DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Do Graph-based Approaches Outperform Vector-based Approaches in Retrieval Augmented Generation for Complex Question Answering? - A Study Using Wikipedia and the Mintaka Dataset

Philippe Saadé

DEPARTMENT OF INFORMATICS

TECHNISCHE UNIVERSITÄT MÜNCHEN

Master's Thesis in Informatics

Do Graph-based Approaches Outperform Vector-based Approaches in Retrieval Augmented Generation for Complex Question Answering? - A Study Using Wikipedia and the Mintaka Dataset

Author:	Philippe Saadé
Supervisor:	Prof. Dr. Florian Matthes
Advisor:	Tim Schopf
Submission Date:	15/05/2024

I confirm that this master's thesis in informatics is my own work and I have documented all sources and material used.

Munich, 15/05/2024 Philippe Saadé

AI Assistant Usage Disclosure

Introduction

Performing work or conducting research at the Chair of Software Engineering for Business Information Systems (sebis) at TUM often entails dynamic and multi-faceted tasks. At sebis, we promote the responsible use of *AI Assistants* in the effective and efficient completion of such work. However, in the spirit of ethical and transparent research, we require all student researchers working with sebis to disclose their usage of such assistants.

For examples of correct and incorrect AI Assistant usage, please refer to the original, unabridged version of this form, located at this link.

Use of *AI Assistants* for Research Purposes

I have used AI Assistant(s) for the purposes of my research as part of this thesis.

Yes ◉ No ☐

Explanation: In this study, AI has been used for writing assistance and for proofreading. We use ChatGPT and Grammarly to correct grammar and spelling mistakes, as well as to reformulate our own text for clarity.

I confirm in signing below, that I have reported all usage of AI Assistants for my research, and that the report is truthful and complete.

Location, Date	Author

Abstract

Large Language Models (LLMs) have recently demonstrated substantial language understanding and almost human-like conversation capabilities. However, relying solely on LLMs for question-answering tasks is insufficient as they are limited by their training data and prone to producing inaccurate information. Retrieval-Augmented Generation (RAG) techniques aim to solve those limitations by introducing external knowledge sources, improving the precision of answers generated by LLM. This study evaluates various state-of-the-art RAG techniques in improving LLM performance across different question complexities. Furthermore, it investigates the strengths and weaknesses of RAG systems that retrieve data from vector databases versus graph databases. Our findings indicate that vector-based RAG systems excel in answering general questions and are better suited for LLMs that are less powerful in the processing of complex rules. In contrast, graph-based systems are more effective on complex questions that require data from multiple documents. This research is based on Wikipedia and the Wikidata Knowledge Graph (KG), providing a comprehensive guide for evaluating and selecting the appropriate RAG technique for other domain-specific datasets.

Kurzfassung

Large Language Models haben in letzter Zeit zunehmend ihre Fähigkeit bewiesen, Sprache zu verstehen und auf fast menschenartige Weise zu kommunizieren. Dennoch sind LLMs fehleranfällig und eingeschränkt durch die ihnen zur Verfügung stehenden Trainingsdaten, weshalb ein alleiniger Verlass auf sie bei der Beantwortung gestellter Fragen nicht zu empfehlen ist. Retrieval-Augmented-Generation (RAG)-Techniken zielen darauf ab, diese Einschränkungen durch die Einbindung externer Wissensquellen zu überwinden und die Präzision der von LLMs generierten Antworten zu erhöhen. Diese Studie bewertet die fortschrittlichsten RAG-Techniken hinsichtlich ihrer Fähigkeit, die Leistung von LLMs bei unterschiedlichen Fragekomplexitäten zu steigern. Darüber hinaus untersucht sie die Stärken und Schwächen von RAG-Systemen, die Daten aus Vektordatenbanken im Vergleich zu Graphdatenbanken abrufen. Unsere Ergebnisse zeigen, dass Vektor-basierte RAG-Systeme bei allgemeinen Fragen besonders gute Leistungen zeigen und deshalb besser geeignet sind im Zusammenhang mit LLMs, die weniger leistungsfähig sind in der Verarbeitung komplexer Regeln. Im Gegensatz dazu sind Graph-basierte Systeme bei komplexen Fragen effektiver, die Daten aus mehreren Dokumenten erfordern. Diese Studie stützt sich auf Wikipedia und den WikiData-Wissensgraph, um einen umfassenden Leitfaden für die Bewertung und Auswahl geeigneter RAG-Techniken für andere domänenspezifische Datensätze bereitzustellen.

Contents

Abstract **iv**

Kurzfassung **v**

1 Introduction **1**
1.1 Motivation . . . 1
1.2 Research Questions . . . 2

2 Related Work **4**
2.1 Large Language Models (LLMs) . . . 4
2.2 Information Retrieval (IR) . . . 5
2.3 Retrieval-Augmented Generation . . . 6
2.3.1 Embedding-based Retrieval . . . 6
2.3.2 Graph-based Retrieval . . . 8

3 Methodology **9**
3.1 Embedding-based RAG Techniques . . . 9
3.1.1 Dense Passage Retrieval (DPR) . . . 9
3.2 Graph-based RAG Techniques . . . 11
3.2.1 Cypher Query Generation . . . 11
3.2.2 Knowledge Graph Traversal . . . 12
3.2.3 Triplets Extraction . . . 13
3.3 Evaluation Approaches . . . 13

4 Experimental Setup **15**
4.1 Dataset . . . 15
4.2 Evaluation . . . 15
4.2.1 Evaluation Dataset . . . 15
4.2.2 Evaluation Approaches . . . 18
4.3 Large Language Models (LLMs) . . . 20
4.4 Embedding-based RAG Techniques . . . 21
4.4.1 Dense Passage Retrieval (DPR) . . . 21
4.5 Graph-based RAG Techniques . . . 23
4.5.1 Cypher Query Generation . . . 23
4.5.2 Knowledge Graph Traversal . . . 27
4.5.3 Triplets Extraction . . . 30

5 Results **33**
5.1 Results of the Embedding-based RAG Techniques 33
5.2 Results of the Graph-based RAG Techniques 35
5.3 Results of the Cypher Generation RAG Technique 38
5.4 Retrieval Runtime 40
5.5 Results on the Retrieval 41
5.6 Accuracy over Prompt Size 43

6 Conclusion **49**
6.1 Findings 49
6.2 Limitations 50
6.3 Future Work 50

List of Figures **52**

List of Tables **54**

Acronyms **55**

Bibliography **56**

1 Introduction

Recent advancements in LLMs have marked a significant milestone in the field of Natural Language Processing (NLP), showcasing their powerful capabilities in understanding and generating almost human-like text. The emergence and continuous improvement of LLMs, such as GPT by openAI [1] and BERT by Google [2], have shifted the NLP world, changing our approach to numerous tasks. This advancement paved the way for the development of conversational language models such as ChatGPT and other open-source LLMs like Llama by Touvron, Lavril, Izacard, et al. [3] and Mistral by Jiang, Sablayrolles, Mensch, et al. [4], consequently reaching the mainstream, revolutionizing research methodologies, content creation, and our general approach to text-based communication.

LLMs in their traditional form, whether based on Transformers [5] or Sequence-to-Sequence (Seq2Seq) architectures, retain information in their parameters, which sometimes causes the model to produce inaccurate results or tendencies to "hallucinate". Moreover, introducing new data to a traditional LLM requires fine-tuning, which is computationally expensive and not feasible for real-time data. To address these limitations, RAG, as introduced by P. Lewis, Perez, Piktus, et al. [6], leverages external document retrieval to enhance LLMs' response accuracy. RAG efficiently augments LLMs with relevant information by retrieving documents based on the user queries, using either vector or graph databases. This process makes introducing new data computationally efficient, streamlining the procedure and facilitating real-time data integration. This approach not only significantly enhances the accuracy of generated responses but also introduces a level of adaptability, allowing LLMs to provide informed responses on a broader range of topics, including those not covered during their training process.

1.1 Motivation

This study aims to systematically compare different embedding-based and graph-based RAG techniques to analyze their abilities in augmenting LLMs. Additionally, our goal is to understand how vector and graph database structures influence the performance of RAG systems across various question complexities. This study paves the way for further research into enhancing retrieval approaches and contributing to refining RAG systems. Furthermore, this study seeks to provide comprehensive guidelines and metrics for evaluating retrieval systems, enabling the selection and subsequent tailoring of an appropriate system for specific datasets and applications.

Various RAG techniques have emerged to enhance the accuracy of LLMs, varying in their retrieval process and database structures. While some RAG approaches rely on vector databases and embeddings to collect documents close to user queries in semantic similarity, others leverage the structured nature of graph databases to gather necessary information for the LLM. Vector databases, known for their ease of implementation and ability to handle unstructured data, connect questions to relevant documents in the database through vector similarity. Graph databases, on the other hand, structure data with deeper relational reasoning, storing additional document relationships on top of the textual data. This architecture allows the retrieval process to navigate from one document to another, constructing richer contextual data at each step.

1.2 Research Questions

We aim to answer several research questions that address various RAG methodologies, focusing specifically on the differences in database structures and their impact on augmenting LLMs for question-answering tasks. By answering the four research questions we have formulated, we seek to identify and compare the advantages of embedding- and graph-based RAG systems in supporting state-of-the-art LLMs, particularly within the general domain of Wikidata. The outcomes of this study provide further insights into various RAG systems and offer guidance on constructing and evaluating these systems for domain-specific tasks across different datasets. The research questions are the following:

1- How do vector and graph databases differ in their performance when augmenting LLMs in question-answering tasks?
This question seeks to uncover the advantages and disadvantages of each database structure in augmenting language models. While vector databases are capable of handling unstructured data, we hypothesize that they fail to take advantage of the dataset's full potential, often providing broad information for general questions. Additionally, we hypothesize that graph databases' rich and logical structure improves the LLM's ability to answer complex, rule-intensive questions.

2- What are existing retrieval approaches for RAG methods using vector and graph databases?
This study provides an overview of existing retrieval approaches in RAG systems that use vector or graph databases. The insights include fundamental approaches that serve as a baseline, which can then be adapted based on specific datasets and applications.

3- How can a vector database be aligned with a graph database to include the same information and be comparable in terms of retrieval performance?
This question addresses the challenge of comparing different database structures and highlights the necessity to ensure equality in the information contained within vector and graph databases. This equivalency is crucial for conducting a fair and insightful evaluation between the two structures.

4- How can the quality of question-answering performance be systematically evaluated across different LLM-based RAG systems?
The goal is to develop a robust framework for evaluating the performance of various RAG implementations, mainly focusing on their ability across different question types and complexities. Furthermore, we will provide effective metrics for comparing generated answers by LLMs to the ground truth.

2 Related Work

2.1 Large Language Models (LLMs)

Language models seek to resolve the task of predicting upcoming words given a sequence of preceding words [7]. This task formed the core language understanding and laid the groundwork for more specific NLP tasks. The research on language models dates back to the 1980s with the development of Statistical Language Models (SLM) [8]. These models are based on n-gram probabilistic methods designed to calculate the likelihood of the following word in an incomplete sentence.

The next significant advancement in language models came with the introduction of Neural Networks and the development of Neural-based Language Models (NLM). NLM opened doors for further NLP research by moving from sparse n-gram representations to dense lower-dimensional vector representations of text [9]. Innovations such as Word2Vec by Mikolov, K. Chen, Corrado, and Dean [10] and GloVe by Pennington, Socher, and Manning [11] helped advance this field by introducing efficient techniques for capturing semantic representations of words, improving the ability to understand word similarity and context. As computing became more powerful and accessible, research in Neural Networks continued to evolve.

Large Language Models, typically referring to models based on the Transformer architecture introduced by Vaswani, Shazeer, Parmar, et al. [5], represent another milestone in NLP. Encoder-based models like BERT [2] use Transformers to produce context-aware word embeddings by considering the complete context of sentences. On the other hand, decoder-based models like GPT [1], Llama [3], and PaLM [12] tackle the core of language models by improving on the task of predicting the next word given a set of preceding words. These models were additionally fine-tuned to generate coherent text sequences given a prompt. Seq2Seq models such as BART [13] and T5 [14] include both encoder and decoder components. Unlike decoder-only models, which process inputs in a uni-directional manner, encoder-decoder models encode the input using bi-directional attention mechanisms to understand the context thoroughly. Afterwards, the vector embedding is given to the decoder to generate text, handling tasks such as summarizing, translation, and many other complex language generations.

Despite these advancements, LLMs are resource-intensive and require sizeable computational power for training, which makes real-time data integration and updating impractical. Moreover, as information is contained within static parameters, LLMs are unreliable in maintaining factual accuracy. These limitations have led to further research, including RAG, which

efficiently incorporates relevant information during the generation process [15]. The main goal of RAG is to improve LLM responses, ensuring they are both informed and factually accurate by leveraging curated data.

2.2 Information Retrieval (IR)

Before its usage for augmenting LLMs, Information Retrieval (IR) was introduced to simplify the navigation of large databases by enabling users to access information via text queries. This NLP task has been widely used in programs such as search engines and recommendation systems [16]. The advancements in IR aligned with the evolution of language models with early techniques relying on sparse word counts and n-grams. Notable methods include Boolean Retrieval [17], Term Frequency-Inverse Document Frequency (TF-IDF) [18], and BM25 [19], which attempted to extract relevant documents by matching words within the query.

With the rise of neural networks, IR evolved alongside language models and diverted to dense vector representations. Now, the most common retrieval technique for various NLP tasks is the Dense Passage Retrieval (DPR) by Karpukhin, Oğuz, S. Min, et al. [20], which introduced the use of dense vector embeddings for data retrieval. DPR is a model that employs two encoders, one designed to encode the documents for storage, while the other encodes the questions to link to the document vectors. The approach demonstrates significant improvements over traditional sparse vector embeddings.

Various research has aimed to refine retrieval results by ranking collected data for more accurate results. RankNet by Burges, Shaked, Renshaw, et al. [21] and ListNet by Cao, Qin, T.-Y. Liu, et al. [22] are two foundational techniques designed to reorder the list of retrieved information to improve their relevance. RankNet uses a pairwise approach where a query and two documents are taken as input to determine the more relevant document between the pair. On the other hand, ListNet uses a listwise approach and processes the entire list of retrieved documents along with the query to score each document by its relevance to the query and arrange the data accordingly. Advancements in Deep Neural Networks and Large Language Models enabled cross-encoding techniques for re-ranking documents, as demonstrated by Nogueira and Cho [23]. In this method, a re-ranker model with a cross-encoding mechanism jointly processes the query and documents together for a more accurate evaluation, compared to independent dense vectors' similarity scoring. This approach has been proven to significantly improve retrieval accuracy due to its ability to thoroughly analyze the relationship between the query and documents. However, the re-ranker models are computationally expensive, mainly when scoring all documents in large databases. Therefore, re-rankers are often used to refine the retrieval results after using DPR or other methods for initial filtering.

2.3 Retrieval-Augmented Generation

Retrieval-Augmented Generation (RAG) has emerged as a prominent approach to improving the capabilities of language models by introducing external information with text-generative modeling.

2.3.1 Embedding-based Retrieval

Various techniques have been introduced that integrate retrieved textual data and language models, some of which emerged prior to the introduction of the term Retrieval-Augmented Generation. For example, the Retrieval Augmented Language Model Pre-Training (REALM) paper introduced by Guu, Lee, Tung, et al. [24] marks one of the earliest techniques to integrate knowledge retrieval within the training of a language model. This approach divides the task into a neural knowledge retriever for document retrieval and a knowledge-augmented encoder for interpreting the retrieved documents. Based on the BERT architecture, the neural knowledge retriever embeds queries and documents to identify and fetch the most relevant document for each query. Upon retrieval, the knowledge-augmented encoder takes a concatenation of the original query and the retrieved document as input. This concatenated input is processed through a Transformer architecture, allowing for rich cross-attention between the query and the document. Unlike many generative models, REALM focuses on extractive tasks, training the system end-to-end to seamlessly leverage retrieved knowledge and language understanding. In contrast to REALM's extractive nature, our study explores the use of generative models in RAG systems, separating the retrieval process from the generative process to assess their efficacy in complex question-answering. We seek to explore the potential for generative models to utilize external knowledge effectively and generate more nuanced and detailed responses, thereby extending the capabilities of RAG systems beyond what is achievable through extractive methods alone.

The term Retrieval-Augmented Generation was first introduced by P. Lewis, Perez, Piktus, et al. [6]. RAG marked a significant advancement in NLP by integrating a neural retriever and a Seq2Seq generator to enhance knowledge-intensive tasks. Unlike REALM, which focuses on extractive QA using a single model for retrieval and encoding, RAG introduces a distinct architecture combining a DPR for document retrieval and BART for sequence generation. Additionally, the retriever and generator models are trained end-to-end while keeping the vector embeddings of each document fixed, as it is computationally expensive and shown to be an unnecessary step to update the embeddings at training time. Our study focuses on generative models within RAG systems, where we explicitly separate the retrieval process from the generative process. This approach investigates the generative model's capability to synthesize new information based on retrieved documents, potentially offering more nuanced responses to complex questions.

Various algorithms emerged afterwards and incorporated different retrieval methods or generative models to improve the RAG technique. For example, in a study conducted by

Izacard and Grave [25], the processing of the retrieved data differs. The answer generation model of the algorithm is designed using a Fusion-in-Decoder (FiD) model capable of scaling to a larger number of textual data. This model involves encoding a concatenation of the question and each document separately, then aggregating these vector encodings in the decoder. The system generates contextually rich answers through attention mechanisms applied across the combined encodings. The study uses a vector database to evaluate BM25 and DPR for document retrieval. Retrieval-Enhanced Transformer (RETRO) by Borgeaud, Mensch, Hoffmann, et al. [26] is another proposed language model that improves RAG and DPR techniques. RETRO allows for repeated retrieval throughout the generation process, not just based on the initial prompt like RAG or FiD but continuously as the retrieved data expands. This ongoing retrieval capability enables RETRO to dynamically adapt the gathered context of the text. RETRO is also trained from scratch, using a Transformer architecture and a chunked cross-attention mechanism; however, the embeddings are calculated using a frozen BERT model. Documents are chunked into multiple segments, encoded, and stored in a vector database for faster retrieval. RETRO performs similarly to GPT-3 and Jurassic-1 with 25 times fewer parameters.

