An Empirical Analysis of Optimization Techniques for Retrieval-Augmented Generation

Sardor Asatillaev

University of Westminster

September 9, 2024

Abstract

This study examines the integration of Retrieval-Augmented Generation (RAG) with large language models, focusing on optimizing retrieval techniques to enhance response accuracy and contextual relevance. I employed the LLamaIndex framework with its in-built vector store and Neo4j for knowledge management to assess various advanced retrieval methods. These included sentence window retrieval, auto-merging retrieval, and knowledge graph-based retrieval, each compared against a conventional RAG approach. My empirical evaluation, based on metrics such as accuracy, precision, recall, contextual relevance and latency, showed that sentence window retrieval, particularly when combined with sentence reranking, generally outperformed other methods, including those leveraging knowledge graphs. The results highlight trade-offs between retrieval precision and computational efficiency, providing insights for further optimization of RAG techniques in AI-driven language models.

Acknowledgements

I would like to express my sincere gratitude to my supervisor, Habeeb Balogun, for his invaluable guidance and support throughout the course of this research project. His expertise and insightful feedback were instrumental in shaping both the direction and success of this work. His encouragement and rigorous academic standards have profoundly influenced my growth as a researcher. I am truly thankful for his mentorship.

Contents

1. Introduction

In recent years, large language models (LLMs) such as GPT-3 and BERT have transformed the field of natural language processing (Brown et al., 2020; Devlin et al., 2019). They demonstrate remarkable capabilities in tasks like text completion, translation, and question-answering. LLMs are neural networks trained on vast amounts of text data, capable of understanding and generating human language with unprecedented fluency. These models, such as GPT-3 and its successors, have shown impressive results in tasks like text completion, translation, and question-answering (Bommasani et al., 2021).

However, despite their remarkable capabilities, LLMs face several significant limitations, LLMs face notable limitations, particularly in dynamically incorporating up-to-date or domain-specific information into their responses (Lin et al., 2022). This limitation is largely due to the static nature of their pre-training, where the knowledge encoded within the model reflects the data available at the time of training and does not evolve post-training (Rogers et al., 2021). Additionally, the phenomenon of "hallucination," where LLMs generate confident but factually incorrect responses, presents significant challenges in their application to critical fields such as healthcare or legal services (Maynez et al., 2020; Bender et al., 2021). These issues are particularly problematic in applications that require high accuracy and reliability, such as medical advice, legal information, or customer support.

To address these limitations, Retrieval-Augmented Generation (RAG) has emerged as a powerful technique that enhances the performance of LLMs by incorporating external information during the text generation process (Lewis et al., 2020). In a RAG system, when a query is received, relevant information is first retrieved from a knowledge base. This retrieved information is then used to augment the input to the LLM, providing it with additional context and facts to generate a more accurate and informed response. This approach not only improves the accuracy of LLM outputs but also significantly enhances their contextual relevance, making them more suitable for handling complex, specialized, or out-of-distribution queries (Izacard & Grave, 2021).

RAG systems have shown significant potential in enhancing the accuracy and reliability of LLM-generated responses. By grounding the generation process in retrieved information, RAG helps mitigate hallucinations and improves the overall quality of outputs (Borgeaud et al., 2022). By retrieving and incorporating up-to-date and relevant data into the generation process, RAG mitigates the risk of hallucination and ensures that the responses are both accurate and contextually appropriate (Petroni et al., 2021). However, the effectiveness of RAG systems is highly dependent on the quality of the retrieval methods employed. As such, optimizing retrieval techniques has become a critical area of research in the field of natural language processing and artificial intelligence (Khattab et al., 2023).

1.1. Study Objective

This study embarks on an empirical analysis and optimization of advanced retrieval methods for Retrieval-Augmented Generation (RAG) systems. We examine three distinct techniques:

1. **Sentence Window Retrieval**: This method extends beyond individual sentences by also capturing their surrounding context, thus providing a more comprehensive input to the language model.
2. **Auto-Merging Retrieval**: This technique synthesizes multiple relevant pieces of information into a coherent context, enriching the input for the language model.
3. **Knowledge Graph-Based Retrieval**: Utilizing structured representations of knowledge, this approach retrieves information that reflects complex relationships between entities and concepts.

We compare these methods against a baseline RAG approach, which relies on basic chunking and simple vector similarity searches for information retrieval. Our research addresses several critical questions:

1. How do these advanced retrieval techniques perform compared to the baseline in terms of accuracy, precision, recall, and contextual relevancy?
2. What trade-offs exist between retrieval accuracy and computational efficiency across these techniques?

1.2. Key Technologies

To facilitate this analysis, I employ the LlamaIndex framework and the Neo4j knowledge graph database. LlamaIndex provides a platform to link Large Language Models (LLMs) with external data sources. It accommodates both high-level and low-level APIs and offers robust indexing features such as list, vector, keyword, and tree structures. It has also advanced retrieval techniques which allow creating efficient Question-Answering systems quickly and tailoring data retrieval processes to specific needs. Meanwhile, Neo4j, a leading graph database, allows managing and querying complex data relationships, particularly in constructing and querying knowledge graphs that are pivotal for retrieving contextually relevant information within our retrieval-augmented generation (RAG) systems. The detail information about these technologies will be provided in the methodology part.

1.3. Significance of this research

The significance of this research lies in its potential to substantially improve the performance of RAG systems, which have wide-ranging applications in areas such as question-answering systems, chatbots, and intelligent search engines (Gao et al., 2022). By refining retrieval

techniques, we aim to boost the accuracy, relevance, and reliability of information that LLMs provide to users across various domains.

Furthermore, my study contributes to the ongoing discussion about the balance between model performance and computational resources in AI systems (Patterson et al., 2021). As LLMs and RAG systems become increasingly prevalent in real-world applications, understanding and optimizing this trade-off becomes crucial for developing scalable and efficient solutions.

2. Literature Review

This literature review examines the current state of research in large language models (LLMs) and Retrieval-Augmented Generation (RAG), focusing on their evolution, limitations, and the various techniques proposed to optimize their performance. It highlights key advancements and challenges, providing the necessary context for the empirical analysis conducted in this study.

2.1. Evolution and Specialization of LLMs

Large Language Models have become pivotal in advancing natural language processing, enabling applications ranging from code generation and chatbot assistants to customer service automation (Chen et al., 2024; Rebedea et al., 2023; Wulf and Meierhofer, 2024). Recently, the focus has shifted towards domain-specific LLMs, which are to address the limitations of general-purpose models and tailored to perform optimally within specific fields such as law (Huang et al., 2023b; Cui et al., 2023; Yue et al., 2023), medicine (Al Nazi and Peng, 2024), and finance (Jeong, 2024). These specialized models utilize specifically designed training datasets, often referred to as copra, to deliver outputs that are more precise and relevant than those of general-purpose models, meeting the distinct needs of each field (Ouyang et al., 2022). Creating domain-specific LLMs typically involves fine-tuning general models on specialized corpora, allowing them to adapt to the nuances of specific domains while retaining broad linguistic capabilities (Ouyang et al., 2022).

2.2. Common Approaches and Limitations

Despite their tailored accuracy, both general and domain-specific LLMs face significant limitations including catastrophic forgetting (Aleixo et al., 2023; Luo et al., 2023), which affects their ability to retain previously learned information when updated with new data. Additionally, the computational cost of training and retraining LLMs is substantial, requiring extensive resources that limit their scalability and periodic updates (Chen et al., 2023; Majumder et al., 2024). In another study, Bender et al. (2021) highlights the environmental and financial costs associated with training these models, raising questions about their sustainability. Roller et al. (2021) demonstrate that these models can generate plausible sounding but incorrect information,

a phenomenon often referred to as "hallucination". Data privacy concerns, especially in sensitive fields like healthcare, further complicate their use, necessitating stringent data handling protocols (Liu et al., 2024).

2.3. Retrieval-Augmented Generation (RAG)

To address these limitations, particularly their tendency to hallucinate and their reliance on potentially outdated pre-training data, Retrieval-Augmented Generation (RAG) integrates external retrieval mechanisms into the text generation process. Lewis et al. (2020) demonstrated that coupling LLMs with a retrieval system that draws on up-to-date sources substantially improves the relevance and accuracy of generated text. This approach grounds LLM outputs in current, contextually relevant data, making RAG particularly effective for handling out-of-distribution queries - those that fall outside the model's pre-trained knowledge (Izacard & Grave, 2021). Building on this work, Borgeaud et al. (2022) demonstrated that retrieval-based language models can achieve strong performance while being more efficient to train and easier to update than standard language models.

2.4. Mechanism of RAG

Bruckhaus describes RAG systems by breaking it down into three primary components: a retriever, a generator, and a knowledge base (Bruckhaus, 2023). Figure 1 below represents this mechanism. The retriever is tasked with searching large collections of text or structured data to identify the most relevant documents or passages based on an input query. This process leverages sophisticated information retrieval techniques, including semantic search, dense vector representations, and sparse encodings, to capture the semantic meaning of queries beyond simple keyword matching (Zhang et al., 2023; Karpukhin et al., 2020). Recently, knowledge graph-based retrievers have gained attention for their ability to capture complex relationships and structured information more effectively (Delile et al., 2024; Ji et al., 2022).

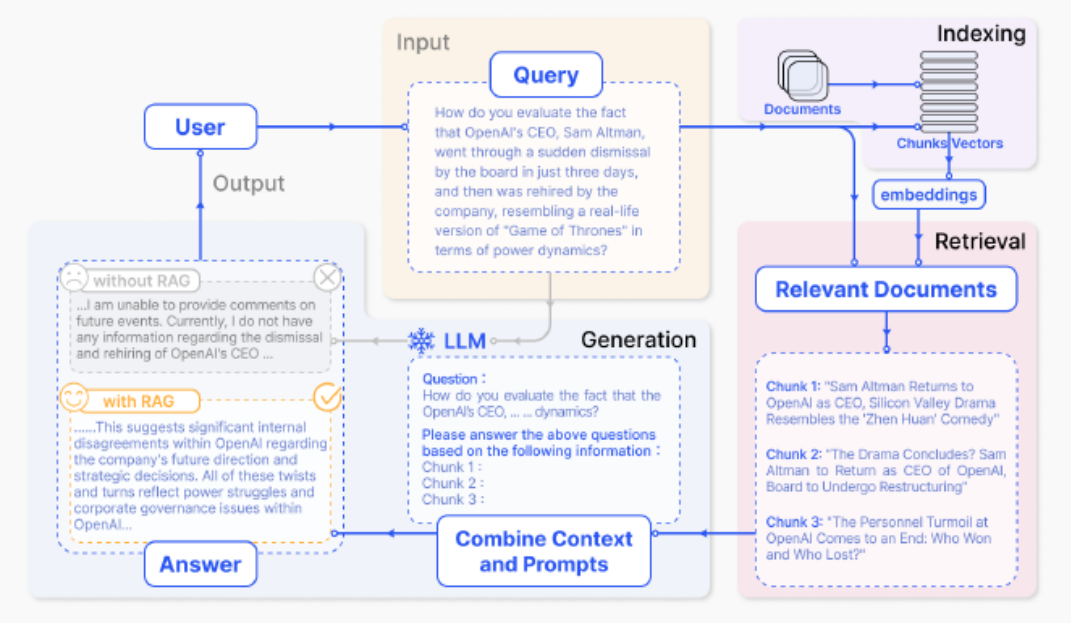

Figure 1: An example of the RAG process (Gao et al. 2024)

Once the relevant documents or data points are retrieved, they are passed, along with the original query, to the generator - a large pre-trained language model such as GPT-3.5 or GPT-4. The generator conditions its output on both the query and the retrieved knowledge, utilizing techniques such as attention, copying, and content selection to effectively integrate the retrieved information into the generated response (Lewis et al., 2020). This allows the RAG system to produce outputs that are not only accurate but also contextually appropriate and consistent with the retrieved knowledge.

The knowledge base, the third component of a RAG system, serves as the repository of information from which the retriever sources relevant documents. This knowledge base can include a wide range of data, from unstructured text sources like web pages and articles to structured data such as databases and proprietary enterprise information (Guu et al., 2020; Khandelwal et al., 2020). The effectiveness of the RAG system is highly dependent on the quality, coverage, and freshness of the knowledge base, as these factors directly influence the accuracy and relevance of the retrieval process.

2.5. Design of RAG development flow

There are five key stages in the development of RAG model (Llama Index, 2024):

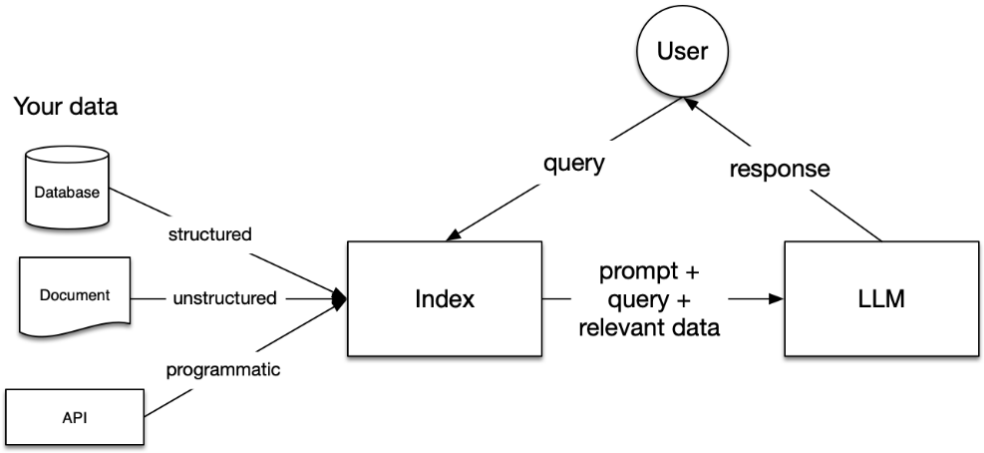

Figure 2: High level concepts of RAG (Figure 2 from Llama Index, 2024)

1. **Loading**: Unstructured and structured data from its original source, which could be text files, PDFs, websites, databases, or APIs are imported.
2. **Indexing:** This step focuses on creating a data structure that facilitates data querying. This typically involves generating vector embeddings - numerical representations that capture the meaning of the data
3. **Storing:** After indexing the data, it's essential to store both the index and any additional metadata. This prevents the need for re-indexing in the future.
4. **Querying**: Depending on the indexing strategy employed, there are numerous methods to utilize LLMs and data structures for querying. These include sub-queries, multi-step queries, and hybrid approaches.
5. **Evaluation**: In the final stage the system will be tested and assessed for effectiveness. There are different objective metrics to gauge the accuracy, fidelity, and speed of responses to queries.

Figure 3: RAG workflow process (Figure 3 from Llama Index, 2024).

2.6. Base RAG

The base retrieval system serves as the foundational component of the Retrieval-Augmented Generation (RAG) framework. According to Gao, the base retrieval method follows a traditional process that includes indexing, retrieval, and generation and is often used for conducting experiments and building prototypes as it is easy to construct and evaluate the model withing limited time (Gao et al., 2023). It is implemented using default algorithms such as simple

chunking methods, naïve indexing and vector store libraries. The pipeline, illustrated in Figure 4, offers a straightforward process for retrieving relevant information and generating responses.

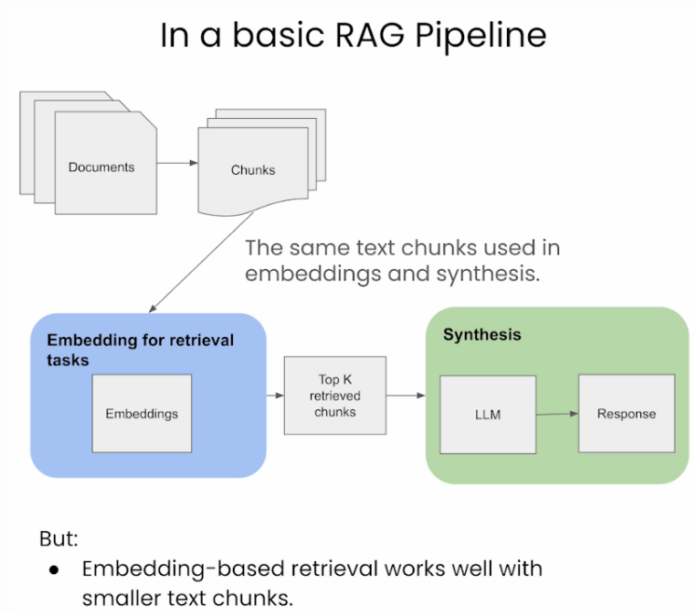

Figure 4: Naïve RAG workflow (Aman.ai 2024; DeepLearning.AI 2024).

2.6.1. How the Naive RAG Works:

Document Chunking: The process begins by breaking down documents into smaller, coherent text segments. This method divides the text into manageable segments that are contextually complete, typically ranging from a few sentences to a paragraph.

Embedding for Retrieval: Once the documents are chunked, each text segment is converted into a vector embedding. Vector embeddings transform words, sentences, and other text data into numerical forms that represent their meanings and relationships (Elastic, n.d.). This can be achieved using open source embedding models or with more sophisticated models like achieved using OpenAI's "text-embedding-ada-002" model, which generates 1536-dimensional vector representations of the text based on its semantic content. These embeddings capture the meaning of the text in a high-dimensional space, allowing for effective comparison and retrieval based on semantic similarity (Aman.ai 2024; DeepLearning.AI 2024).

Indexing and Retrieval: The VectorStoreIndex database facilitated the indexing of these vector embeddings, making them easily searchable during the retrieval process. When a query is input into the system, the base retrieval method identifies the top-k most relevant text chunks by comparing the query's embedding with those stored in the database. The system ranks the chunks based on their semantic similarity to the query, enabling the retrieval of the most relevant information (Aman.ai 2024).

Synthesis: The retrieved chunks are then passed to the Language Model (e.g., GPT-3.5-turbo), which synthesizes the information to generate a response. The LLM integrates the retrieved text

with the query to produce a coherent and contextually relevant output (Aman.ai 2024)). This synthesis process ensures that the response is not only based on relevant information but is also logically structured and easy to understand.

2.6.2. Significance as a Foundation

The base retrieval system is crucial as it provides the underlying structure for more advanced retrieval techniques, such as sentence-window retrieval and auto-merging retrieval. By establishing a robust method for chunking, embedding, and retrieving text, the base retrieval approach serves as a reliable benchmark. It allows for direct comparisons with more sophisticated methods, highlighting the strengths and limitations of each approach. The simplicity and efficiency of the base retrieval system make it a versatile tool for a wide range of applications, providing a solid foundation for exploring and evaluating more complex RAG techniques.

2.6.3. Advantages and Limitations

The base RAG approach is simple and computationally efficient, making it suitable for a wide range of applications. Its use of chunked text ensures that retrieved information aligns closely with the query. However, it has limitations, such as potential context loss when splitting documents, which can lead to less coherent responses. Additionally, it may struggle with complex queries that require synthesizing information from multiple sections and risks retrieving redundant chunks, which can affect response quality.

These limitations underscore the need for more sophisticated retrieval techniques, which are explored further in this study.

2.7. Advanced Retrieval Techniques

The effectiveness of RAG systems heavily depends on the quality and relevance of the retrieved information. Traditional methods, such as keyword-based retrieval, have been used extensively but often fall short in terms of contextual relevance and precision (Karpukhin et al., 2020). To address these limitations, several advanced retrieval techniques have been proposed:

2.7.1. Sentence Window Retrieval

The basic principle behind sentence window retrieval is that each sentence of a document is treated as a central node, surrounded by its neighboring sentences to form a "window" (Llama Index, 2024). This approach allows each sentence and its immediate context to be indexed and retrieved as a single unit (Llama Index, 2024; Yang, 2023). Guu et al. (2020) demonstrated the importance of context in their REALM model, which retrieves and attends to entire Wikipedia

articles. Similarly, Karpukhin et al. (2020) showed that retrieving passages rather than single sentences improved performance in open-domain question answering tasks. It is particularly useful for extracting detailed and contextually relevant information from large documents or databases, enhancing the precision of search results and the relevance of retrieved content (Llama Index, 2024). Sentence Window parser is often used with Sentence Transformers. Reimers and Gurevych (2019) employed them in their study and noticed that the relevance of the retrieved content is significantly improved, particularly in complex queries requiring detailed and specific information.

Figure 5 below describes sentence window retrieval pipeline.

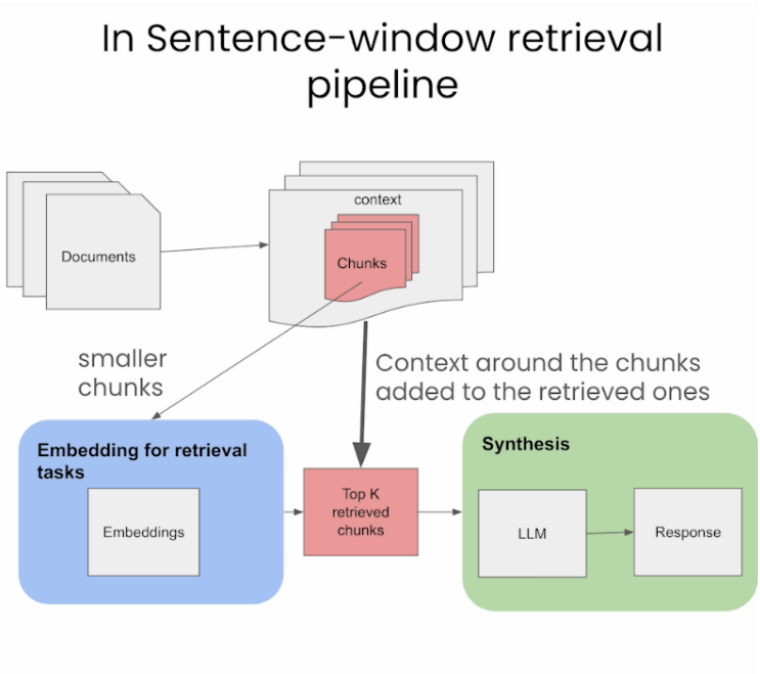

Figure 5: Sentence Window Retrieval Pipeline

2.7.1.1. How Sentence Window Retrieval Works:

Document Chunking with Sentence Windows: Similar to the base retrieval approach, the sentence window retrieval method begins with the segmentation of documents into manageable chunks. However, instead of general text segments, this method specifically uses windows of sentences. In generation, it incorporates additional sentences around the initial one, providing the LLM with extended context for producing richer, more detailed outputs. This ensures that each sentence window not only contains relevant information but also preserves the surrounding context that may be crucial for fully understanding the content (Aman.ai 2024).

Embedding and Indexing: Once the documents are divided into sentence windows, each window is converted into a vector embedding in a similar way to the base retriever (Aman.ai 2024).

Retrieval Process: The retrieval phase in sentence window retrieval involves searching for the most semantically similar sentence windows based on the input query. The system compares the

query's embedding with the stored embeddings in the vector store, identifying and ranking the top-k most relevant windows. By retrieving slightly larger chunks than those used in the base retrieval method, this approach captures more context around the relevant content, which can significantly improve the precision and relevance of the results as shown in the figure below (Aman.ai 2024; DeepLearning.AI 2024).

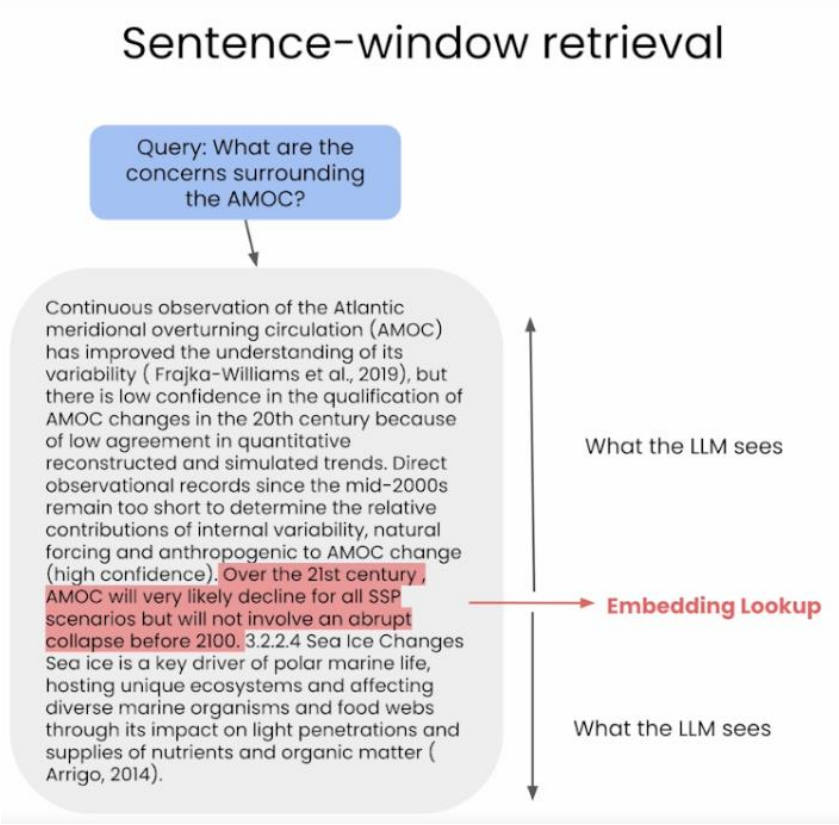

Figure 6: An Instance of Sentence Window Retrieval (Aman.ai 2024).

Context Reintroduction for Synthesis: One of the key features of the sentence window retrieval method is the reintroduction of surrounding context during the synthesis phase. After retrieving the most relevant sentence windows, the surrounding context (the sentences immediately before and after the retrieved window) is added back to the content before it is passed to the Language Model (e.g., GPT-3.5-turbo) for response generation. This step ensures that the LLM has a broader understanding of the text, allowing it to generate responses that are both contextually rich and precise (Aman.ai 2024).

2.7.1.2. Advantages in Terms of Accuracy and Precision:

Enhanced Precision: By segmenting the text into smaller, contextually coherent windows, the sentence window retrieval method significantly improves the precision of the retrieval process. This allows the system to focus on the most relevant segments of text, leading to more accurate and targeted responses.

Contextual Integrity: The reintroduction of surrounding context during synthesis helps maintain the integrity of the information flow, ensuring that the final output is not only accurate but also contextually appropriate. This balance between focused retrieval and contextual richness enhances the overall quality of the generated responses.

Improved Query Handling: The method is particularly effective in handling complex queries that require detailed and specific information, as it ensures that the relevant content is retrieved and presented with sufficient context to support comprehensive understanding.

2.7.2. Auto-Merging Retrieval

Another promising approach is auto-merging retrieval, where results from different retrieval strategies are combined to enhance overall accuracy. In auto-merging retrieval, documents are organized in a hierarchical tree structure, which allows for an efficient merging of information from various parts of the document. The process works by initially retrieving a child node within the tree. After retrieving this node, the system then fetches the parent node, which includes the combined content of its child nodes. This method helps to overcome the challenge of missing contextual information, as each parent node inherently aggregates and reflects the content of its children, providing a more comprehensive view of the document's overall context and details. This approach is particularly beneficial in situations where understanding the broader context and connections between various document segments is crucial (Liu, LlamaIndex 2022).

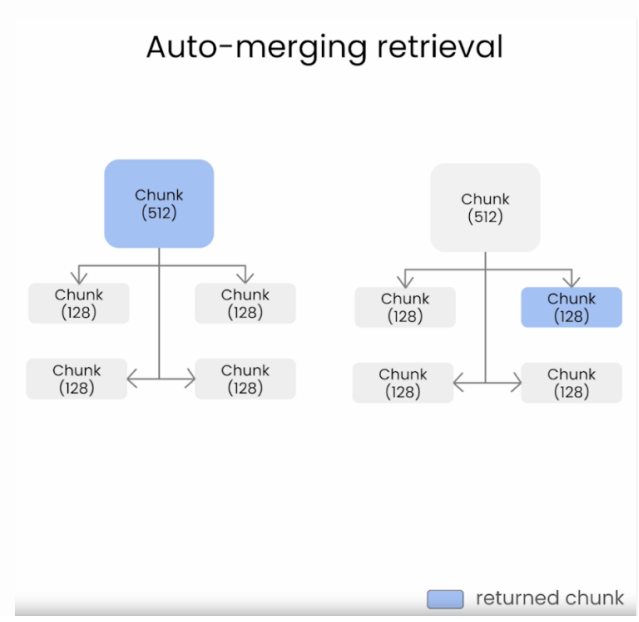

Figure 7: Auto-merging retrieval (Aman.ai 2024).

2.7.2.1. Hierarchical Chunking and Embedding

The auto-merging retrieval process begins with a hierarchical chunking of documents. Text is divided into multiple levels of granularity using a HierarchicalNodeParser, which segments the document into larger parent chunks (512) and smaller child chunks (128). This hierarchical structure allows the system to maintain contextual relationships across different levels of the document. Each chunk, whether large or small, is then converted into a vector embedding.

2.7.2.2. Dynamic Merging Process

The core of the auto-merging retrieval method lies in its ability to dynamically combine related chunks. Once the hierarchical structure is established, the system evaluates the similarity between smaller child chunks and their parent chunks. If the similarity between a set of child chunks and a parent chunk exceeds a predefined threshold (e.g., cosine similarity), these smaller chunks are merged into the larger parent chunk.

This merging process is dynamic, meaning that it adapts based on the specific query and the relationships between the chunks. By merging smaller, contextually related segments into larger, more informative chunks, the system creates a richer and more comprehensive set of information that is better suited to answering complex queries.

2.7.2.3. Advantages of Auto-Merging Retrieval

Comprehensive Contextual Responses: By merging information from multiple related segments, auto-merging retrieval produces responses that are more comprehensive and contextually relevant, addressing queries that require a deep understanding of the content.

Reduced Fragmentation: This approach significantly reduces the issue of fragmented information retrieval, which is common in naive retrieval methods, especially when dealing with smaller chunks.

Dynamic Content Integration: Auto-merging dynamically combines smaller chunks into larger, more informative ones, enriching the information provided to the LLM and improving the overall quality of the generated responses.

2.7.3. Knowledge Graph-Based Retrieval

Knowledge graphs provide structured representations of information, potentially capturing complex relationships between entities and concepts. They organize information in a structured way into entities and relationships. This allows the retrieval process to consider not just the content of documents, but also the interconnectedness of the information (Ji et al., 2022; Pandit, 2024). This method is particularly effective in domains where it is crucial to understand relationships such as individuals, organizations, events, or medical conditions (Ji et al., 2022). Zhang et al. (2022) proposed a knowledge graph-augmented language model that outperformed traditional LLMs in knowledge-intensive tasks. Similarly, Yasunaga et al. (2021) demonstrated the effectiveness of retrieving from knowledge graphs for question answering tasks. Knowledge graphs enable more contextually aware searches by utilizing graph traversal algorithms and

semantic embeddings to extract and rank information based on both direct and inferred connections between entities (Hogan et al., 2021). In RAG systems, knowledge graph-based retrieval significantly enhances the ability to provide precise and contextually relevant answers, especially in fields requiring complex reasoning and relationship understanding, such as legal or financial analysis (Chen et al., 2024).

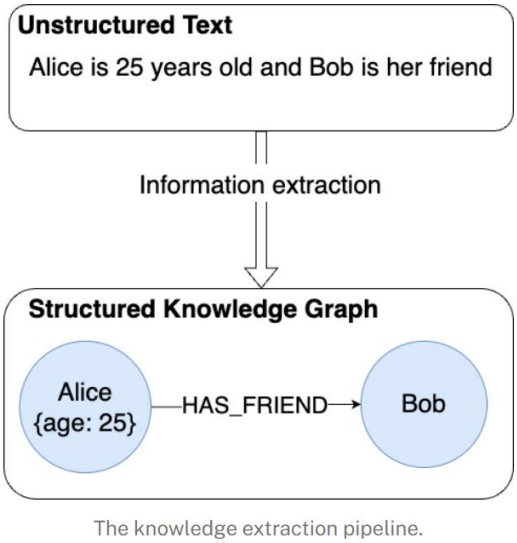

Figure 8: Knowledge extraction pipeline (Mayerhofer, 2024).

2.7.3.1. Building the Knowledge Graph

The constructing a knowledge graph involves several key steps:

- **Data Ingestion and Triplet Extraction:** Relevant data from documents is first ingested into the system. With the help of indexing methods triplets from the text is extracted. A triplet typically consists of two entities and the relationship between them (e.g., [Entity 1]—[Relationship]—[Entity 2]). Number of triplets is hyper parameter and is usually set or chosen default value. For instance, if the parameter maximum triplets is set to 2, meaning a maximum of two triplets is extracted from each chunk of text. This controlled approach ensured that only the most relevant relationships are captured, preventing the graph from becoming overly dense and difficult to query (Narayana, 2024).
- **Embedding and Indexing:** Each triplet, along with the associated text chunk, is converted into a vector embedding using embedding model. The generated embeddings is then stored in Neo4j, facilitating efficient similarity searches during retrieval operations. The embeddings capture both the semantic content of the entities and their relationships, enabling precise and contextually relevant retrieval. (Rojo-Echeburúa, 2024; Narayana, 2024).
- **Graph Construction:** The extracted triplets are organized into a graph structure within Neo4j, where nodes represent entities and edges represent the relationships between them. This structure allows the system to understand and utilize the complex

relationships between different concepts, authors, institutions, and topics within the corpus (Rojo-Echeburúa, 2024).

2.7.3.2. Retrieval Process

During the retrieval phase, the system leverages the knowledge graph to find relevant entities and their associated relationships based on the input query. The query triggers a search within the Neo4j database, identifying nodes that semantically match the query and exploring their relationships within the graph. This allows the retrieval of a network of related information that collectively addresses the query (Rojo-Echeburúa, 2024).

For example, if a query involves a specific AI technique, the system might retrieve not only the direct information about the technique, but also related concepts, key researchers, and institutions associated with its development. The inclusion of embeddings in the index ensures that the retrieval is both semantically accurate and contextually comprehensive (Rojo-Echeburúa, 2024).

2.7.3.3. Advantages of Knowledge Graph-Based Retrieval

- **Structured Information Retrieval:** The use of a knowledge graph allows for highly structured and contextually rich retrieval, particularly useful for complex queries that require understanding relationships between different data points (Narayana, 2024).
- **Contextual Richness:** By retrieving related entities and their relationships, the knowledge graph-based retrieval method provides a more comprehensive dataset for the LLM to generate responses, enhancing the quality and relevance of the output (Narayana, 2024).
- **Efficient Querying:** The use of Neo4j as the vector store, combined with the LlamaIndex's indexing capabilities, enables efficient retrieval of complex relationships, making this method particularly effective for queries that require deep contextual understanding (Narayana, 2024).

2.8. Evaluation Metrics

Evaluating the performance of RAG systems and their retrieval components requires appropriate metrics. Traditional information retrieval metrics like precision, recall, and F1-score remain relevant (Manning et al., 2008). However, for RAG systems, additional metrics are often employed. Karpukhin et al. (2020) used top-k accuracy to evaluate retrieval performance, while Lewis et al. (2020) employed perplexity and ROUGE scores to assess the quality of generated text.

Context relevancy, a metric that measures how well the retrieved information aligns with the query, has gained importance in RAG evaluation. Khattab et al. (2023) proposed novel

evaluation frameworks that consider both retrieval accuracy and the quality of the final generated output.

2.9. Challenges and Future Directions

Despite the promising results of RAG systems, several challenges remain. A key issue is balancing retrieval accuracy with computational efficiency. Techniques like dense passage retrieval (DPR) (Karpukhin et al., 2020) improve accuracy but are computationally expensive, making real-time application difficult, particularly with large-scale datasets. Efficiency is also a major concern, as noted by Izacard and Grave (2021), who proposed methods to scale RAG to large knowledge bases. Another challenge is maintaining consistency between retrieved information and generated text, an issue explored by Rashkin et al. (2021).

Handling noisy or contradictory information within the knowledge base presents further challenges. If outdated or incorrect data is retrieved, it can lead to misleading outputs, which is particularly problematic in high-stakes domains like legal or medical information (Hogan et al., 2021).

Lastly, scaling RAG systems for real-world applications, especially in resource-constrained environments, remains a significant hurdle. As the knowledge base grows, maintaining retrieval speed and accuracy becomes increasingly difficult (Gao et al., 2023). Future research must focus on optimizing these systems for broader deployment while minimizing computational demands.

The integration of knowledge graphs with neural retrieval methods presents an exciting direction for future research. Xu et al. (2022) demonstrated promising results in combining knowledge graph embeddings with text embeddings for improved retrieval.

As RAG systems continue to evolve, there is a growing need for standardized benchmarks and evaluation metrics that can adequately capture the nuances of retrieval-augmented generation. Developing such benchmarks and metrics remains an active area of research in the field.

3. Methodology

This study employs a systematic approach to evaluate and compare different retrieval techniques for Retrieval-Augmented Generation (RAG) systems. