’, *IEEE Transactions on Neural Networks and Learning Systems*, 27(3), 579–592, available: https://doi.org/10.1109/TNNLS.2015.2415257.

McKenna, N., Li, T., Cheng, L., Hosseini, M.J., Johnson, M., and Steedman, M. (2023) ‘Sources of Hallucination by Large Language Models on Inference Tasks’, available: http://arxiv.org/abs/2305.14552 [accessed 19 Jul 2024].

Metrics | Ragas [online] (2024) available: https://docs.ragas.io/en/latest/concepts/metrics/index.html [accessed 13 Aug 2024].

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013) 'Efficient Estimation of Word Representations in Vector Space', available: http://arxiv.org/abs/1301.3781 [accessed 28 Jul 2024].

Muennighoff, N., Tazi, N., Magne, L., and Reimers, N. (2023) 'MTEB: Massive Text Embedding Benchmark', available: http://arxiv.org/abs/2210.07316 [accessed 8 Aug 2024].

Multiple Negatives Ranking Loss for Sentence Embeddings [online] (2023) available: https://www.youtube.com/watch?v=b_2v9Hpfnbw [accessed 11 Aug 2024].

Özker, U. (2024) 'Advanced RAG Architecture', *Medium*, available: https://ugurozker.medium.com/advanced-rag-architecture-b9f8a26e2608 [accessed 13 Aug 2024].

Peng, C., Xia, F., Naseriparsa, M., and Osborne, F. (2023) 'Knowledge Graphs: Opportunities and Challenges', *Artificial Intelligence Review*, 56(11), 13071–13102, available: https://doi.org/10.1007/s10462-023-10465-9.

Potter, M., Liu, H., Lala, Y., Loanzon, C., and Sun, Y. (2022) 'GRU4RecBE: A Hybrid Session-Based Movie Recommendation System (Student Abstract)', *Proceedings of the AAAI Conference on Artificial Intelligence*, 36(11), 13029–13030, available: https://doi.org/10.1609/aaai.v36i11.21651.

Rafailov, R., Sharma, A., Mitchell, E., Ermon, S., Manning, C.D., and Finn, C. (2023) 'Direct Preference Optimization: Your Language Model is Secretly a Reward Model', available: http://arxiv.org/abs/2305.18290 [accessed 29 Jul 2024].

Ramnarain-Seetohul, V., Bassoo, V., and Rosunally, Y. (2022) 'Work-in-Progress: Computing Sentence Similarity for Short Texts using Transformer models', in *2022 IEEE Global Engineering Education Conference (EDUCON)*, Presented at the 2022 IEEE Global Engineering Education Conference (EDUCON), 1765–1768, available: https://doi.org/10.1109/EDUCON52537.2022.9766649.

Reimers, N. and Gurevych, I. (2019) 'Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks', available: http://arxiv.org/abs/1908.10084 [accessed 29 Jul 2024].

Retriever Models for Open Domain Question-Answering | Pinecone [online] (2024) available: https://www.pinecone.io/learn/series/nlp/retriever-models/ [accessed 12 Aug 2024].

Salton, G., Wong, A., and Yang, C.S. (1975) 'A vector space model for automatic indexing', *Communications of the ACM*, 18(11), 613–620, available: https://doi.org/10.1145/361219.361220.

Sarwar, B., Karypis, G., Konstan, J., and Riedl, J. (2001) 'Item-based collaborative filtering recommendation algorithms', in *Proceedings of the 10th International Conference on World Wide Web*, Presented at the WWW01: Hypermedia Track of the 10th International World Wide Web Conference, Hong Kong Hong Kong: ACM, 285–295, available: https://doi.org/10.1145/371920.372071.

Schick, T. and Schütze, H. (2021) 'It's Not Just Size That Matters: Small Language Models Are Also Few-Shot Learners', available: http://arxiv.org/abs/2009.07118 [accessed 1 Aug 2024].

Schuster, M. and Nakajima, K. (2012) 'Japanese and Korean voice search', in *2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP)*, Presented at

the 2012 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 5149–5152, available: https://doi.org/10.1109/ICASSP.2012.6289079.

Sennrich, R., Haddow, B., and Birch, A. (2016) 'Neural Machine Translation of Rare Words with Subword Units', in Erk, K. and Smith, N.A., eds., *Proceedings of the 54th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, Presented at the ACL 2016, Berlin, Germany: Association for Computational Linguistics, 1715–1725, available: https://doi.org/10.18653/v1/P16-1162.

Sentence-Transformers/All-MiniLM-L6-v2 · Hugging Face [online] (2024) available: https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2 [accessed 13 Aug 2024].

Shazeer, N. (2019) 'Fast Transformer Decoding: One Write-Head is All You Need', available: http://arxiv.org/abs/1911.02150 [accessed 14 Aug 2024].

Shi, Y., Larson, M., and Hanjalic, A. (2014) 'Collaborative Filtering beyond the User-Item Matrix: A Survey of the State of the Art and Future Challenges', *ACM Computing Surveys*, 47(1), 1–45, available: https://doi.org/10.1145/2556270.

Singh, V. (2024) 'Building LLM Applications: Sentence Transformers (Part 3)', *Medium*, available: https://medium.com/@vipra_singh/building-llm-applications-sentence-transformers-part-3-a9e2529f99c1 [accessed 12 Aug 2024].

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., Bikel, D., Blecher, L., Ferrer, C.C., Chen, M., Cucurull, G., Esiobu, D., Fernandes, J., Fu, J., Fu, W., Fuller, B., Gao, C., Goswami, V., Goyal, N., Hartshorn, A., Hosseini, S., Hou, R., Inan, H., Kardas, M., Kerkez, V., Khabsa, M., Kloumann, I., Korenev, A., Koura, P.S., Lachaux, M.-A., Lavril, T., Lee, J., Liskovich, D., Lu, Y., Mao, Y., Martinet, X., Mihaylov, T., Mishra, P., Molybog, I., Nie, Y., Poulton, A., Reizenstein, J., Rungta, R., Saladi, K., Schelten, A., Silva, R., Smith, E.M., Subramanian, R., Tan, X.E., Tang, B., Taylor, R., Williams, A., Kuan, J.X., Xu, P., Yan, Z., Zarov, I., Zhang, Y., Fan, A., Kambadur, M., Narang, S., Rodriguez, A., Stojnic, R., Edunov, S., and Scialom, T. (2023) 'Llama 2: Open Foundation and Fine-Tuned Chat Models', available: http://arxiv.org/abs/2307.09288 [accessed 25 Jul 2024].

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, L., and Polosukhin, I. (2023) 'Attention Is All You Need', available: http://arxiv.org/abs/1706.03762 [accessed 29 Jul 2024].

Wang, H., Liu, X., Fan, W., Zhao, X., Kini, V., Yadav, D., Wang, F., Wen, Z., Tang, J., and Liu, H. (2024) 'Rethinking Large Language Model Architectures for Sequential Recommendations', available: http://arxiv.org/abs/2402.09543 [accessed 18 Jul 2024].

Wang, L. and Lim, E.-P. (2023) 'Zero-Shot Next-Item Recommendation using Large Pretrained Language Models', available: http://arxiv.org/abs/2304.03153 [accessed 20 Aug 2024].

Wang, L., Yang, N., Huang, X., Yang, L., Majumder, R., and Wei, F. (2024) 'Multilingual E5 Text Embeddings: A Technical Report', available: http://arxiv.org/abs/2402.05672 [accessed 29 Jul 2024].

Wang, S., Hu, L., Wang, Y., He, X., Sheng, Q.Z., Orgun, M.A., Cao, L., Ricci, F., and Yu, P.S. (2021) 'Graph Learning based Recommender Systems: A Review', in *Proceedings of the Thirtieth International Joint Conference on Artificial Intelligence*, Presented at the Thirtieth

International Joint Conference on Artificial Intelligence {IJCAI-21}, Montreal, Canada: International Joint Conferences on Artificial Intelligence Organization, 4644–4652, available: https://doi.org/10.24963/ijcai.2021/630.

Wang, Y., Wang, L., and Li, Y. (n.d.) 'A Theoretical Analysis of NDCG Ranking Measures'.

Wei, J., Wang, X., Schuurmans, D., Bosma, M., Ichter, B., Xia, F., Chi, E., Le, Q., and Zhou, D. (2023) 'Chain-of-Thought Prompting Elicits Reasoning in Large Language Models', available: http://arxiv.org/abs/2201.11903 [accessed 18 Jul 2024].

Wu, Y., Schuster, M., Chen, Z., Le, Q.V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., Klingner, J., Shah, A., Johnson, M., Liu, X., Kaiser, Ł., Gouws, S., Kato, Y., Kudo, T., Kazawa, H., Stevens, K., Kurian, G., Patil, N., Wang, W., Young, C., Smith, J., Riesa, J., Rudnick, A., Vinyals, O., Corrado, G., Hughes, M., and Dean, J. (2016) 'Google's Neural Machine Translation System: Bridging the Gap between Human and Machine Translation', available: http://arxiv.org/abs/1609.08144 [accessed 4 Aug 2024].

Yang, X. and Mao, K. (2017) 'Task Independent Fine Tuning for Word Embeddings', *IEEE/ACM Transactions on Audio, Speech, and Language Processing*, 25(4), 885–894, available: https://doi.org/10.1109/TASLP.2016.2644863.

Yawalkar, P., Birari, A., Bharathan, G., Vakayil, S., and Sharma, R. (2022) 'Subscriber Preference and Content Consumption Pattern toward OTT platform: An Opinion Mining', in *2022 International Conference on Trends in Quantum Computing and Emerging Business Technologies (TQCEBT)*, Presented at the 2022 International Conference on Trends in Quantum Computing and Emerging Business Technologies (TQCEBT), 1–7, available: https://doi.org/10.1109/TQCEBT54229.2022.10041265.

Zhang, F., Yuan, N.J., Lian, D., Xie, X., and Ma, W.-Y. (2016) 'Collaborative Knowledge Base Embedding for Recommender Systems', in *Proceedings of the 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining*, Presented at the KDD '16: The 22nd ACM SIGKDD International Conference on Knowledge Discovery and Data Mining, San Francisco California USA: ACM, 353–362, available: https://doi.org/10.1145/2939672.2939673.

Zhang, Q., Lu, J., and Jin, Y. (2021) 'Artificial intelligence in recommender systems', *Complex & Intelligent Systems*, 7(1), 439–457, available: https://doi.org/10.1007/s40747-020-00212-w.

Zhang, S., Yao, L., Sun, A., and Tay, Y. (2020) 'Deep Learning Based Recommender System: A Survey and New Perspectives', *ACM Computing Surveys*, 52(1), 1–38, available: https://doi.org/10.1145/3285029.

Zhao, Z., Fan, W., Li, J., Liu, Y., Mei, X., Wang, Y., Wen, Z., Wang, F., Zhao, X., Tang, J., and Li, Q. (2024) 'Recommender Systems in the Era of Large Language Models (LLMs)', available: http://arxiv.org/abs/2307.02046 [accessed 18 Jul 2024].

Zhen, Y., Li, W.-J., and Yeung, D.-Y. (2009) 'TagiCoFi: tag informed collaborative filtering', in *Proceedings of the Third ACM Conference on Recommender Systems*, Presented at the RecSys '09: Third ACM Conference on Recommender Systems, New York New York USA: ACM, 69–76, available: https://doi.org/10.1145/1639714.1639727.