Many papers have emerged that study the relevance of incorporating multiple data sources and types into RAG systems. "Unified representations of structured and unstructured knowledge for open-domain question answering" (UniK-QA) by Oguz, X. Chen, Karpukhin, et al. [27] analyses the relevance of including different data types in augmenting language models. The FiD model, introduced by Izacard and Grave [25], has been used for RAG with a database incorporating structured, unstructured, and semi-structured data, combining text and knowledge base data. All data types are heuristically transformed into text, and a retrieval process is performed using the DPR method. Y. Li, Peng, Shen, et al. [28] also include different datatypes and sources for retrieval by presenting the PLUG language model. This model is trained using different knowledge sources, from text to KG data. The retrieval process is simple, relying on search queries and keyword matching to find relevant data. Afterwards, data is transformed into text and filtered using first statistical and then semantic ranking. The statistical ranking uses TF-IDF to create an initial set of data. Then, this set is additionally filtered by semantic ranking using a sentence-BERT similarity score and a preset threshold. Unlike other studies that fine-tuned models for generating answers given retrieved documents and studies that use a cross-attention mechanism between documents and inputs, Re-plug by Weijia, Sewon, Michihiro, et al. [29] considers the generative Large Language Model as a black box. Each document retrieved from the RAG system is used individually to augment the LLM and generate probabilities for the output tokens, which are then aggregated across documents to compute the final token probabilities. This technique prevents the truncation of the appended documents if they exceed the LLM's context window size.

2.3.2 Graph-based Retrieval

Various studies have explored ways to extract information from Knowledge Graphs (KGs) and graph databases, leveraging their structured format. One of the earliest KG-based question-answering systems followed a multi-stage process to link user questions to entities in the graph and extract an answer. First, entities within the user question are extracted and linked to corresponding nodes in the KG. Afterwards, relation detection is employed to find the appropriate edge in the graph that leads to the correct answer. Berant, Chou, Frostig, and Liang [30], for instance, proposed a semantic parsing approach to convert textual questions into logical forms or executable queries that extract a response from the graph database. Other techniques, such as those introduced by Z. Wang, Ng, Nallapati, and Xiang [31], employ retrieval, re-ranking, and multi-task learning to improve entity detection, linking, and ranking accuracy.

Recent advancements aim to eliminate the multi-step processes of traditional graph-based retrieval to prevent error accumulation. One such approach is the Direct Fact Retrieval (DiFaR) method proposed by Baek, Aji, Lehmann, and Hwang [32]. It extracts all triplets from the database and encodes them into a dense vector embedding similar to the DPR technique. The method, therefore, stores logical relationships in a vector database instead of only textual data. The study also employs a re-ranker model that takes the question and document as input and outputs a rank score indicating the document's usefulness in answering the question. The top K triplets are retrieved from the vector database, and only these K triplets are then passed through the re-ranker to rearrange their relevance to the question.

Y. Wang, Lipka, Rossi, et al. [33] also leverage the structured nature of graph databases at retrieval in a RAG system. This paper constructs a KG using a set of documents and other data structures. Subsequently, an LLM-based graph traversal is designed to retrieve relevant information from the graph database. Initially, TF-IDF is used to compare the content of nodes with the given query. The node with the highest similarity score is extracted and considered a starting point for the graph traversal. Then, the function iteratively ranks and navigates through neighboring nodes to gather valuable information for prompting the LLM. A fine-tuned LLM is employed to rank neighboring nodes, extract the node with the most relevant data, and identify the next traversal steps. The LLM is trained to consider both the question and the previously traversed nodes to select the next most promising node, ensuring that an understanding of the cumulative retrieved data informs each traversal step. This technique enhances the likelihood of selecting nodes with fewer information repetitions and higher quality as an addition to the gathered information. Compared to vector embedding techniques, which typically rely on disconnected data with condensed vector representations, graph traversal captures the inherent logical connections in the Knowledge Graph, leveraging the structural properties often lost in vector spaces. This makes graph-based methods particularly powerful for question-answering that benefit from relational reasoning and contextual awareness.

3 Methodology

3.1 Embedding-based RAG Techniques

Vector-based RAG techniques are flexible and straightforward, capable of retrieving a diverse range of data types that can be transformed into vector representations. They aim to query items closest in vector space to a given query vector. The retrieval accuracy relies on the quality of the vector embeddings, which must effectively represent the underlying data. Moreover, embedding-based RAG supports dense vector embeddings produced using deep encoder models, as well as sparse vector formats, such as TF-IDF, commonly used in traditional information retrieval systems.

3.1.1 Dense Passage Retrieval (DPR)

One of the most common techniques in Retrieval-Augmented Generation (RAG) is using a Dense Passage Retrieval (DPR). This approach transforms documents into low-dimensional, dense vectors, allowing for a semantic search process [20]. This results in a retrieval process based on the contextual content of documents rather than syntactic matches, allowing the retrieval process to understand and capture the depth of semantic relationships for a more accurate retrieval. A vector database efficiently stores these embeddings for quick access, eliminating the need to compute the similarity scores between the user query and every stored document. At inference, user queries are similarly embedded into vectors, and a fast nearest-neighbor search is performed to fetch the documents whose vector similarity scores are the closest.

Chunking

DPR is capable of handling unstructured data, as it processes independent documents and only requires textual data for embedding. However, chunking algorithms are utilized in many applications, splitting the documents into smaller segments and avoiding the loss of context in large documents. Chunking can be implemented by splitting the text at token, word, or sentence level with a chunking size to determine the average length of each segment. Additionally, an amount of overlap is optionally introduced between chunks to ensure continuity and to preserve context. This ensures that critical information that might span the boundaries of two chunks is not cut off, preventing loss of context.

Re-ranking

Since the introduction of DPR for document retrieval, various enhancements have been proposed to refine the accuracy. Re-ranking, introduced by Nogueira and Cho [23], is one common approach to improve the results by re-ordering the relevance of documents after retrieval. These models surpass encoder-based similarity scoring by taking both the query and document as input and cross-encoding for a more precise relevance assessment. While resulting in improved accuracy, re-ranking is computationally expensive when used for the initial document retrieval, as it requires an iterative calculation on all data when given a new user query. Therefore, DPR is still used to first gather the top K^1 relevant documents, and re-ranking is then applied to the K^1 documents for a more refined arrangement of data. Following this, the top K^2 documents are selected post re-ranking, where K^2 is less than K^1, ensuring an accurate yet efficient document retrieval.

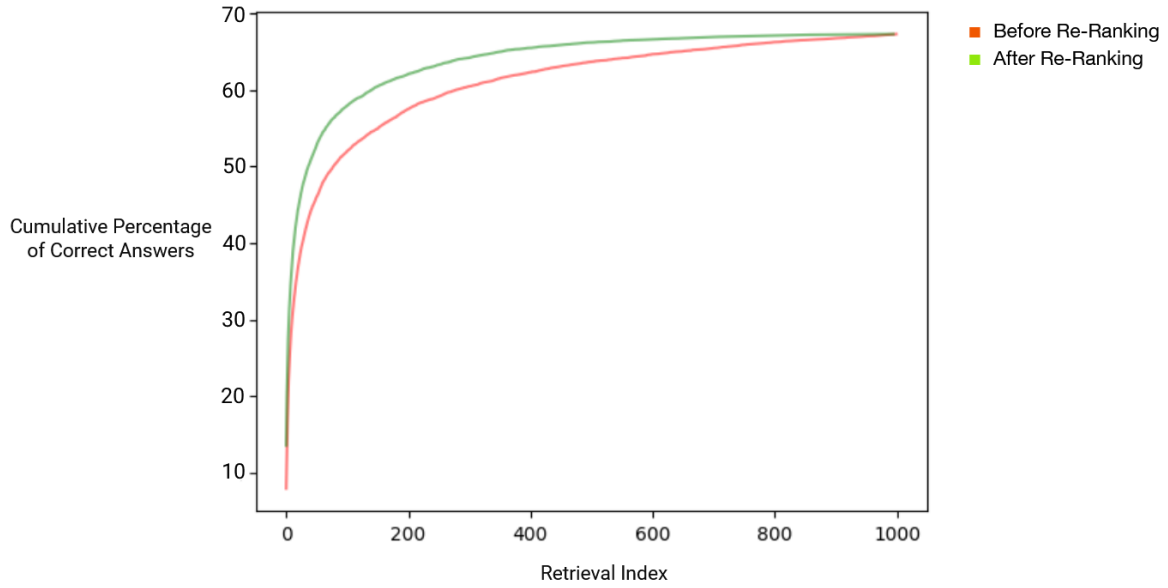


Figure 3.1: The cumulative success rates of data retrieval using DPR before and after re-ranking

Figure 3.1 shows the accuracy of retrieving the correct WikiData article from the vector database for the set of questions in our evaluation dataset. The accuracy increases with the number of retrieved nodes and plateaus at around 70%, given that 79% of the questions have corresponding entities in our vector database. The red line represents the accuracy without re-ranking the nodes and only relying on the vector embedding similarity. In contrast, the green line shows the accuracy when most similar 1000 nodes are re-ranked before selecting the set of top nodes. As we can see, the re-ranker improves the accuracy by approximately 5-10% on average, demonstrating its effectiveness.

3.2 Graph-based RAG Techniques

We present various graph-based RAG techniques that extract relevant data from Knowledge Graphs. In KGs, structured data enriches retrieval by enhancing accuracy through linking entities with relationships. However, adding such a structure introduces more retrieval complexity, mainly due to the various methods of constructing these graphs. One of the standard retrieval techniques that interact with graph databases follows a multi-step approach involving entity detection to identify relevant entities within a user query, entity matching to link these entities with nodes in the KG, and relation classification to select relevant relations associated with the identified entities [34]. These processes heavily depend on the database's structure and the defined relationship types, requiring specific labels and relying on query-triplet pairs for training. This results in a lack of generalization, as the traditional method struggles to accommodate newly emerging entities and relations. Moreover, this pipeline is susceptible to error propagation, where inaccurate results in the early stages of entity detection can compromise the performance of later steps. This study explores more advanced and flexible retrieval techniques that can perform efficiently across various RAGs without fine-tuning.

3.2.1 Cypher Query Generation

A promising technique that has gained attention through various sources involves leveraging powerful generative language models to produce Cypher queries for graph databases, facilitating the extraction of relevant information. Despite its simplicity, this technique has been mentioned and implemented in numerous platforms, including the LangChain framework [35], the Neo4j database's blogs [36], and the LlamaIndex framework's documentation [37].

Cypher, the query language designed explicitly for graph databases, is renowned for its expressive power. It facilitates powerful graph traversal and node matching, thereby accommodating complex queries. Thus, adapting LLMs to generate Cypher queries enables more flexibility and unlocks the potential for advanced data retrieval from graph databases.

In generating Cypher queries with LLMs, general information about the graph database is provided and added to the prompt. However, with standard Knowledge Graphs having a large number of entities in the database, it is not feasible to provide the model with all the details. Therefore, only the names of significant node properties are added to the LLM prompt, allowing for the possibility of filtering and string-matching operations. It is crucial to additionally include the labels of all relationship types to enable the LLM to select valid relationship types, ensuring the generation of effective queries that accurately navigate the graph structure.

Limitation

While Cypher queries offer considerable flexibility, they also introduce greater error possibilities. Given that we rely on generative language models to generate machine code, there is no guarantee of producing a valid and acceptable query. These models, often developed for question-answering, are trained on diverse datasets comprised of mostly human language text, which only sometimes equips them with the capability of accurate code generation. Despite these challenges, many models, even those with fewer parameters, demonstrate strong capabilities in code generation, making them more suitable for generating Cypher queries. However, the risk of generating invalid queries remains a concern. Implementing an error handling and validation mechanism is crucial to avoid these risks.

Subgraph Selection

When generating the Cypher query, we do not provide the LLM with a list of nodes; thus, it must infer the existence of relevant nodes in the graph. Additionally, although we include the complete list of distinct relationships, we do not specify their locations in the graph or the nodes they connect, requiring the LLM to also assume the existence of connections. With a sparse Knowledge Graph, the mentioned assumptions of nodes and edges often lead to queries that yield no results and poor retrieval.

To overcome this limitation, we extract a relevant subgraph from the graph database and instruct the LLM to generate the Cypher query based on this subgraph. Working with a subgraph with fewer nodes and edges enables us to provide more context to the LLM. The approach helps focus the model's attention on a more manageable portion of the KG. Instead of the extensive list of relationships, we can provide the LLM with the outgoing relationships for each node in the subgraph. Moreover, we include additional details about the nodes, such as their labels and names, allowing the model to perform more precise filtering and generate more effective Cypher queries.

3.2.2 Knowledge Graph Traversal

In this study, we additionally evaluate a technique involving traversing the graph database and collecting relevant information. As implemented in the paper by Y. Wang, Lipka, Rossi, et al. [33], the technique begins with one node in the graph, referred to as seed, and explores candidate nodes by considering nodes neighboring the selected seed. The algorithm iteratively gathers the textual data of selected nodes and traverses edges in the KG until a stop criterion is met. Unlike the DPR technique, which relies solely on text similarities between the question and articles, this technique leverages the graph's structure, allowing for the exploration of additional options. As the algorithm progresses through neighboring nodes, it explores candidate nodes that are logically related to the previously explored data, uncovering more relevant information that could assist in answering the user question. The neighboring nodes are ranked at each iteration based on their relevance to the user query. The node from this candidate list with the highest rank score is selected. After selection, the node's

content is appended to the prompt, and its neighboring nodes are subsequently considered as additional candidates for further ranking. The ranking is calculated by considering both the user question and the content of the previously collected nodes. This approach aims to select nodes that minimize latency and repetition of content by choosing nodes that contain the most valuable addition to the already gathered information.

3.2.3 Triplets Extraction

Unlike the embedding-based technique, which relies on textual data to identify relevant candidates, this technique, proposed by Baek, Aji, Lehmann, and Hwang [32], considers relationships within the graph database for information retrieval. Triplets, comprising a source node, a destination node, and a relationship between the two, are extracted from the KG and compared against the user question. Since a triplet includes relationship information, defining how nodes are interconnected, this technique targets logical connections within the graph database, connections that may be buried in the textual content derived from the vector embeddings. While the vector-based technique considers the complete textual data, the connections to other neighboring nodes are not immediately apparent due to the density of information. This method puts an emphasis on the relationships over the context of each separate node, highlighting structural rather than textual information. Similar to the vector-based approach, the triplets are embedded and stored in the database for retrieval at inference. Baek, Aji, Lehmann, and Hwang [32] also use a re-ranker model that performs cross-encoding between the most relevant triplets and the user query. This re-ranker model re-orders the relevance of the triplets based on the content of the user query. Consequently, the top triplets are extracted, highlighting the most relevant nodes and edges and indicating which data to include in the LLM prompt.

3.3 Evaluation Approaches

To accurately evaluate and compare various RAG techniques, it is essential to compare each of the final generated answers in response to user queries. Benchmarking the performance of the RAG systems is performed by calculating the similarity of the LLM-generated answers to those in the evaluation dataset. Finding a robust similarity scoring between LLM-generated answers and the ground truth is crucial. There are several evaluation metrics to score the similarity between LLM-generated answers and the ground truth. Techniques include exact matching, Bilingual Evaluation Understudy (BLEU) [38], Recall-Oriented Understudy for Gisting Evaluation (ROUGE) [39], and BERTscore [40]. Additionally, we propose an evaluation metric inspired by Zero-Shot classification [41], [42] that utilizes entailment prediction models.

Exact matching simply compares the sting equivalency of the generated answer against the ground truth. For exact matching to be effective, answers LLMs must be direct and concise. BLEU and ROUGE are robust evaluation techniques that utilize n-grams to calculate text similarity and can handle answers with unnecessary text generated by LLMs. BLEU

calculates the precision of n-grams found in both texts being compared; this value is then adjusted with a brevity penalty to account for size differences. ROUGE, on the other hand, calculates the recall of n-grams common in both texts. BLEU is commonly used to evaluate text translations, while ROUGE is often used to evaluate text summarization. BERTscore is another evaluation metric that measures textual similarity through vector embeddings. Using the BERT encoder, BERTscore computes contextual embeddings of both texts and calculates the similarity score by performing a cosine similarity between the two vectors.

We introduce the approach of employing an entailment prediction model to evaluate the generated responses. Entailment or verdict prediction is a sub-task of automated fact-checking, which involves the process of verifying the validity of a claim against provided evidence, typically split into four tasks: claim detection, evidence retrieval, verdict prediction, and justification production [43]. Given a document, claim detection segments text into claims considered worthy of fact-checking. Evidence retrieval gathers relevant information for each extracted claim from a reliable corpus to support or refute these claims. Verdict prediction uses a language models to classify the relationship between claim and evidence into *entailment*, *contradiction*, or *neutral*. A high entailment probability confirms that the evidence supports the claim, suggesting it is true; a high contradiction probability indicates a conflict between the evidence and the claim. Neutral applies when evidence does not directly relate to the claim. Finally, justification production, an optional step in fact-checking systems, generates explanations for verdicts, especially when claims are predicted as false.

In the process of evaluating the LLM-generated answers against the ground truth, we focus solely on the verdict prediction step since the claim and evidence are predefined. We consider the claim to be the answer from the LLM and the evidence, the correct answer from the evaluation dataset. We evaluate the entailment versus contradiction probabilities, determining the similarity or inference of the LLM answer given the ground truth. Like BERTscore, using LLMs mitigates formatting issues and extra text in generated answers. However, unlike BERTscore, which independently encodes the answers, entailment classification models consider both answers simultaneously, adding further context for more accurate scoring. This technique is similar to Zero-Shot classification, which utilizes pre-trained verdict prediction models for unseen classification tasks without the need for fine-tuning. In summary, for our experiment, we apply Zero-Shot as a binary classification to compare the similarity between generated answers and the ground truth.

4 Experimental Setup

4.1 Dataset

This study uses the T-REx dataset by Elsahar, Vougiouklis, Remaci, et al. [44] to populate the vector and graph databases and provide data for augmenting the LLMs. The dataset aligns textual data with knowledge base triplets extracted from Wikipedia abstracts and Wikidata. T-REx comprises 5.4 million triplets and 4.6 million documents, specifically Wikipedia abstracts, linked with 642 unique relationship types. The alignment of the Wikipedia abstracts with the Wikidata KB triplets ensures that the triplets are extracted from information found within the textual data. This consistent information basis enables a balanced comparison of vector and graph-based RAG techniques, facilitating an in-depth assessment of their performance in information retrieval.

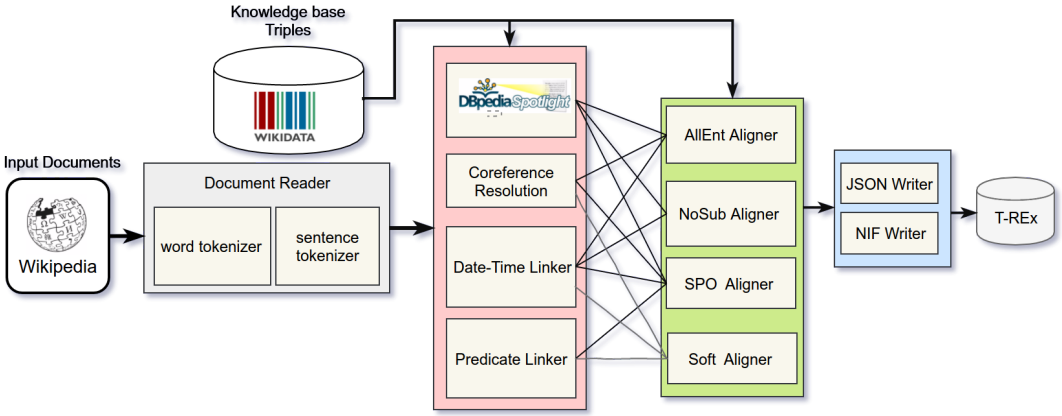


Figure 4.1: Overview of the pipeline and components used to produce the T-REx dataset, produced by Elsahar, Vougiouklis, Remaci, et al. [44]

4.2 Evaluation

4.2.1 Evaluation Dataset

For evaluation, we use the Mintaka dataset by Sen, Aji, and Saffari [45], a diverse set of 20,000 question-answer pairs, each with answers of various types, including Wikidata entities, numerical values, booleans, dates, and strings. Constructed using the Wikidata Knowledge

Graph, Mintaka also links all questions and 61% of answers to Wikidata, allowing the evaluation of not only the LLM-generated answers after augmentation but also the RAG systems' retrieval capabilities. The data collection and linking between the question-answer pairs and Wikidata entities were performed manually. Moreover, each question is classified by nine complexity types, ranging from generic and yes/no questions to multi-hop and intersection questions that require information from various Wikidata articles to answer. Table 4.1 shows all the complexity types of the questions, including a description of each type and an example. This classification enables a more detailed analysis of the performance of the RAG methods across various scenarios. Additionally, upon integrating the population and evaluation datasets, we found that 84% of entity type answers correspond to entries in the T-REx dataset, and 86% of questions are linked to articles in T-REx. This alignment results in coverage of 79%

Complexity Type	**Description**	**Example**
Generic	Questions that require simple fact lookups	Where was Michael Phelps born?
Yes/No	Questions where the answer is a Yes or No	Has Lady Gaga ever made a song with Ariana Grande?
Count	Questions where the answer requires counting	How many astronauts have been elected to Congress?
Superlative	Questions about the max or min of given attribute	Who was the youngest tribute in the Hunger Games?
Comparative	Questions that compare 2 items on a given attribute	Is Mont Blanc taller than Mount Rainier?
Ordinal	Questions based on an entity's position in an ordered list	Who was the last Ptolemaic ruler of Egypt?
Difference	Questions containing negations	Which Mario Kart game did Yoshi not appear in?
Intersection	Questions containing multiple conditions	Which actress played an Avenger and was in Lost In Translation?
Multi-hop	Questions that require multiple steps and entities to answer	Who was the quarterback of the team that won Super Bowl 50?

Table 4.1: The nine different complexity types of the Mintaka dataset questions

To associate each question type from Mintaka to a difficulty level, we display the accuracy of generated answers of each type across various LLMs in figure 4.2. This accuracy is based on the correctness of answers generated from the questions without additional context from a RAG system. Based on these results and the definitions of the question types, we categorize "Yes/No" and "Generic" as more straightforward types. LLMs generally perform well on "Comparative" and "Intersection" questions, though these typically require two or more Wikidata abstracts for proper evaluation. Conversely, LLMs struggle with "Multi-hop",

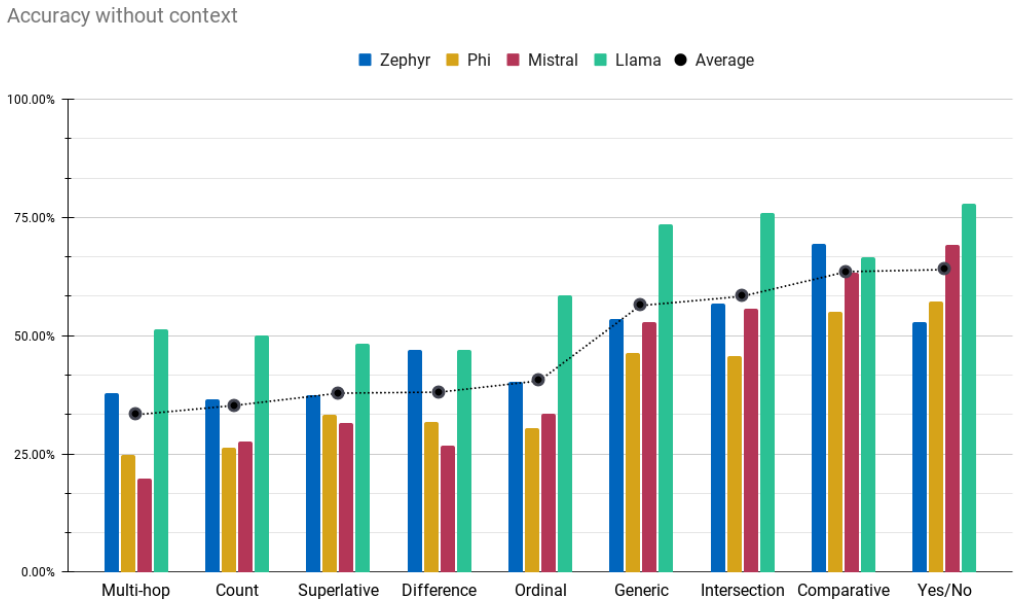


Figure 4.2: Response accuracy of different LLMs to the Mintaka question without data augmentation, categorized by question complexity

"Superlative", "Difference", and "Ordinal" questions, which require logical reasoning and information from multiple abstracts. LLMs also face challenges with "Count" questions; however, the information necessary to answer these questions could be derived from a single abstract.

Given that Wikipedia and Wikidata are open source, the LLMs used in our study may have previously been exposed to this data in the training process, potentially influencing their performance with augmented data. The training data of all LLMs is not disclosed, and the exact version and date of the Wikipedia or Wikidata dumps used for training are unspecified. Additionally, there is a discrepancy in the Wikidata versions between the evaluation and population datasets, with the question-answer pairs produced in 2022 and T-REx using a 2017 Wikidata dump. This temporal gap may affect the alignment and relevance of information, resulting in an uncertainty of improvement when augmenting the LLMs.

Figure 4.3 shows the accuracy of answers from Llama and Phi when augmented with the correct Wikidata abstract from the T-REx dataset, aligned with the entities from Mintaka. The results increase when providing the appropriate Wikidata text, indicating that the chosen population dataset, despite potential overlaps in the training data of LLMs with our source materials and discrepancies in dataset versions, provides valuable information for accurately answering evaluation questions. Thus, the T-REx and Mintaka combination effectively supports our study assessing RAG systems.

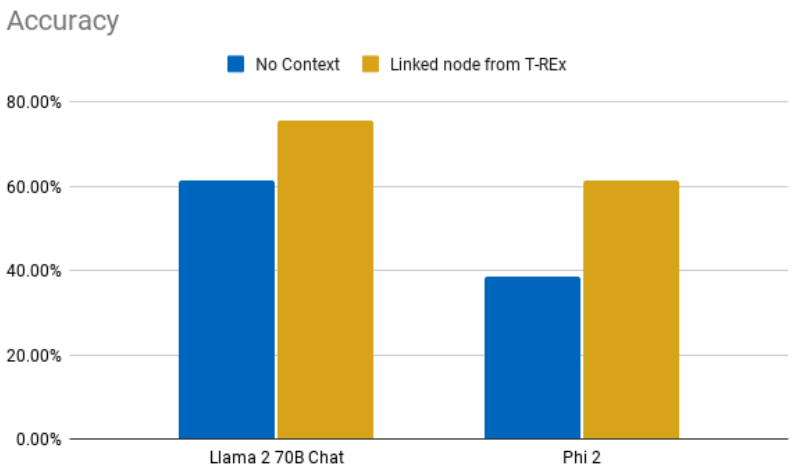


Figure 4.3: Accuracy of LLMs answers on the Mintaka questions when augmented with the correct Wikidata article from T-REx

4.2.2 Evaluation Approaches

Various evaluation techniques are tested to compare the LLMs' generated answers to the ground truth from Mintaka. The comparison assesses the performance of each technique across different types or complexities of questions, as previously discussed. Evaluating the retrieval process regarding document fetching accuracy is insufficient due to the varied nature of the many RAG techniques and the diverse data they provide for augmentation. Additionally, finding a robust similarity scoring between LLM-generated answers and the ground truth is crucial because the direct linkage of answers to Wikidata entities does not accommodate the variety of answer types in Mintaka, such as entities, numerical values, dates, booleans, and strings. We manually tested exact matching, BLEU, ROUGE, and BERTscore and used the entailment prediction technique for a more comprehensive assessment.

Exact Match, BLEU, and ROUGE

Exact matching relies on the ability of Large Language Models to generate concise answers similar to the ground truth. Despite enforcing LLMs to generate brief, straightforward responses with clear instructions like "provide only the answer to the question" and "do not include explanations, apologies, or text other than the answer", including additional text is often unavoidable. For example, instead of simply providing the expected answer of "32" in response to the following question, "How many Grammys does Beyoncé have?", LLMs might output "Beyoncé has 32 Grammys". Therefore, exact matching fails to accurately assess the performance of the RAG systems in our study.

BLEU, ROUGE, and other n-gram-based similarity scoring techniques can handle answers with unnecessary text from LLMs. However, their robustness fails to properly evaluate certain

LLM answer formats. For instance, numerical answers presented alphabetically ("five" instead of "5") lead to mismatches. Moreover, their reliance on n-grams does not distinguish correct answers in comparative questions. For example, when given the following question, "Who is older, Beyoncé or Rihanna?", both generated answers, "Beyoncé is older than Rihanna" and "Rihanna is older than Beyoncé", yield equal results. Given these limitations and the challenge of accommodating all unique answer formats, we decided against using n-gram-based evaluation techniques.

We additionally tested BERTscore for our study, which measures textual similarity through vector embeddings. Designed to evaluate applications like summaries and machine translation, BERTscore proved inadequate for our case. Manual testing revealed imprecise results when evaluating short answers from our dataset. Consequently, BERTscore, alongside BLEU and ROUGE, was considered unsuitable for accurately evaluating the performance of LLMs in our specific scenario.

Entailment Classification

We finally chose to employ entailment classification models to evaluate our generated responses. The score is calculated by setting the answer from the LLM as the claim and the correct answer from the evaluation dataset as the evidence for the verdict prediction. We further enhance the accuracy by incorporating the question within both the claim and evidence for additional context. For example, for the question "How many rings are there in the Olympic Games?" with the correct answer being "5" and the LLM's answer "There are five rings," we frame the claim as: "How many rings are there in the Olympic Games? There are five rings". The evidence is phrased as: "How many rings are there in the Olympic Games? 5". This structuring facilitates the fact-checking model's evaluation of the accuracy of the LLM's response against the factual answer. Finally, we compare the probability scores of the entailment and contradiction classes to determine the accuracy of the LLM's response.

However, the mentioned evaluation technique might inaccurately classify broad LLM answers as correct if they are positioned higher in the taxonomy hierarchy than the correct answer. For instance, if "What is the capital city of France?" receives the answer "A major city in Europe", this incorrectly scores high in entailment with a value of 97.5% because it is factually correct but overly broad, failing to directly answer the question. This scenario results in many false positive cases, highlighting the need to refine the evaluation method to accurately match the specificity and directness of LLM-generated responses. To address this issue, we adjusted our approach by additionally calculating the verdict probabilities with reversed roles of claim and evidence. We set the generated response as the evidence and the ground truth as the claim, resulting in two probabilities: one with the original claim-evidence setup and another with their roles swapped. In our previous example, flipping the claim and evidence significantly decreases the entailment probability from 97.5% to 41%. As "A major city in Europe" is set as the evidence, "Paris" is considered as an incorrect claim. Therefore, we consider a generated answer to align with the ground truth only if both probabilities exceed 50%, ensuring that

LLM responses closely match the specific information in the evaluation dataset.

Evaluating the Evaluation Technique

To validate our evaluation technique further, we tested it on 1800 GPT-4 generated answers, half positive and half negative samples, based on Mintaka's answers. The generated samples are equally divided across different question complexities (200 samples for each complexity). For positive samples, we paraphrased correct answers to maintain accuracy, and for negative samples, we instructed GPT-4 to generate incorrect answers, ensuring they differed from the ground truth.

Base Model	Training Datasets	Accuracy	Recall	Specificity	Runtime /s
Bart Large	MultiNLI	94.67%	92.2%	98.78%	0.1575s
DeBERTa Base	MultiNLI, Fever-NLI, ANLI	94.76%	94.6%	96.66%	0.059s
DeBERTa Large	MultiNLI, Fever-NLI, ANLI, LingNLI, WANLI	97.44%	97.6%	98.2%	0.18s

Table 4.2: Evaluation of various verdict prediction models on the generated samples

In table 4.2, we display the results of various verdict prediction models. We tested three open-source pre-trained models based on BART provided by Shu, Xu, B. Liu, and J. Chen [46] and DeBERTa provided by Laurer, Van Atteveldt, A. Casas, and Welbers [47]. These models are fine-tuned on various natural language inference (NLI) datasets that contain a pair of sentences, a premise and a hypothesis, and labels of "entailement", "contradiction", or "neutral". Datasets include MultiNLI [48], Fever-NLI [49], Adversarial-NLI (ANLI) [50], LingNLI [51], and Worker-AI Collaboration for NLI (WANLI) [52]. Results show that the large DeBERTa model, fine-tuned on all mentioned datasets, is the most effective, achieving 97.44% accuracy. This model is therefore chosen to assess the accuracy of the final LLM-generated answers.

4.3 Large Language Models (LLMs)

To properly evaluate the various RAG techniques on the Mintaka dataset, we employ different Large Language Models to better generalize the results. Table 4.3 lists the LLMs used in our experiments. Mistral [4], Phi [53], and Zephyr [54] are run locally on an Nvidia Ampere A100 GPU. Llama [3] is operated through an API provided by Deep Infra [55], which enables efficient use of an LLM with 70 billion parameters.

LLM Name	# Parameters	Inference Time /s
Mistral 7B Instruct [4]	7 billion	2.15s
Zephyr 3B Beta [54]	3 billion	1.16s
Phi 2 [53]	2.7 billion	2.12s
Llama 2 70B Chat [3]	70 billion	1.13s

Table 4.3: List of LLMs used to generate responses for our experiments

4.4 Embedding-based RAG Techniques

In this study, we use the Weaviate framework [56] for our vector database to store the document embeddings, with the Hierarchical Navigable Small World (HNSW) algorithm [57].

4.4.1 Dense Passage Retrieval (DPR)

To set up the DPR algorithm, we evaluated various fine-tuned embedding models, particularly for QA systems. This fine-tuning enhances their capability to comprehend the specific context of user queries and enables them to connect short questions to extensive, relevant documents. One model is based on DistilBERT [58], which was fine-tuned using the MS MARCO passage ranking dataset [59]. This dataset contains pairs of search queries along with relevant documents from different web sources of diverse domains. We also evaluate an MPNET [60] model, fine-tuned on the Multi-QA dataset [61], which comprises over 215 million question-answer pairs from various sources including search engines, blogs, and many more. All mentioned models are open-source, provided by the Sentence Transformers framework [62]. For the re-ranking process, we use a cross-encoder model also provided by Sentence Transformers, based on MiniLM [63], [64], and fine-tuned on the MS MARCO dataset.

Chunking

In this study, we evaluate multiple chunking strategies to identify the optimal one for our dataset. As mentioned, chunking can be implemented by dividing the text at the token, word, or sentence level. We chose a chunking size of 1024 characters, which aligns with the average text size used in the training data of our encoder model. This size ensures that each chunk contains sufficient textual data without losing contextual information post-encoding. Additionally, we introduce a 20% overlap between chunks to maintain continuity and coherence in the data.

Table 4.4 presents the Mean Reciprocal Rank (MRR) metric [65] of each model and chunking technique. Our final setup, which yielded the best performance, is the MPNET model fine-tuned on the Multi-QA dataset, with chunking performed at the sentence level using the spaCy framework [66].

Chunking	Model	MRR@1000	MRR@1000 + re-ranking
Split by words	Multi-QA MPNET	0.09334	0.11414
Split by tokens	MSMARCO DistilBERT	0.12399	0.20302
Split by tokens	Multi-QA MPNET	0.14152	0.20583
Split by sentence (NLTK)	Multi-QA MPNET	0.13746	0.21251
Split by sentence (spaCy)	MSMARCO DistilBERT	0.12853	0.20990
Split by sentence (spaCy)	Multi-QA MPNET	0.14817	0.21310

Table 4.4: Retrieval score of different embedding models and chunking techniques

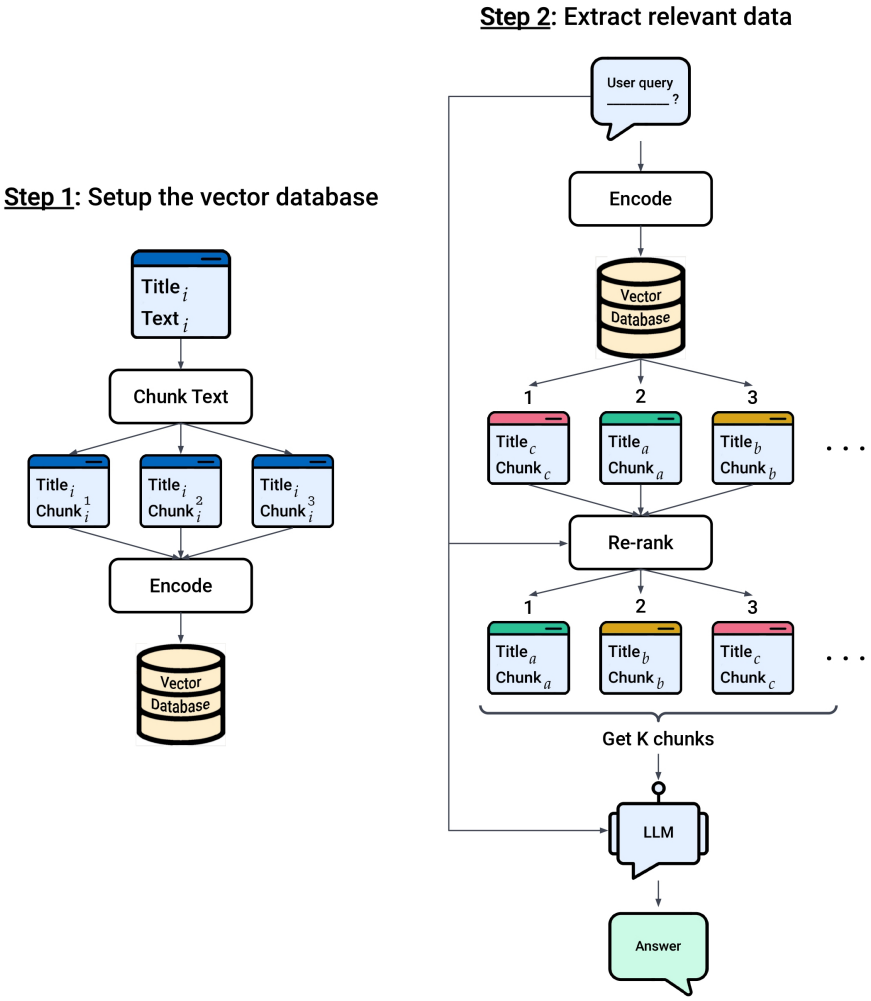


Figure 4.4: The setup and inference processes of the DPR technique

Figure 4.4 illustrates the steps to store the data in the vector database as well as the steps at inference to extract relevant data and prompt the LLM. For storing, the Wikidata articles, which include $Title_i$ and $Text_i$, are first chunked into multiple sections. Each $Chunk_i^j$ is embedded using the selected embedding model, along with the $Title_i$, and stored in the vector database. At inference, when given a user query, the query first gets embedded using the same embedding model used to store the data. We then identify the chunks with the closest embedding in vector distance compared to the query embedding. In our case, we use the dot product as the distance metric, which is used to train the embedding model. The most relevant chunks are then re-ordered by the re-ranker model, which also takes into consideration the user query. The top K chunks, based on the ranking scores of the re-ranker, are concatenated and added to the LLM prompt, along with the user query, to produce an answer.

4.5 Graph-based RAG Techniques

The Knowledge Graph structure is already provided by the T-REx dataset, which uses Wikidata entities as nodes and includes corresponding edges that connect these nodes. To store and query the data, we utilize the Neo4j framework [67]. We additionally incorporate the Awesome Procedures On Cypher (APOC) [68] add-on, which contains a vast collection of procedures and functions that significantly extend the capabilities of Cypher queries within Neo4j.

4.5.1 Cypher Query Generation

Cypher Generation On The Full Graph

With around 5 million entities in the database, providing the LLM with all node information exceeds the capacity of the context window. Therefore, in our experiment, we inform the model about the graph structure by specifying that nodes contain *Title* and *Text* properties. When providing unique relationship types, we opt to narrow down the list to 118 names that can be found more than once in the graph, resulting in approximately 775 tokens in the LLM prompt dedicated to the relationship names.

We manually tested multiple instructions to help the LLM generate effective Cypher queries. To reduce the likelihood of producing invalid Cypher queries, we carefully crafted our prompt with the following set of rules to mitigate errors:

1- *Encapsulate the Cypher query within <cypher></cypher> tags for clarity*
Despite clear instructions to generate only the Cypher query, LLMs tend to include extra text in the generated answer. Instructing the model to enclose the Cypher query in XML tags facilitates the extraction of the query from the generated answer. Although this technique does not guarantee a correct query, especially with incorrect variable names, it ensures the extraction of a syntactically valid Cypher query.

2- *Employ "toLower" for case-insensitive matching of the node's "text" property and the search keyword, using a "CONTAINS" clause. Avoid using the equals sign.*
We enforce the use of functions in Cypher such as *toLower* and *CONTAINS* for case-insensitive and accurate matching with the *Text* property of the nodes. Without this rule, the LLM tends to use the default equality sign, hindering effective node filtering and matching.

3- *Filter nodes based exclusively on their "text" property.*
While providing the LLM with the properties of the nodes, which include *Text* and *Title*, we enforce the model to filter the nodes based on the *Text*. This gives more opportunity for the query to capture appropriate nodes since the *Text* property has the most information in our database.

4- *Derive the query directly from the user's provided question without assuming or inferring potential answers.*
We observed that many generated questions resulted in the model assuming the answer before retrieving information from the database. These assumptions manifest in the generated Cypher query, leading the model to filter nodes based on the assumed answer. Therefore, we aim to prevent that by prompting the LLM to generate the query exclusively based on the provided question.

In addition to all the previously mentioned rules, we provide the model with an example of the desired output format, enabling the model to mimic this example and tailor it according to the provided question:

```
<cypher>MATCH (n0)-[:relationship]->(n1) WHERE toLower(n0.text) CONTAINS
toLower('Keyword') RETURN n1</cypher>
```

Figure 4.5 shows the final design of the prompt used to generate Cypher queries using the entire graph database. *relationship_list* is an array of all the names of relationships found in the database, and *user_query* is the question from the user that the model aims to answer.

```
prompt = f""" As a specialized agent, your task is to generate Cypher queries that
    accurately fetch information from a graph database, directly answering the
    user's query. Please follow these guidelines to construct the Cypher query:

1. Encapsulate the Cypher query within <cypher></cypher> tags for clarity.
2. Filter nodes based exclusively on their "title" and "text" properties, nodes only
    have "title" and "text" and no other properties.
3. Derive the query directly from the user's provided question without assuming or
    inferring potential answers.
4. Employ "toLower" for case-insensitive matching of the node's "text" property and
    the search keyword, using a "CONTAINS" clause. Avoid using the equals sign.
5. Only use the provided relationship types and properties from the database schema,
    which are:
- Relationship Types: "{relationships_list}"

Given the user's question:
"{user_query}"

Here is an example:
<cypher>MATCH (n0)-[:relationship]->(n1) WHERE toLower(n0.text) CONTAINS
    toLower('Keyword') RETURN n1</cypher>

Now, based on the actual user question provided, generate a Cypher query following
    the above format and rules."""
```

Figure 4.5: the structure of the prompt used to generate Cypher queries from the complete graph database

Cypher Generation on a Subgraph

To define a relevant subgraph within the Knowledge Graph, it is sufficient to identify a set of relevant nodes in the KG. We use the already existing vector database, populated for the DPR method, to extract a set of nodes. We then link text chunks closest in vector similarity back to nodes in the graph database. Additionally, we order the relevance of the set of nodes with the re-ranker model before filtering down to the top-ranked nodes, similar to the DPR technique. Figure 4.6 illustrates the steps for extracting a subgraph from the KG. Given the current graph database structure, with a maximum of 8 unique outgoing relationships per node and an average of 2-3 words per node title and relationship name, we define a subgraph consisting of 20 nodes to avoid exceeding the context window of our LLMs.

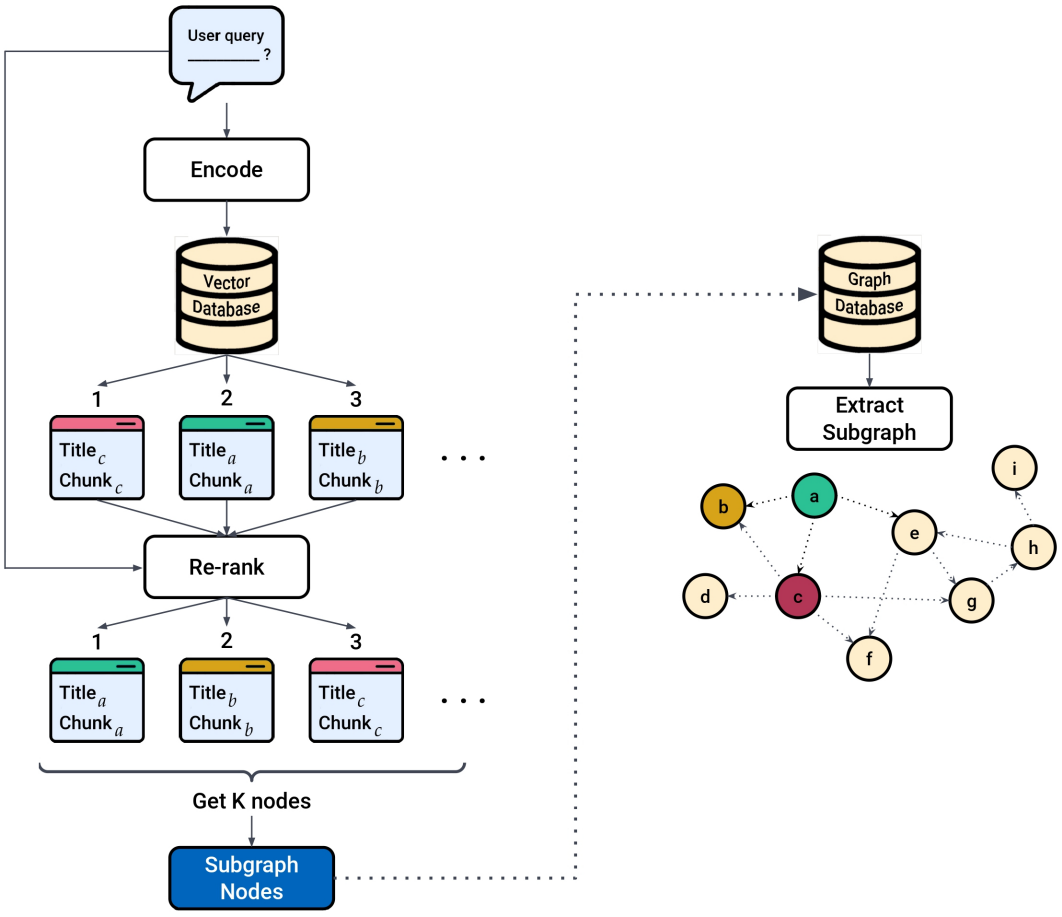


Figure 4.6: The process of extracting a subgraph from the graph database

The defined subgraph is not necessarily connected as we extract the top-ranked nodes from the vector database without considering their interrelationships. We chose not to impose an additional condition for picking nodes that form a connected subgraph to keep the focus on the most relevant nodes based on the vector embeddings and ranking scores. Since all the node titles and outgoing relationships are included in the prompt, the LLM can still opt to query disconnected nodes and extract valuable information they provide.

Therefore, the prompt for generating a Cypher query on a subgraph includes the list of node titles and the list of relationships, eliminating the complete list of relationships from the previous prompt. The relationships present in the subgraph are presented as JSON format where each key is a title of a node in the subgraph, with a value as an array of the names of outgoing relationships of that node. We assume that language models that have the coding capabilities to produce valid Cypher queries, are also able to comprehend JSON objects.

Language Model for Cypher Query Generation

To properly evaluate the Cypher generation RAG technique across various LLMs, the initial goal was to have each model independently produce a Cypher code and query the database before responding to the questions from the evaluation dataset. However, after further investigations, we found that only a few models are capable of generating valid Cypher queries based on the prompted instructions. Among the models we tested in this paper, Phi 2 emerged as the most proficient in coding despite having only 2.7 billion parameters [69]. Therefore, we opted to generate the Cypher queries only using Phi 2 and then produce the answers using the other models.

4.5.2 Knowledge Graph Traversal

Seed Extraction

In the paper by Y. Wang, Lipka, Rossi, et al. [33], seed nodes are selected using TF-IDF by finding common words between the question and text of the nodes. For our research, given the setup of the vector database from the DPR technique, we opted to use vector embeddings along with re-ranking to choose our starting node. Our experiments demonstrated that the combination of vector similarity and a pre-trained re-ranker model is an effective approach to identifying relevant articles given the user query. Moreover, utilizing a different seed extraction technique introduces an extra layer of evaluation, leading to variation in results due to the initial seed choice. With the Knowledge Graph Traversal and the DPR techniques relying on the same initial retrieval process, our focus shifts to solely evaluating the importance of traversing the graph in RAG systems.

Candidate Selection

Y. Wang, Lipka, Rossi, et al. [33] scored the ranks of edges using an LLM-based graph traversal agent. The LLM is fine-tuned to rank candidates given the user question and the content of the previously collected nodes. For our experiments, however, we ranked the candidates by calculating scores using the re-ranker model previously employed in the DPR technique. The re-ranker compares the text of the candidate nodes, making it suitable for assessing the relevance of a text in response to a user question. Although the re-ranker's context window is smaller than the LLM used in the original paper, we have adapted our approach by including only the titles and relationship information of previously selected nodes on top of the text of the candidate node. Excluding the article text of collected nodes avoids exceeding the capacity of the re-ranker model while still including information from the previous steps, aiming to give higher priority to more diverse information. The advantage of using the re-ranker is its computational efficiency at inference time and its lack of fine-tuning, unlike the LLM graph traversal agent. Since the re-ranker is also used in the DPR technique, using the same re-ranker model ensures consistency in evaluating the RAG techniques and fair comparison of the benefits derived from traversing a graph instead of ranking or similarity scoring.

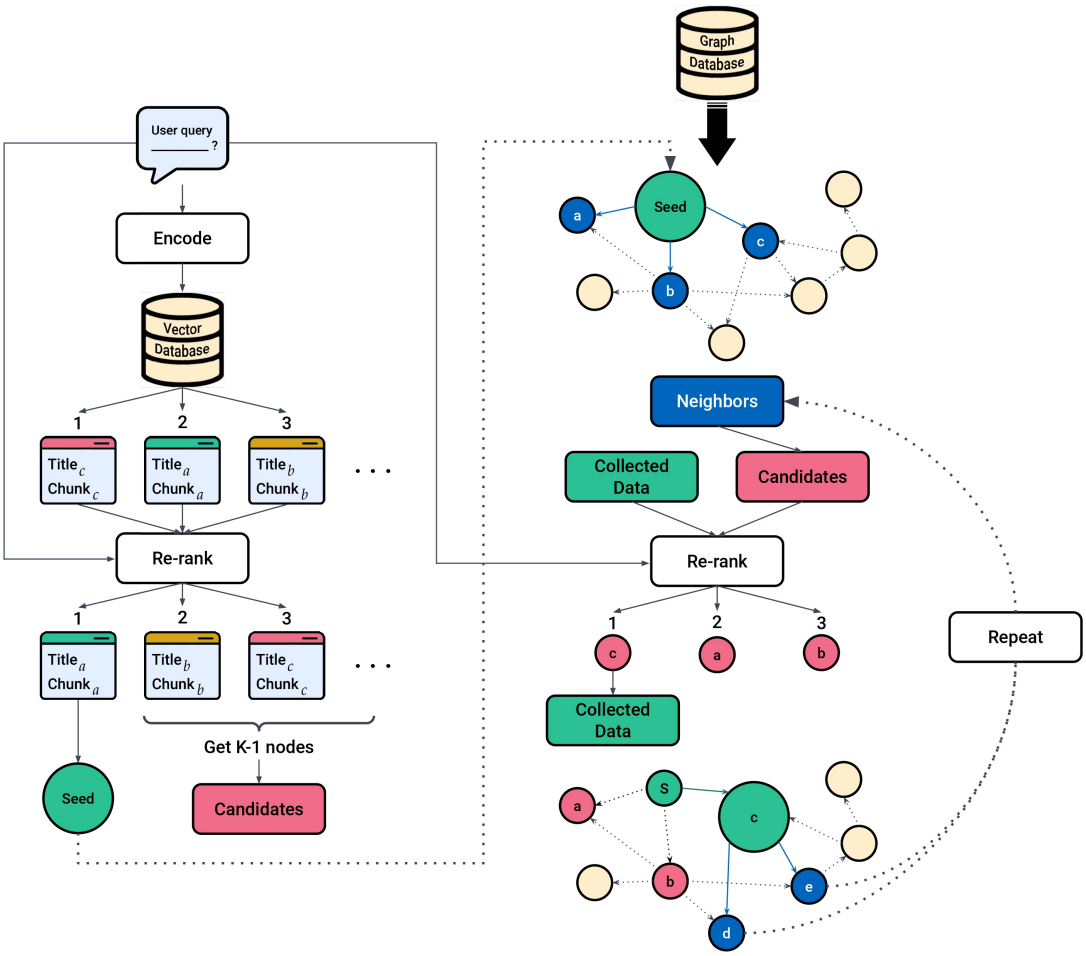


Figure 4.7: The steps for retrieving data using the Graph Traversal RAG method

Figure 4.7 illustrates the steps for performing the Graph Traversal technique, and figure 4.8 showcases the corresponding Python code for clarification. After extracting the top K seeds from the vector database and ranking them, the top-ranked seed is included in the collected data, while all other potential seeds are added to the list of candidates. We then iteratively extract the neighboring nodes of the highest-ranked node and include them in the candidates list. The candidates are always sorted by rank score, and at each iteration, only the top-ranked node, at the last index, is added to the collected data. Additionally, the list of candidates is trimmed by the *max_depth* to manage its size, as candidates with the lowest scores will never be included in the collected data if the list is larger than the *max_depth*. The stopping criterion is met when collected nodes reach the pre-defined maximum depth. Alternatively, the data collection process could stop if, at some point, the list of candidates is empty, mainly if the subgraph is sparse and K is less than *max_depth*.

```python
def graph_traversal_retriever(query, max_depth, K)
    '''
    query: The user question.
    max_depth: The number of nodes to collect from the Knowledge Graph.
    K: The number of seeds the algorithm should start with
    '''

    collected_data = []
    candidates = get_seeds(query, K=K) # Fetch initial seed nodes based on the query
    n = 0
    while n < max_depth:
        # Assuming candidates are already ranked, the last one is considered the
            top-ranked
        top_node = candidates.pop() # Pop the top-ranked candidate node from the
            candidates list
        collected_data.append(top_node) # Add the top node to the collected data
        next_candidates = get_candidates(top_node.destination, query) # Fetch
            neighbors of the top node
        candidates = [*candidates. *next_candidates] # Add the new candidates to the
            list of candidates
        candidates = rank_candidates(candidates, collected_data, query) # Re-rank
            candidates including the new ones
        if len(candidates) > max_depth:
            # Trim the candidate list
            candidates = candidates[:max_depth]
        else if len(candidates) == 0:
            # Stop the loop if the list of candidates is empty
            break
        n = n+1
    return collected_data
```

Figure 4.8: Python code implementing the Graph Traversal RAG technique

4.5.3 Triplets Extraction

Triplets Ranking

In this technique, we highlight the content of edges rather than the nodes and their descriptions from Wikidata. When comparing triplets, we only consider the nodes' titles and the name of the edge connecting them. Each triplet in the database is transformed into the following string: `"{source.title} is {edge.name} of {destination.title}."`

We use the same re-ranker model from the DPR RAG system to score each triplet based on the user query and identify the best triplet with the most relevant content. With around 5.4 million triplets in the database, it is computationally expensive to calculate the score of each triplet against the user query at inference time. Additionally, given that the graph database contains only 642 distinct relationship types, the string transformation of the triplets often leads to a narrow search space with many similarities within triplets. To mitigate the limitation of correctly identifying appropriate triplets for extraction, we took advantage of the full text of Wikidata abstracts to perform a pre-filter before ranking and selecting the triplet representations.

Pre-filtering

The initial pre-filtering process is performed by extracting a subgraph and only considering the triplets that are contained within that subgraph. Similar to the subgraph extraction method of the Cypher query generation technique, we compare embeddings from the DPR technique's vector database and employ the re-ranker model to extract a set of relevant nodes, defining the target subgraph. Triplets are then extracted by considering all edges that connect source and destination nodes within this defined subgraph. As demonstrated in our prior experiments, using vector representations and the re-ranker effectively identifies relevant entities. By aligning the same setup across all previous methods, including DPR and the Graph Traversal techniques, we can maintain our focus on evaluating the relevance of relationship selection.

However, finding a set of triplets within a subgraph is not guaranteed, as the KG is sparse, potentially resulting in an extracted subgraph that could be disconnected. Imposing additional constraints and limiting our pre-filtering to only considering connected subgraphs could lead to excluding valuable nodes. Additionally, restricting the search to only triplets might cause the same problem and potentially discard disconnected nodes with valuable information. Therefore, in addition to triplets, we also include each node separately in our list of candidates, ranking only the titles of each node. This approach allows the potential of appending singular nodes to the collected information, adding more flexibility to the method. Additionally, considering singular nodes enables the ranking method to extract a triplet or a relationship only when it has added value in answering the user question.

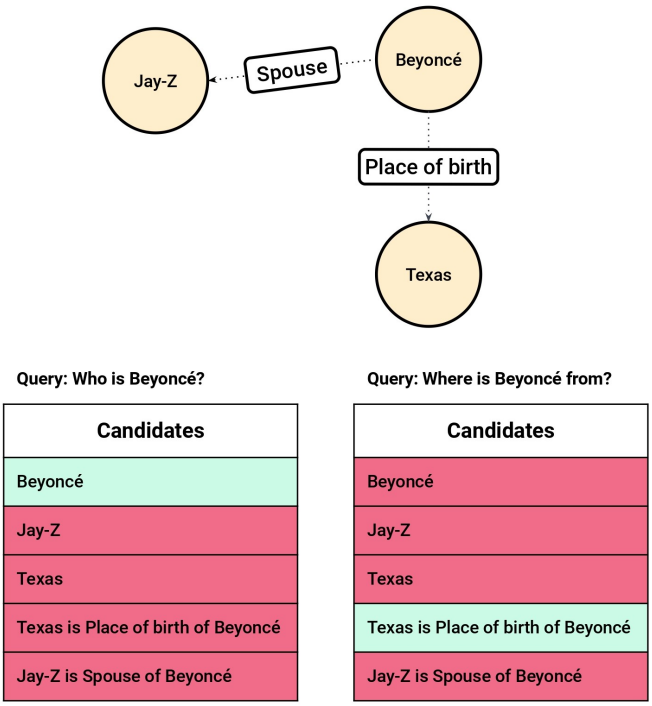


Figure 4.9: Example of selecting and ranking candidates in the Triplets Extraction technique

The figure 4.10 above demonstrates the benefit of including singular nodes as candidates. The individual titles of each node ("Beyoncé", "Jay-Z", and "Texas") are added to the list of candidates and ranked alongside the triplets featuring relationships like "Place of birth" and "Spouse". Given a generic question such as "Who is Beyoncé?", the model assigns a higher rank score to the singular node "Beyoncé", thereby preventing the addition of irrelevant information. However, in response to a more specific question such as "Where is Beyoncé from?", the re-ranker acknowledges the relevance of the relationship "Place of birth", resulting in the inclusion of both relevant nodes and the edge into the retrieved data.

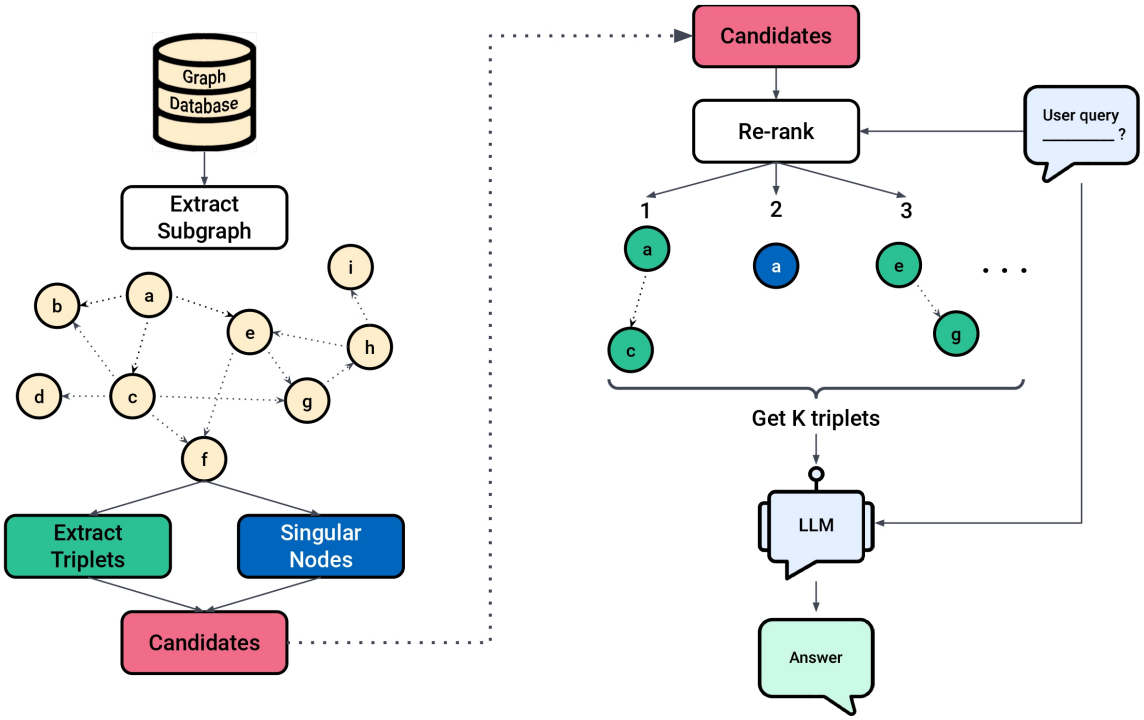


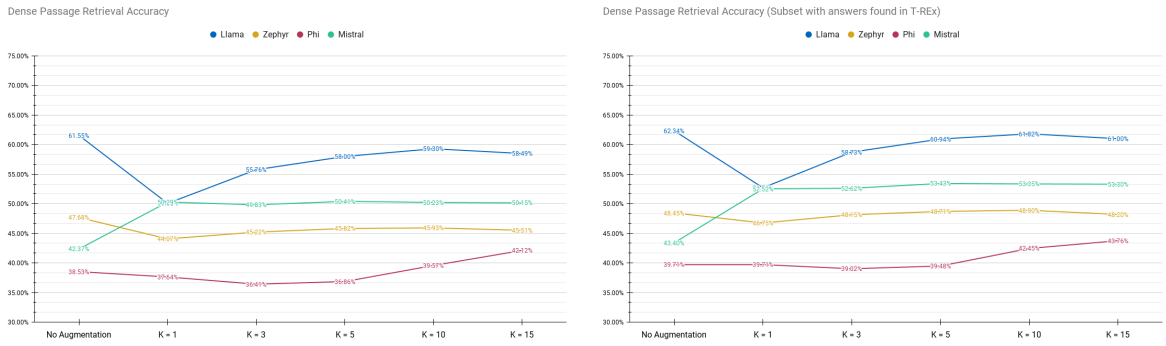
Figure 4.10: The steps for the Triplets Extraction technique

Figure 4.10 illustrates the procedures for performing the Triplets Extraction technique. The first step is to extract a subgraph from the graph database; the steps for this process are shown in figure 4.6. Triplets and singular nodes are extracted from the subgraph and considered as candidates for augmenting the LLM. The re-ranker model ranks all the candidates given the user query, and then the top-ranked candidates are given to the language model to answer the question.

5 Results

In this section, we present and interpret the results of the RAG techniques outlined in the experimental setup across different generative language models. Results are segmented in various ways to highlight the differences between embedding-based and graph-based RAG systems, showcasing accuracy across different question complexities and data sizes. Additionally, we benchmark the runtime of each retrieval technique to provide a comprehensive comparison.

5.1 Results of the Embedding-based RAG Techniques



(a) Accuracy on all the Mintaka questions

(b) Accuracy on a subset of Mintaka questions where T-REx contains the linked Wikidata article

Figure 5.1: Accuracy of the LLMs responses using the DPR method given the K top-ranked chunks extracted from the vector database

Figure 5.1 illustrates the accuracy of various Large Language Models when collecting data with the DPR retrieval technique. *K* represents the number of text chunks extracted from the vector database and provided to the LLM. The left figure 5.1a displays the accuracy of the generated answers across all the questions from the Mintaka dataset. Given that approximately 80% of these questions are linked to Wikidata entries present in the T-REx dataset, we additionally present results in figure 5.1b on the right, calculated on a subset of the questions linked to entries found within the population dataset.

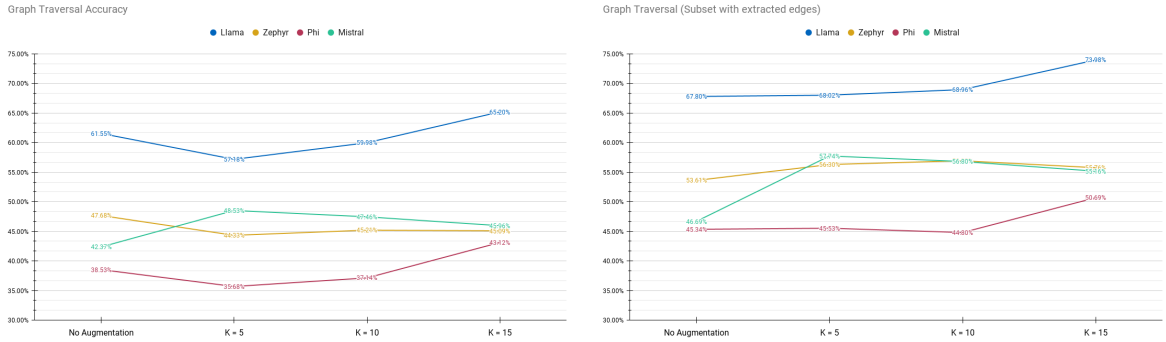
Accuracy generally improves as more information is added to the prompt. Specifically, Mistral and Phi showed enhanced performance when presented with data compared to results with no added context from RAG methods. However, Llama and Zephyr demonstrated a similar trend of decreased accuracy when little data was retrieved, then reaching the accuracy of responses with no augmentation as the quantity of data increased. However, when comparing the results between figure 5.1a and figure 5.1b, after excluding questions with answers that are not found in T-REx, the accuracy gain with added information becomes more profound. Mistral and Phi's improvement increases from 8% to 10% at $K = 15$. At the same time, Zephyr surpassed the accuracy of the no-context baseline at $K = 10$, and Llama shows a reduced decline compared to the baseline at the same K value.

The variation in results among different LLMs may stem from different factors: Attention dilution or the limitation of context windows could have an impact on the results. As more data is included in the prompt, the model's effectiveness in concentrating on the most relevant parts of the information could be reduced. This effect could explain Mistral and Zephyr's stable results on larger K values. However, this phenomenon does not address the problem of the initial decrease in accuracy when few contexts are provided, particularly with Llama and Phi, which showed a consistent upward trend in accuracy as more data was added.

One preliminary hypothesis is that the models, potentially trained on Wikipedia data or other data containing similar info, expect the provided information from the retriever to contain the answer. However, in many cases, especially at lower K values, the retriever process may fail to augment the model with valuable information. Those models already storing most of the answers in their trainable parameters can provide promising results without additional retrieval context when asked the evaluation question. The observed decline in accuracy when the information provided lacks relevant answers may be due to an "obedience" factor, suggesting that models anticipate helpful information and may struggle to respond appropriately when the collected data at lower K levels is irrelevant.

Given that in figure 4.3, we demonstrated the usability of the T-REx dataset in augmenting LLMs on the Mintaka question, the decline in performance due to lack of irrelevant data is on the chunk level selected by the retriever. For this study, we supplied the models with the top-ranked documents to accurately assess the differences among RAG techniques. However, in practice, we recommend implementing a threshold for document relevance to ensure that only pertinent information is provided to the LLMs.

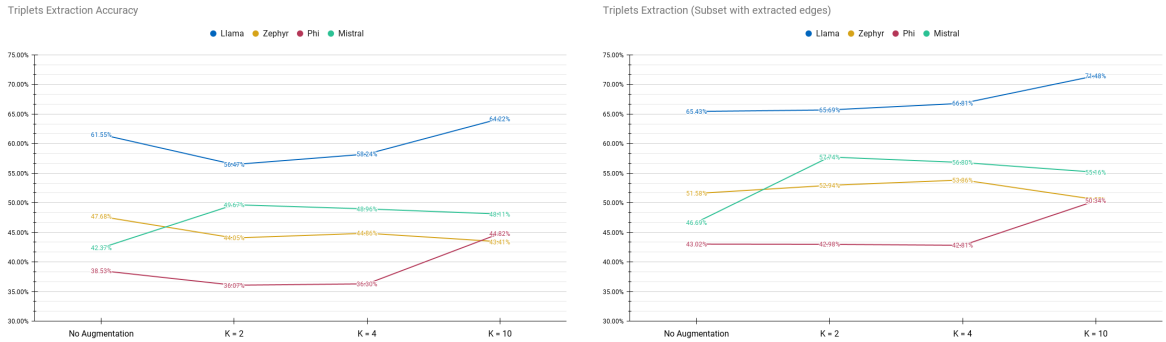
5.2 Results of the Graph-based RAG Techniques



(a) Accuracy on all the Mintaka questions

(b) Accuracy on a subset of Mintaka questions where an edge was extracted at $K \leq 5$ steps

Figure 5.2: Accuracy of LLMs answers using the Graph Traversal RAG technique given the K number of traversals



(a) Accuracy on all the Mintaka questions

(b) Accuracy on a subset of Mintaka questions where an edge was extracted at $K \leq 2$ steps

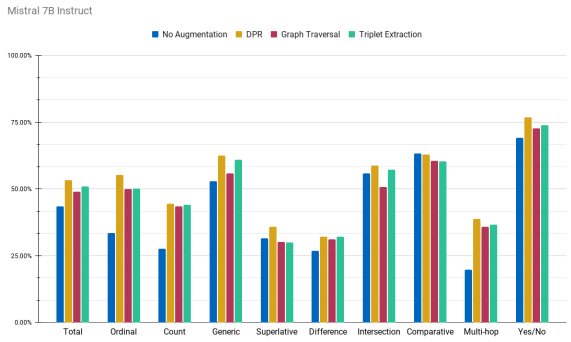
Figure 5.3: Accuracy of LLMs answers using the Triplets Extraction RAG technique given the K number of retrieved triplets

Figures 5.2 and 5.3 display the accuracies of each LLM model utilizing the Graph Traversal and the Triplets Extraction retrieval techniques, respectively. The figures on the left side (5.2a and 5.3a) represent the accuracy across all questions from the evaluation dataset. In contrast, the figures on the right side (5.2b and 5.3b) present the accuracies on a subset of the questions to illustrate the impact of including edges in the collected data.

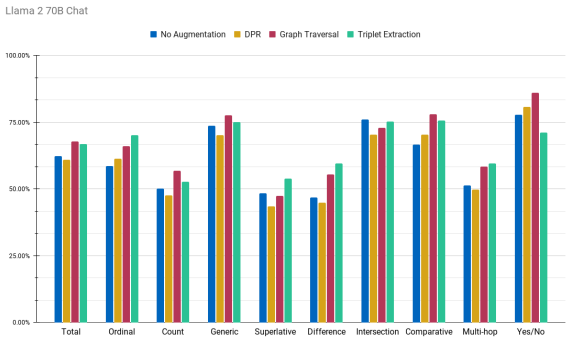
In figure 5.2b, we calculate the accuracy of the Graph Traversal RAG system by only considering questions where the retrieval traverses the graph at least once with $K \leq 5$ instead of relying on candidate nodes from the vector database. This accounts for 29.53% of the questions. Furthermore, figure 5.3b assesses the accuracy of the Triplets Extraction technique on the subset of questions where this technique captures at least one edge in the graph at $K \leq 2$ instead of ranking singular nodes higher. This subset represents 40.95% of the questions.

Overall, both retrieval techniques demonstrate strong performance at higher K values across most of the models, including Llama and Phi. When using the Mistral model, the graph techniques slightly improve at a lower K. In contrast, with the Zephyr model, the accuracy remains stable compared to the answers with no retrieved data. When filtering questions based on edge extraction at an early retrieval step, the figures on the right reveal that the graph techniques are less likely to experience a decline in performance compared to results without additional context, flattening the decreasing points corresponding to the figures on the left side. Additionally, Zephyr, which initially displayed stable yet slightly lower accuracy values, exhibits slightly increased accuracy after filtering.

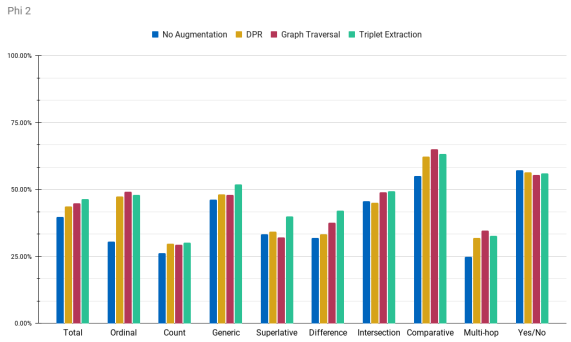
Both graph extraction techniques show promising results in augmenting LLMs and demonstrate robustness in maintaining or even improving performance. Moreover, the filtered results highlight the improvement in accuracy when edge data is provided, suggesting that including comprehensive graph details can benefit retrieval performance.



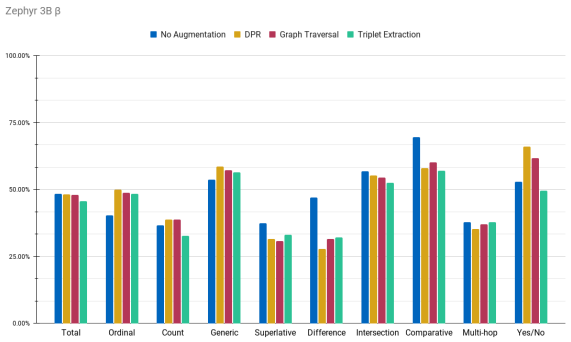
(a) Accuracy of Mistral 3B Instruct



(b) Accuracy of Llama 2 70B Chat



(c) Accuracy of Phi 2



(d) Accuracy of Zephyr 3B Beta

Figure 5.4: Accuracy of LLMs-generated responses across different RAG techniques, segmented by question complexity

Figure 5.4 showcases the performance of the four LLMs we experimented with across the various RAG systems; the accuracy metric is further split by question complexity to analyze the impact of the RAG systems in depth.

Mistral demonstrates superior performance with the DPR technique overall compared to the graph techniques, particularly excelling in "Generic", "Yes/No", "Ordinal", "Count", and "Multi-Hop" questions.

When using Zephyr, although the results are generally similar to Mistral, the Triplets Extraction technique outperforms DPR in "Difference", "Multi-Hop", and "Superlative" questions. Additionally, the Graph Traversal technique surpassed DPR in "Comparative" questions. Notably, Zephyr shows better performance when the context is absent in many of the complexities where graph techniques excel, including "Difference", "Comparative", "Intersection", and "Superlative" questions.

In contrast, for the Phi 2 model, the graph-based techniques generally yield the highest accuracy across many question complexities, including "Multi-hop", "Comparative", and "Ordinal" questions. The Triplets Extraction technique has the best performance overall, specifically in "Superlative" and "Difference" questions and even in "Generic" questions. All RAG techniques struggled to answer "Yes/No" questions and could not improve on the accuracy compared to results with no augmentation.

Observing the results from Llama, the Triplets Extraction technique has the best performance on "Superlative", "Difference", "Ordinal", and "Multi-hop" questions, whereas the Graph Traversal technique excels in all remaining question complexities. While the DPR technique decreases the performance compared to the results with no context, it holds a high accuracy on "Yes/No" questions, surpassing the Triplets Extraction RAG method.

Overall, in three out of the four models, graph-based retrieval achieved higher accuracy compared to vector-based RAG techniques for "Superlative," "Multi-Hop," "Difference," and "Comparative" questions. Meanwhile, the DPR technique is more effective in "Yes/No" questions across three models, with two also showing DPR scoring the highest on "Generic" questions.

5.3 Results of the Cypher Generation RAG Technique

		Cypher Query on the Full Graph	
		Invalid Query	**Valid Query**
Cypher Query on a Subgraph	**Invalid Query**	14740	1292
	Valid Query	3673	295

Table 5.1: Confusion matrix showing the number of valid Cypher queries generated, comparing results from the full graph versus a target subgraph

Table 5.1 displays a confusion matrix that evaluates the validity of Cypher queries generated using two approaches: one on an extracted subgraph and the other with the entire graph structure. In this table, a Cypher query is considered valid if it is free from any syntax errors and if it was able to successfully retrieve data from the graph, regardless of the utility of the data.

The data showcases a significant difference in the quality of Cypher queries, with the subgraph-based generation of Cypher queries resulting in 3,968 valid queries, which is double the 1,587 valid queries generated from the entire graph. This highlights the improved effectiveness of generating valid and executable Cypher queries when providing detailed information on a target subgraph. However, over 73% of the questions remain without a valid Cypher query from either technique.

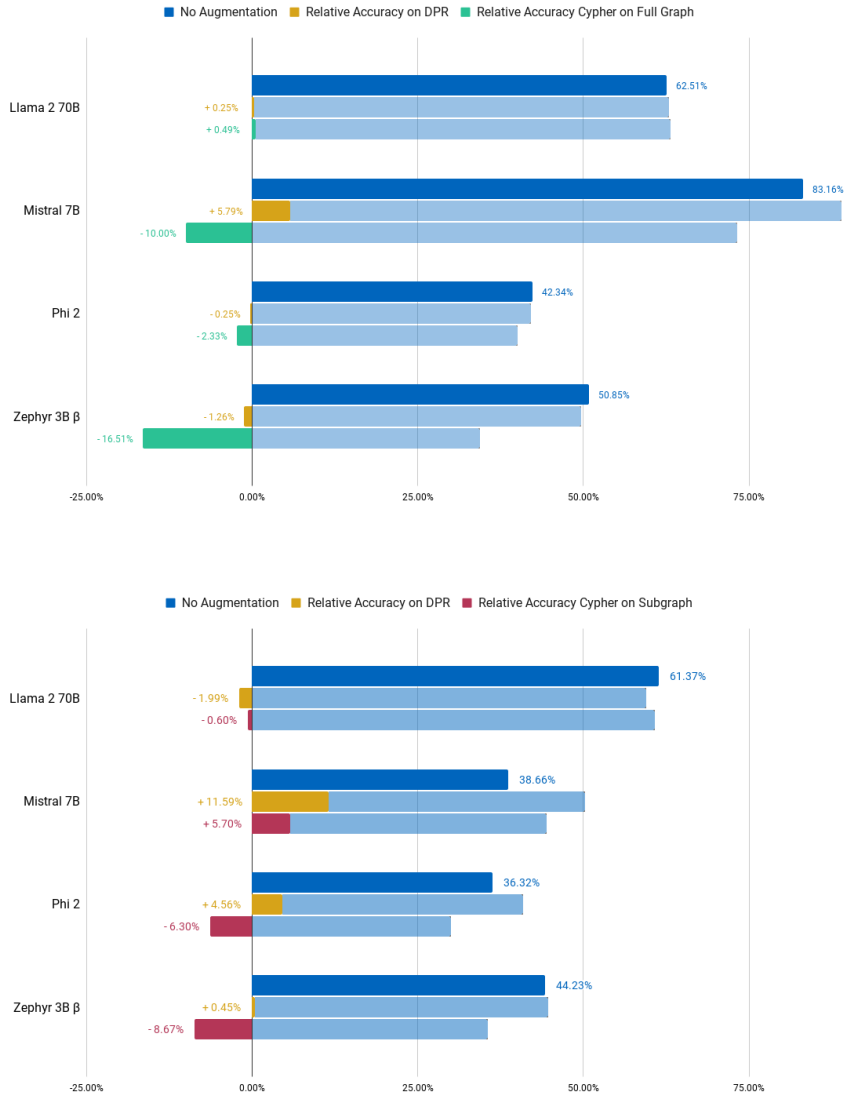


Figure 5.5: The relative accuracy of the DPR and Cypher query generation techniques compared to the accuracy without augmentation

Further analyzing these valid queries, we calculate the accuracy results of the generated answers from the LLMs, as displayed in figure 5.5. For a more rigorous comparison, the accuracy is assessed only on the subset of questions where the Cypher query was considered valid. Additionally, we included the accuracy of the DPR technique calculated on the same subset to ensure a comprehensive evaluation. To compare results across the different Cypher generation techniques, calculated across different question subsets, we displayed the relative accuracy, quantifying the difference in accuracy compared to the results obtained without augmentation.

Regarding relative accuracy, the DPR technique generally outperforms Cypher generation techniques across all models except Llama. Notably, the subgraph approach outperforms the complete graph approach when augmenting Mistral and Zephyr with the data collected from the generated Cypher queries. However, Llama and Phi performed better when the entire graph was used for the Cypher generation.

An interesting observation from the analysis is the increase in accuracy of results without augmentation when a subgraph is utilized, compared to the subset of questions where the Cypher queries generated from the complete graph were valid. This outcome suggests that the subgraph technique may be more proficient at handling complex or less straightforward queries.

5.4 Retrieval Runtime

RAG Technique	**Inference Time /s**			
	$K^2 = 1$	$K^2 = 2$	$K^2 = 5$	$K^2 = 15$
DPR	0.928s	0.928s	0.928s	0.928s
Cypher Generation on the full graph	3.362s	-	-	-
Cypher Generation on a subgraph	4.338s	-	-	-
Triplets Extraction	1.054s	1.054s	1.054s	1.054s
Graph Traversal	0.928s	0.975s	1.426s	3.808s

Table 5.2: Runtime of the retrieval process of each RAG technique

Table 5.2 shows the average inference times in seconds of the retrieval processes of each RAG technique. For DPR and the subgraph selection of Triplets Extraction, Graph Traversal, and Cypher generation RAG techniques, the value K^1 of the initial retrieval, which indicates the number of documents extracted and ranked from the vector database, is set to 100. The target subgraphs are also defined using 20 nodes. The Large Language Models, including the embedding and re-ranking components, are executed on an Nvidia Tesla T4 GPU, while all databases are hosted on a virtual machine with four virtual CPUs and 16 GB of memory.

In DPR and Triplets Extraction, the final retrieval count K^2 does not affect inference time, as all documents are gathered, ranked, and sorted prior to selecting the subset of documents passed to the LLM. Triplets Extraction shows a higher inference time than DPR because of the extra steps of extracting and re-ranking edges within the subgraph. The difference is insignificant due to the shorter inputs used in triplet ranking, which include only the node and edge titles.

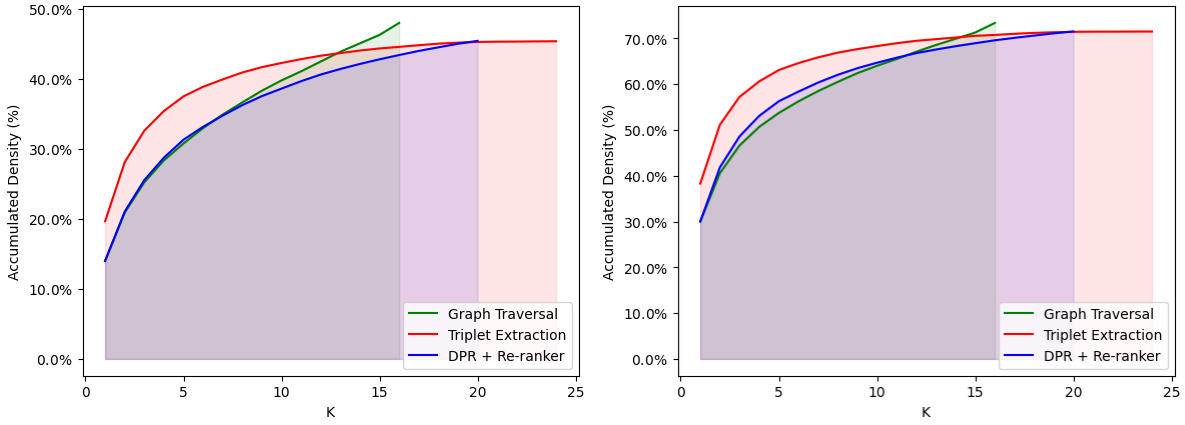
The inference time of the Graph Traversal technique increases with K^2 due to the need to extract neighboring nodes at each K^2 step and recalculate new ranking scores for candidates. Moreover, the increase in inference time is exponential with K^2. At each step, a subset

of at least K^2 candidates is ranked, including K^2 filtered candidates of previous stages and newly identified neighboring nodes. Graph Traversal is equivalent to DPR at $K^2 = 1$ since no traversals occur.

For Cypher query generation, inference time largely depends on the query structure, particularly on the number of results and filtering properties. However, query generation takes up most of the runtime, with Phi taking an average of 2.5 seconds to generate a query in our setup.

5.5 Results on the Retrieval

The Mintaka question-answer pair dataset links each pair to relevant Wikidata entries, allowing us to identify the correct documents each retrieval technique must retrieve for each question. This preliminary step enables us to independently evaluate the retrieval techniques before generating the answers from the LLMs. Figure 5.6 displays the accumulated accuracy of capturing correct information at retrieval based on the number of retrieved documents, denoted by K.



(a) Calculated based on retrieving correct documents at each K retrieval step

(b) Calculated based on retrieving one correct document per question at each K retrieval step

Figure 5.6: Cumulative accuracy of the retrieval of each RAG technique

Mintaka contains two sets of entities per question: *QuestionEntities*, a set of Wikidata articles listing entities found in the question, and *AnswerEntities*, a set of Wikidata articles that potentially contain the answer to the question. All entities are annotated manually by the authors of the Mintaka dataset. Unlike in table 4.4, where the MRR is calculated only on the *AnswerEntities*, figure 5.6 and table 5.3 calculate the retrieval accuracy on a concatenated list of both *QuestionEntities* and *AnswerEntities*. This is particularly helpful with "Count" and "Yes/No" questions, where the answers are not linked to any Wikidata article, making the most relevant information likely to be found in the *QuestionEntities* set. Consequently, as

each question is linked to one or multiple entities in Wikidata, we calculated the accuracies using two different metrics.

Figure 5.6a calculates the accuracy per document/entity linked to the questions. For each set of correct documents associated with a question, we track and visualize the specific K value at which the retrieval technique finds each correct document. Thus, the accuracy is calculated independently of the question, providing a granular view of each retrieval technique's effectiveness in identifying all relevant documents across varying depths of retrieval.

Conversely, figure 5.6b calculates the accuracy per question in Mintaka by measuring whether at least one correct document is retrieved for each question at varying K values. It focuses on the retrieval technique's ability to capture at least one correct article within the top K documents rather than ensuring all correct documents are retrieved.

Retrieval Technique	**MRR@15**	**NDCG@15**
DPR + Re-ranking	0.41426	0.35626
Graph Traversal	0.40982	0.36820
Triplets Extraction	0.48982	0.41803

Table 5.3: Retrieval results of the different RAG techniques

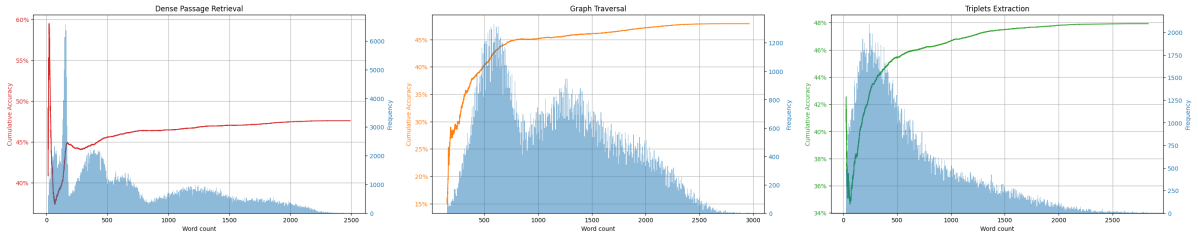
Table 5.3 presents the MRR [65] and Normalized Discounted Cumulative Gain (NDCG) [70] metrics for each technique, calculated with $K = 15$. Those metrics are used to evaluate the effectiveness of information retrieval systems. MRR focuses on the retrieval position of the first relevant document, providing a single score based on the reciprocal of the rank at which the first relevant result is found, emphasizing the importance of the top result. In contrast, NDCG considers the positions of all relevant documents and applies a logarithmic discount based on their ranks, making it a more comprehensive measure that values the quality of the entire list of retrieved documents.

We can observe that the Triplets Extraction technique consistently outperforms all other techniques in retrieving the correct data and then plateaus at $K = 20$. This plateau occurs because this technique is limited by the initial number of nodes that defined the subgraph, which was set to 20. With higher MRR and NDCG values, the triplets retrieval method demonstrates significant improvement in ranking data when edges are taken into consideration. Conversely, the Graph Traversal technique performs similarly to the DPR technique at early retrieval stages, as the initial seeds are extracted from the vector database. However, it eventually surpasses the embedding technique at a particular K value. This phenomenon is evident in both figures 5.6a and 5.6b, although the point at which the Graph Traversal technique surpasses both the DPR technique and Triplets Extraction technique occurs at lower K values when looking into the accuracy calculated per document 5.6a.

When observing the MRR and NDCG metrics in table 5.3, the Triplets Extraction achieves the highest scores compared to all other retrieval techniques. Moreover, DPR scores are higher in MRR, while the Graph Traversal technique achieves a higher NDCG score. While MRR is calculated considering the index of the first correctly retrieved document per question, NDCG accounts for all indices of the correctly retrieved documents. These scores, alongside the varying intersection point in figure 5.6, indicate that the DPR is particularly more effective in retrieving a correct document early in the retrieval process, whereas the Graph Traversal technique performs better when the objective is to identify as many correct documents as possible.

5.6 Accuracy over Prompt Size

In the figures 5.7, 5.8, 5.9, 5.10, 5.11 and 5.12, we present the accuracy scores in relation to sizes of the prompt or accumulated retrieved data. This method helps us assess the quality of retrieved data instead of merely relying on K values, which could produce varying prompt sizes across different RAG systems. For all of the mentioned figures, we counted the words in each constructed prompt and calculated the cumulative accuracy of the generated responses as the prompt sizes increased. For example, at a word count of 1000, we measure the accuracy of answers generated from prompts with sizes up to 1000 words.



(a) Results of the DPR technique (b) Results of the Graph Traversal technique (c) Results of the Triplets Extraction technique

Figure 5.7: Cumulative accuracy of generated responses of all LLMs, over the number of words in a prompt

Figure 5.7 presents the cumulative accuracy relative to the word counts of the prompts for each RAG technique, with the size distribution of prompts shown in blue. In figure 5.8, we combine these accuracies into a single chart for a more precise comparison across all retrieval methods. All techniques show a trend of increasing accuracy as prompts grow larger. The incline is more evident in graph-based techniques, whereas the DPR technique initially shows a sudden decrease before a slow and gradual improvement. The Graph Traversal techniques experience a steeper incline in accuracy. While both DPR and Triplets Extraction techniques tend to converge, this pattern may be influenced by the distribution of the prompt sizes with fewer data points at higher word counts.

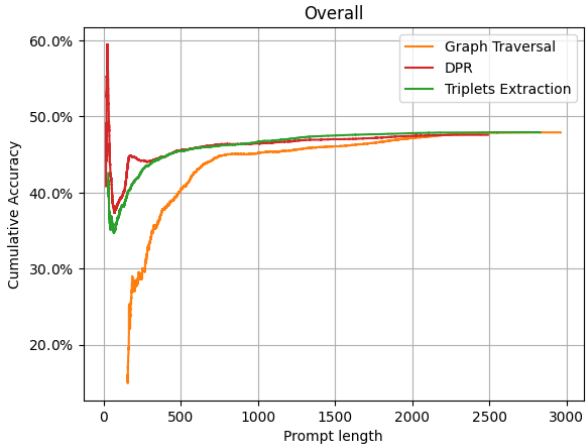
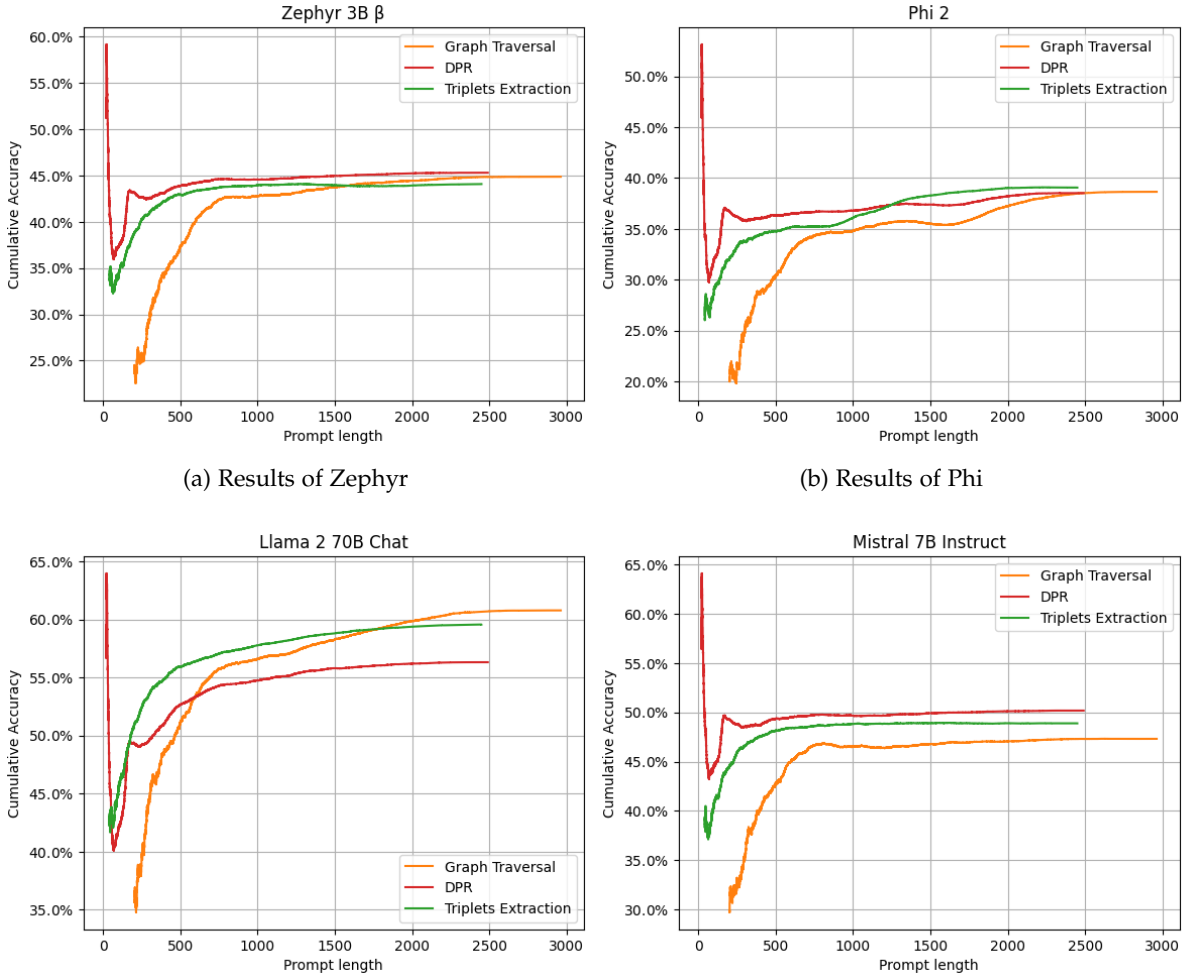


Figure 5.8: Cumulative accuracy of generated responses of all LLMs over the number of words in a prompt, segmented by RAG techniques

DPR has been experimented with a few retrieval steps, increasing the density of prompts with smaller word counts. Prompts with sizes smaller than 100 resulted in a peak in accuracy, reaching 60%. This only highlights the re-ranker model favoring small article chunks if sufficient. However, as prompts grow, the graph-based techniques achieve higher accuracy gains. This suggests that DPR is more effective in identifying relevant information at the initial stages, while graph-based techniques excel at accumulating and leveraging useful information over time.



(a) Results of Zephyr

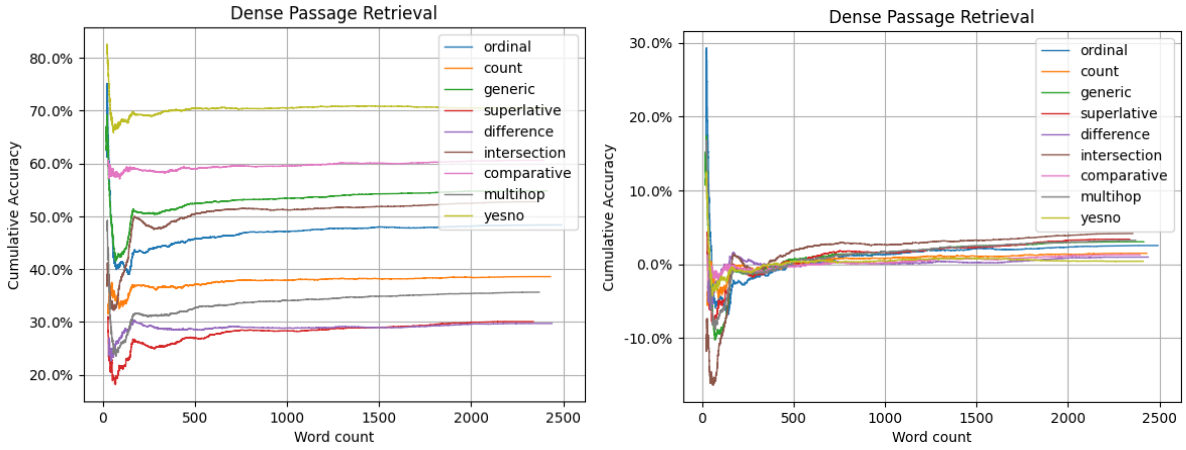
(b) Results of Phi

(c) Results of Llama

(d) Results of Mistral

Figure 5.9: Cumulative accuracy of generated answers over the prompt size per RAG system, segmented by LLM

Figure 5.9 presents the calculated cumulative accuracy relative to the number of words in each prompt, segmenting results by language models. We observe that the graph-based techniques enhance accuracy on the Llama and Phi models as the number of words increases. In contrast, DPR exhibits relative stability across all LLMs, showing only a slight and gradual increase in accuracy with increasing word counts. Moreover, DPR performs best with the Mistral and Zephyr models, primarily due to the weaker performance of the graph-based techniques in those models, which also tend to plateau at larger prompt sizes. This indicates that Mistral and Zephyr may be unable to fully benefit from the advantages offered by graph-based techniques at higher word counts.



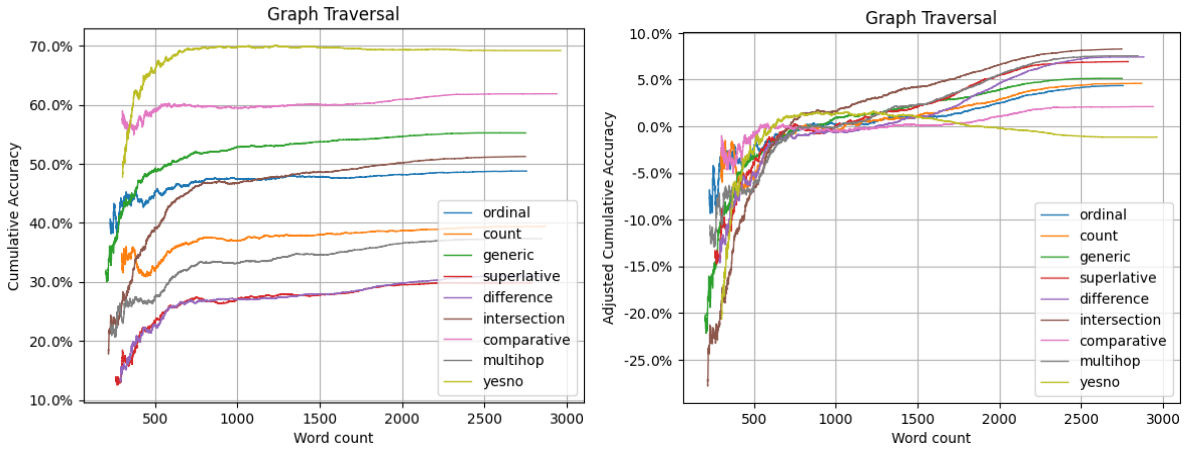
(a) Cumulative accuracy over the prompt size

(b) Adjusted cumulative accuracy over the prompt size

Figure 5.10: Cumulative accuracy of all LLM-generated answers over the prompt size, using the DPR technique, segmented by question complexity

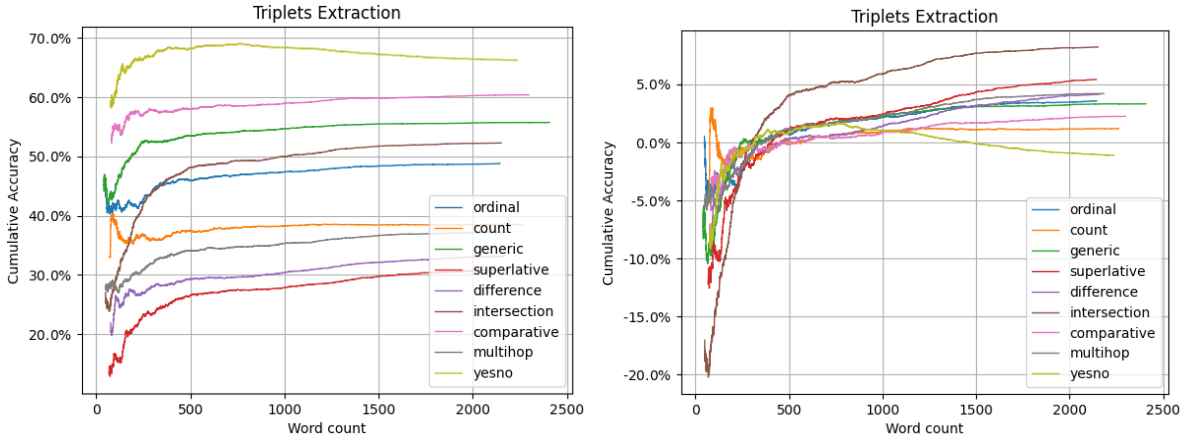
Figures 5.10, 5.11, and 5.12 display the cumulative accuracy relative to the word counts of prompts, segmented by question complexities. We calculate the cumulative accuracy of each RAG method, focusing on the development of the accuracy across different question types as prompts increase in size. Figures 5.10b, 5.11b, and 5.12b calculate an adjusted cumulative accuracy aligning the values of different question complexities. This adjustment is achieved by subtracting the mean of each complexity, allowing a focused analysis on the improvement of accuracy specific to each question type, enabling the comparison of question complexities.

All techniques follow a similar order of performance across complexity types, with "Yes/No," "Comparative," and "Generic" questions achieving the highest accuracies, while "Superlative," "Differential," and "Multi-hop" questions result in the lowest accuracy. More complex questions that initially present lower accuracy benefit the most from increased prompt sizes. This pattern is especially evident in the adjusted cumulative accuracy graphs, where those particular complexities show higher accuracy improvements at higher word counts. The cumulative accuracy on the DPR method across all question complexities tends to stabilize as the number of words increases. However, graph-based techniques, both Triplets Extraction and Graph Traversal, demonstrate an inclining trend in accuracy on more complex questions as prompts get larger. In contrast, simpler questions like "Yes/No" may decline in accuracy with graph-based methods. "Intersection", "Multi-hop", and "Difference" questions particularly benefit more from the Graph Traversal technique as data accumulates, whereas "Intersection" and "Superlative" questions experience a more profound incline at higher word counts with the Triplets Extraction method. These question types are especially responsive to the graph-based method as data increases and more information from the graph's edges is provided.



(a) Cumulative accuracy over the prompt size (b) Adjusted cumulative accuracy over the prompt size

Figure 5.11: Cumulative accuracy of all LLM-generated answers over the prompt size, using the Graph Traversal technique, segmented by question complexity



(a) Cumulative accuracy over the prompt size (b) Adjusted cumulative accuracy over the prompt size

Figure 5.12: Cumulative accuracy of all LLM-generated answers over the prompt size, using the Triplets Extraction technique, segmented by question complexity

Through our analysis of cumulative accuracy across different LLMs and in relation to various question complexities, we found that some models, such as Mistral and Zephyr, experience a decline in performance with graph-based techniques. In contrast, models like Llama and Phi demonstrate notable improvements when using Graph Traversal or Triplets Extraction techniques. Additionally, graph-based methods have shown higher effectiveness in addressing more complex questions, whereas their performance tends to diminish with broad questions. Therefore, a possible theory for Mistral and Zephyr's poor performance is their inability to effectively process and respond to complex questions. While graph-based techniques boost the inherent capabilities of models such as Llama and Phi in handling complex questions, they may not compensate for models less capable of processing such complexities.

6 Conclusion

This study analyzed various RAG techniques, comparing their performance in enhancing generative language models for question-answering tasks. We assessed the effectiveness of RAG systems that retrieve information from vector databases versus graph databases across different question types.

6.1 Findings

Our results reveal that graph-based RAG techniques outperform vector-based methods on complex questions that require multiple retrieval steps. Conversely, vector-based RAG systems are more efficient in speed and excel at general questions that can be answered with a single extract. While vector-based RAG methods are powerful at extracting one relevant document early in the retrieval process, graph-based RAG techniques are more effective at retrieving an overall more accurate collection of data.

Acknowledging the capabilities of the selected LLM to generate responses is essential. As reflected in the results, some LLMs are unable to benefit from the information provided by graph-based techniques and handle complex questions. Language models that struggle to process and comprehend complex queries cannot correctly respond to questions containing multiple rules and conditions, even if the provided information is sufficient.

In many cases in our experiments, accuracy dropped as retrieved information increased. For example, the accuracy of answers to "Yes/No" questions decreased for graph-based methods as prompt length increased. A possible reason is the inclusion of data related to the general topic but does not contain the expected answers, which complicates the gathered information for the LLMs. The overall accuracy also decreased in the early retrieval steps of all RAG techniques compared to the accuracy without augmentation. The decrease is more prominent in the subset of questions that did not contain answers in the population dataset. This further proves the LLMs' expectation and reliance on finding answers in the retrieved data, which is not always the case, particularly in early retrieval steps. This underscores the importance of filtering irrelevant data prior to augmenting the LLM.

Cypher query generation is more advanced and flexible, yet it remains unreliable. In our experiments, these methods were unable to improve the accuracy of generated answers, with lower performance than the DPR technique. Additionally, they only generated valid queries for a limited subset of questions. Utilizing LLMs with solid coding abilities or fine-tuned

on Cypher queries enhances the likelihood of generating valid queries. Overall, using the re-ranker models for ranking and extracting data from the KG proves more effective than generating Cypher queries.

6.2 Limitations

RAG systems are highly dependent on the domain and dataset content. Our results relied on the Wikidata-based T-REx dataset, but the outcomes may vary across different datasets. Factors that could alter the results include the sparsity and logical structure of relationships in the Knowledge Graph, as well as the conciseness of the textual data that defines the number of retrieval hops required to answer a question. While our findings provide a foundational understanding of the differences between embedding-based and graph-based RAG systems, the specific outcomes may vary when applying these methods to different domains.

The Cypher generation RAG technique presents another limitation. References that introduced this RAG technique relied on powerful generative models such as ChatGPT-4. However, due to limited resources, we only tested this approach using smaller models that lacked the understanding capabilities and struggled to follow all the instructions for effective Cypher query generation. While Phi successfully produced valid Cypher queries for 20% of the questions, the retrieved data lacked the valuable information needed to augment the LLMs.

Furthermore, the RAG systems we investigated were highly dependent on the cross-encoder ranking model, which has inherent limitations. The performance of LLMs hinges on the accuracy of the re-ranker and its ability to identify relevant information. The re-ranker model also defines the subgraph of the Triplets Extraction method and the initial seeds of the Graph Traversal method, which limits the search to specific graph areas. For our experiments, we used a model with a small context window that worked well for chunked text but struggled to rank concatenated retrieved data accurately. Instead, it calculated the ranks of newly retrieved data independently of previous steps, resulting in repetitions and lacking a logical flow. Unlike search engines, which aim to deliver the most relevant web pages independently, RAG systems aim to provide an optimal data collection to support LLMs, creating dependencies between the documents. We attempt to mitigate this by continuously including node titles and edge information from prior retrieval steps for the Graph Traversal technique; however, titles alone lack the full context of the documents.

6.3 Future Work

Future research expands the evaluation of vector-based and graph-based RAG systems across different datasets or domains. As results could vary depending on the dataset, such an investigation will further solidify and generalize our findings on the differences between vector and graph databases in RAG techniques. Notably, our study uses a population dataset based on general information that may align well with the training data of LLMs. Therefore, examining

the performance of question-answering RAG systems on exclusive or closed-source datasets could yield promising findings. Furthermore, future work could explore algorithms for KG construction given unstructured textual data and analyze the performance of graph-based RAG systems across different construction factors.

In the context of Cypher generation, providing minimal information on the structure of the graph or subgraph is insufficient. One potential approach to improving the quality of Cypher queries is to feed the results back to the LLM for iterative refinement. In some cases, invalid Cypher queries arise not necessarily from syntax errors but from misuse of relationships, labels, and other variables due to incorrect assumptions about the graph structure. Therefore, providing the LLM with the query results or errors would give it further insights into the graph's content, potentially improving the query at each step. It is worth investigating flexible techniques like Cypher generation and adequately assessing its performance against embedding-based RAG methods.

Our study demonstrates that providing LLMs with relevant information is crucial to improving response quality, otherwise, they may "hallucinate". Future work should explore techniques to classify the relevance of cumulative data and establish thresholds or stopping criteria once sufficient information has been gathered. While our current methods utilize a cross-encoder that scores the relevance of information based on the question, they lack a threshold to exclude irrelevant documents. Results may vary across different RAG techniques with the inclusion of stopping criteria. Additionally, it is worth investigating the number of steps required for each RAG technique to find sufficient data. Techniques, such as the one proposed by Carraro and Bridge [71], could rank data higher not just by its independent relevance to the query but also by its cumulative usefulness on the previous retrievals, diversifying the overall collected data. The rank score's dependency on previous data could additionally benefit from search algorithms such as beam search, as proposed by J. Zhang, H. Zhang, D. Zhang, et al. [72].

While retrieving data that contains the correct answer to the query is vital, minimizing irrelevant content is advantageous. The accuracy of generated answers decreases in the early retrieval stages, showcasing the LLMs' inability to filter out irrelevant information. Extractive summarization techniques could refine the retrieved content at each retrieval step and extract relevant information from the documents before providing them to LLMs. Arumae and F. Liu [73] proposed an extractive model that is fine-tuned for question-answering tasks and considers user queries when extracting relevant information from documents. Such techniques could be beneficial for RAG systems that require many retrieval steps, ensuring that accumulated information does not exceed the LLM's context window. Future work could study the advantages of such techniques across various embedding-based and graph-based RAG techniques and investigate whether the results align with our findings.

List of Figures

3.1 The cumulative success rates of data retrieval using DPR before and after re-ranking . . . 10

4.1 Overview of the pipeline and components used to produce the T-REx dataset, produced by Elsahar, Vougiouklis, Remaci, et al. [44] . . . 15
4.2 Response accuracy of different LLMs to the Mintaka question without data augmentation, categorized by question complexity . . . 17
4.3 Accuracy of LLMs answers on the Mintaka questions when augmented with the correct Wikidata article from T-REx . . . 18
4.4 The setup and inference processes of the DPR technique . . . 22
4.5 the structure of the prompt used to generate Cypher queries from the complete graph database . . . 25
4.6 The process of extracting a subgraph from the graph database . . . 26
4.7 The steps for retrieving data using the Graph Traversal RAG method . . . 28
4.8 Python code implementing the Graph Traversal RAG technique . . . 29
4.9 Example of selecting and ranking candidates in the Triplets Extraction technique 31
4.10 The steps for the Triplets Extraction technique . . . 32

5.1 Accuracy of the LLMs responses using the DPR method given the K top-ranked chunks extracted from the vector database . . . 33
5.2 Accuracy of LLMs answers using the Graph Traversal RAG technique given the K number of traversals . . . 35
5.3 Accuracy of LLMs answers using the Triplets Extraction RAG technique given the K number of retrieved triplets . . . 35
5.4 Accuracy of LLMs-generated responses across different RAG techniques, segmented by question complexity . . . 37
5.5 The relative accuracy of the DPR and Cypher query generation techniques compared to the accuracy without augmentation . . . 39
5.6 Cumulative accuracy of the retrieval of each RAG technique . . . 41
5.7 Cumulative accuracy of generated responses of all LLMs, over the number of words in a prompt . . . 43
5.8 Cumulative accuracy of generated responses of all LLMs over the number of words in a prompt, segmented by RAG techniques . . . 44
5.9 Cumulative accuracy of generated answers over the prompt size per RAG system, segmented by LLM . . . 45

5.10 Cumulative accuracy of all LLM-generated answers over the prompt size, using the DPR technique, segmented by question complexity 46
5.11 Cumulative accuracy of all LLM-generated answers over the prompt size, using the Graph Traversal technique, segmented by question complexity 47
5.12 Cumulative accuracy of all LLM-generated answers over the prompt size, using the Triplets Extraction technique, segmented by question complexity 47

List of Tables

4.1 The nine different complexity types of the Mintaka dataset questions 16
4.2 Evaluation of various verdict prediction models on the generated samples . . . 20
4.3 List of LLMs used to generate responses for our experiments 21
4.4 Retrieval score of different embedding models and chunking techniques 22

5.1 Confusion matrix showing the number of valid Cypher queries generated, comparing results from the full graph versus a target subgraph 38
5.2 Runtime of the retrieval process of each RAG technique 40
5.3 Retrieval results of the different RAG techniques 42

Acronyms

BLEU Bilingual Evaluation Understudy. 13, 14, 18, 19

DPR Dense Passage Retrieval. 5–10, 12, 21, 22, 25, 27, 30, 33, 37–46, 49, 52, 53

FiD Fusion-in-Decoder. 7

HNSW Hierarchical Navigable Small World. 21

IR Information Retrieval. 5

KG Knowledge Graph. iv, 7, 8, 11–13, 25, 30, 50, 51

LLM Large Language Model. iv, 1–5, 7, 8, 11–21, 23–27, 32–37, 39–41, 43–54

MRR Mean Reciprocal Rank. 21, 41–43

NDCG Normalized Discounted Cumulative Gain. 42, 43

NLM Neural-based Language Models. 4

NLP Natural Language Processing. 1, 4–6

RAG Retrieval-Augmented Generation. iv, vii, 1–9, 11, 13, 15–18, 20, 27–30, 33–38, 40–46, 49–52, 54

REALM Retrieval Augmented Language Model Pre-Training. 6

RETRO Retrieval-Enhanced Transformer. 7

ROUGE Recall-Oriented Understudy for Gisting Evaluation. 13, 18, 19

Seq2Seq Sequence-to-Sequence. 1, 4, 6

TF-IDF Term Frequency-Inverse Document Frequency. 5, 7–9, 27

Bibliography

[1] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, et al. "Improving language understanding by generative pre-training". In: (2018).

[2] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova. "Bert: Pre-training of deep bidirectional transformers for language understanding". In: *arXiv preprint arXiv:1810.04805* (2018).

[3] H. Touvron, T. Lavril, G. Izacard, X. Martinet, M.-A. Lachaux, T. Lacroix, B. Rozière, N. Goyal, E. Hambro, F. Azhar, et al. "Llama: Open and efficient foundation language models". In: *arXiv preprint arXiv:2302.13971* (2023).

[4] A. Q. Jiang, A. Sablayrolles, A. Mensch, C. Bamford, D. S. Chaplot, D. d. l. Casas, F. Bressand, G. Lengyel, G. Lample, L. Saulnier, et al. "Mistral 7B". In: *arXiv preprint arXiv:2310.06825* (2023).

[5] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin. "Attention is all you need". In: *Advances in neural information processing systems* 30 (2017).

[6] P. Lewis, E. Perez, A. Piktus, F. Petroni, V. Karpukhin, N. Goyal, H. Küttler, M. Lewis, W.-t. Yih, T. Rocktäschel, et al. "Retrieval-augmented generation for knowledge-intensive nlp tasks". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 9459–9474.

[7] W. X. Zhao, K. Zhou, J. Li, T. Tang, X. Wang, Y. Hou, Y. Min, B. Zhang, J. Zhang, Z. Dong, et al. "A survey of large language models". In: *arXiv preprint arXiv:2303.18223* (2023).

[8] R. Rosenfeld. "Two decades of statistical language modeling: Where do we go from here?" In: *Proceedings of the IEEE* 88.8 (2000), pp. 1270–1278.

[9] Y. Bengio, R. Ducharme, and P. Vincent. "A neural probabilistic language model". In: *Advances in neural information processing systems* 13 (2000).

[10] T. Mikolov, K. Chen, G. Corrado, and J. Dean. "Efficient estimation of word representations in vector space". In: *arXiv preprint arXiv:1301.3781* (2013).

[11] J. Pennington, R. Socher, and C. D. Manning. "Glove: Global vectors for word representation". In: *Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP)*. 2014, pp. 1532–1543.

[12] A. Chowdhery, S. Narang, J. Devlin, M. Bosma, G. Mishra, A. Roberts, P. Barham, H. W. Chung, C. Sutton, S. Gehrmann, et al. "Palm: Scaling language modeling with pathways". In: *Journal of Machine Learning Research* 24.240 (2023), pp. 1–113.

[13] M. Lewis, Y. Liu, N. Goyal, M. Ghazvininejad, A. Mohamed, O. Levy, V. Stoyanov, and L. Zettlemoyer. "Bart: Denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension". In: *arXiv preprint arXiv:1910.13461* (2019).

[14] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, and P. J. Liu. "Exploring the limits of transfer learning with a unified text-to-text transformer". In: *Journal of machine learning research* 21.140 (2020), pp. 1–67.

[15] Y. Gao, Y. Xiong, X. Gao, K. Jia, J. Pan, Y. Bi, Y. Dai, J. Sun, and H. Wang. "Retrieval-augmented generation for large language models: A survey". In: *arXiv preprint arXiv:2312.10997* (2023).

[16] X. Li, J. Jin, Y. Zhou, Y. Zhang, P. Zhang, Y. Zhu, and Z. Dou. "From Matching to Generation: A Survey on Generative Information Retrieval". In: *arXiv preprint arXiv:2404.14851* (2024).

[17] G. Salton, E. A. Fox, and H. Wu. "Extended boolean information retrieval". In: *Communications of the ACM* 26.11 (1983), pp. 1022–1036.

[18] H. P. Luhn. "A statistical approach to mechanized encoding and searching of literary information". In: *IBM Journal of research and development* 1.4 (1957), pp. 309–317.

[19] S. Robertson, H. Zaragoza, et al. "The probabilistic relevance framework: BM25 and beyond". In: *Foundations and Trends® in Information Retrieval* 3.4 (2009), pp. 333–389.

[20] V. Karpukhin, B. Oğuz, S. Min, P. Lewis, L. Wu, S. Edunov, D. Chen, and W.-t. Yih. "Dense passage retrieval for open-domain question answering". In: *arXiv preprint arXiv:2004.04906* (2020).

[21] C. Burges, T. Shaked, E. Renshaw, A. Lazier, M. Deeds, N. Hamilton, and G. Hullender. "Learning to rank using gradient descent". In: *Proceedings of the 22nd international conference on Machine learning*. 2005, pp. 89–96.

[22] Z. Cao, T. Qin, T.-Y. Liu, M.-F. Tsai, and H. Li. "Learning to rank: from pairwise approach to listwise approach". In: *Proceedings of the 24th international conference on Machine learning*. 2007, pp. 129–136.

[23] R. Nogueira and K. Cho. "Passage Re-ranking with BERT". In: *arXiv preprint arXiv:1901.04085* (2019).

[24] K. Guu, K. Lee, Z. Tung, P. Pasupat, and M. Chang. "REALM: Retrieval augmented language model pre-training". In: *International conference on machine learning*. PMLR. 2020, pp. 3929–3938.

[25] G. Izacard and E. Grave. "Leveraging passage retrieval with generative models for open domain question answering". In: *arXiv preprint arXiv:2007.01282* (2020).

[26] S. Borgeaud, A. Mensch, J. Hoffmann, T. Cai, E. Rutherford, K. Millican, G. B. Van Den Driessche, J.-B. Lespiau, B. Damoc, A. Clark, et al. "Improving language models by retrieving from trillions of tokens". In: *International conference on machine learning*. PMLR. 2022, pp. 2206–2240.

[27] B. Oguz, X. Chen, V. Karpukhin, S. Peshterliev, D. Okhonko, M. Schlichtkrull, S. Gupta, Y. Mehdad, and S. Yih. "Unik-qa: Unified representations of structured and unstructured knowledge for open-domain question answering". In: *arXiv preprint arXiv:2012.14610* (2020).

[28] Y. Li, B. Peng, Y. Shen, Y. Mao, L. Liden, Z. Yu, and J. Gao. "Knowledge-grounded dialogue generation with a unified knowledge representation". In: *arXiv preprint arXiv:2112.07924* (2021).

[29] S. Weijia, M. Sewon, Y. Michihiro, S. Minjoon, J. Rich, L. Mike, et al. "REPLUG: Retrieval-augmented black-box language models". In: *ArXiv: 2301.12652* (2023).

[30] J. Berant, A. Chou, R. Frostig, and P. Liang. "Semantic parsing on freebase from question-answer pairs". In: *Proceedings of the 2013 conference on empirical methods in natural language processing*. 2013, pp. 1533–1544.

[31] Z. Wang, P. Ng, R. Nallapati, and B. Xiang. "Retrieval, re-ranking and multi-task learning for knowledge-base question answering". In: *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume*. 2021, pp. 347–357.

[32] J. Baek, A. F. Aji, J. Lehmann, and S. J. Hwang. "Direct fact retrieval from knowledge graphs without entity linking". In: *arXiv preprint arXiv:2305.12416* (2023).

[33] Y. Wang, N. Lipka, R. A. Rossi, A. Siu, R. Zhang, and T. Derr. "Knowledge graph prompting for multi-document question answering". In: *Proceedings of the AAAI Conference on Artificial Intelligence*. Vol. 38. 17. 2024, pp. 19206–19214.

[34] B. Fu, Y. Qiu, C. Tang, Y. Li, H. Yu, and J. Sun. "A survey on complex question answering over knowledge base: Recent advances and challenges". In: *arXiv preprint arXiv:2007.13069* (2020).

[35] LangChain. *LangChain: Chain for question-answering against a graph by generating Cypher statements.* LangChain Documentation. Apr. 2024. URL: `https://api.python.langchain.com/en/latest/chains/langchain.chains.graph_qa.cypher.GraphCypherQAChain.html`.

[36] T. Bratanic. *Generating Cypher Queries with ChatGPT-4 on Any Graph Schema*. Neo4j Developer Blog. Apr. 2024. URL: `https://neo4j.com/developer-blog/generating-cypher-queries-with-chatgpt-4-on-any-graph-schema/`.

[37] LlamaIndex. *LlamaIndex: Querying*. LlamaIndex Documentation. Apr. 2024. URL: `https://docs.llamaindex.ai/en/stable/understanding/querying/querying/`.

[38] K. Papineni, S. Roukos, T. Ward, and W.-J. Zhu. "Bleu: a method for automatic evaluation of machine translation". In: *Proceedings of the 40th annual meeting of the Association for Computational Linguistics*. 2002, pp. 311–318.

[39] C.-Y. Lin. "Rouge: A package for automatic evaluation of summaries". In: *Text summarization branches out*. 2004, pp. 74–81.

[40] T. Zhang, V. Kishore, F. Wu, K. Q. Weinberger, and Y. Artzi. "Bertscore: Evaluating text generation with bert". In: *arXiv preprint arXiv:1904.09675* (2019).

[41] A. Obamuyide and A. Vlachos. "Zero-shot relation classification as textual entailment". In: *Proceedings of the first workshop on fact extraction and VERification (FEVER)*. 2018, pp. 72–78.

[42] W. Yin, J. Hay, and D. Roth. "Benchmarking zero-shot text classification: Datasets, evaluation and entailment approach". In: *arXiv preprint arXiv:1909.00161* (2019).

[43] Z. Guo, M. Schlichtkrull, and A. Vlachos. "A survey on automated fact-checking". In: *Transactions of the Association for Computational Linguistics* 10 (2022), pp. 178–206.

[44] H. Elsahar, P. Vougiouklis, A. Remaci, C. Gravier, J. Hare, F. Laforest, and E. Simperl. "T-REx: A large scale alignment of natural language with knowledge base triples". In: *Proceedings of the Eleventh International Conference on Language Resources and Evaluation (LREC 2018)*. 2018.

[45] P. Sen, A. F. Aji, and A. Saffari. "Mintaka: A complex, natural, and multilingual dataset for end-to-end question answering". In: *arXiv preprint arXiv:2210.01613* (2022).

[46] L. Shu, H. Xu, B. Liu, and J. Chen. "Zero-shot aspect-based sentiment analysis". In: *arXiv preprint arXiv:2202.01924* (2022).

[47] M. Laurer, W. Van Atteveldt, A. Casas, and K. Welbers. "Less annotating, more classifying: Addressing the data scarcity issue of supervised machine learning with deep transfer learning and BERT-NLI". In: *Political Analysis* 32.1 (2024), pp. 84–100.

[48] A. Williams, N. Nangia, and S. Bowman. "A Broad-Coverage Challenge Corpus for Sentence Understanding through Inference". In: *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*. New Orleans, Louisiana: Association for Computational Linguistics, 2018, pp. 1112–1122. URL: http://aclweb.org/anthology/N18-1101.

[49] Y. Nie, H. Chen, and M. Bansal. "Combining Fact Extraction and Verification with Neural Semantic Matching Networks". In: *Association for the Advancement of Artificial Intelligence (AAAI)*. 2019.

[50] Y. Nie, A. Williams, E. Dinan, M. Bansal, J. Weston, and D. Kiela. "Adversarial NLI: A New Benchmark for Natural Language Understanding". In: *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. Association for Computational Linguistics, 2020.

[51] A. Parrish, W. Huang, O. Agha, S.-H. Lee, N. Nangia, A. Warstadt, K. Aggarwal, E. Allaway, T. Linzen, and S. R. Bowman. "Does putting a linguist in the loop improve NLU data collection?" In: *arXiv preprint arXiv:2104.07179* (2021).

[52] A. Liu, S. Swayamdipta, N. A. Smith, and Y. Choi. *WANLI: Worker and AI Collaboration for Natural Language Inference Dataset Creation*. Jan. 2022. URL: https://arxiv.org/pdf/2201.05955.

[53] Y. Li, S. Bubeck, R. Eldan, A. Del Giorno, S. Gunasekar, and Y. T. Lee. "Textbooks are all you need ii: phi-1.5 technical report". In: *arXiv preprint arXiv:2309.05463* (2023).

[54] L. Tunstall, E. Beeching, N. Lambert, N. Rajani, K. Rasul, Y. Belkada, S. Huang, L. von Werra, C. Fourrier, N. Habib, et al. "Zephyr: Direct distillation of lm alignment". In: *arXiv preprint arXiv:2310.16944* (2023).

[55] Deep Infra. *Deep Infra: Run the top AI models using a simple API, pay per use. Low cost, scalable and production ready infrastructure*. Online. Apr. 2024. URL: `https://deepinfra.com/`.

[56] SeMI Technologies. *Weaviate is an open source, AI-native vector database that helps developers create intuitive and reliable AI-powered applications.* Online. Apr. 2024. URL: `https://weaviate.io/`.

[57] Y. A. Malkov and D. A. Yashunin. "Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs". In: *IEEE transactions on pattern analysis and machine intelligence* 42.4 (2018), pp. 824–836.

[58] V. Sanh, L. Debut, J. Chaumond, and T. Wolf. "DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter". In: *arXiv preprint arXiv:1910.01108* (2019).

[59] T. Nguyen, M. Rosenberg, X. Song, J. Gao, S. Tiwary, R. Majumder, and L. Deng. "Ms marco: A human-generated machine reading comprehension dataset". In: (2016).

[60] K. Song, X. Tan, T. Qin, J. Lu, and T.-Y. Liu. "Mpnet: Masked and permuted pre-training for language understanding". In: *Advances in neural information processing systems* 33 (2020), pp. 16857–16867.

[61] A. Talmor and J. Berant. "MultiQA: An empirical investigation of generalization and transfer in reading comprehension". In: *arXiv preprint arXiv:1905.13453* (2019).

[62] N. Reimers and I. Gurevych. "Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks". In: *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, Nov. 2019. URL: `https://arxiv.org/abs/1908.10084`.

[63] W. Wang, F. Wei, L. Dong, H. Bao, N. Yang, and M. Zhou. "Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers". In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 5776–5788.

[64] N. Reimers and I. Gurevych. "The Curse of Dense Low-Dimensional Information Retrieval for Large Index Sizes". In: *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*. Online: Association for Computational Linguistics, Aug. 2021, pp. 605–611. URL: `https://arxiv.org/abs/2012.14210`.

[65] E. M. Voorhees. "The TREC question answering track". In: *Natural Language Engineering* 7.4 (2001), pp. 361–378.

[66] M. Honnibal and I. Montani. *spaCy 2: Natural language understanding with Bloom embeddings, convolutional neural networks and incremental parsing*. Apr. 2024. URL: https://spacy.io/.

[67] Neo4j, Inc. *Neo4j – The Leader in Graph Technology*. Online. Apr. 2024. URL: https://neo4j.com/.

[68] Neo4j, Inc. *Awesome Procedures On Cypher (APOC)*. Online. Apr. 2024. URL: https://neo4j.com/labs/apoc/.

[69] Mojan Javaheripi and Sébastien Bubeck. *Phi-2: The surprising power of small language models*. Microsoft Research Blogs. Apr. 2024. URL: https://www.microsoft.com/en-us/research/blog/phi-2-the-surprising-power-of-small-language-models/.

[70] K. Järvelin and J. Kekäläinen. "Cumulated gain-based evaluation of IR techniques". In: *ACM Transactions on Information Systems (TOIS)* 20.4 (2002), pp. 422–446.

[71] D. Carraro and D. Bridge. "Enhancing Recommendation Diversity by Re-ranking with Large Language Models". In: *arXiv preprint arXiv:2401.11506* (2024).

[72] J. Zhang, H. Zhang, D. Zhang, Y. Liu, and S. Huang. "Beam retrieval: General end-to-end retrieval for multi-hop question answering". In: *arXiv preprint arXiv:2308.08973* (2023).

[73] K. Arumae and F. Liu. "Guiding extractive summarization with question-answering rewards". In: *arXiv preprint arXiv:1904.02321* (2019).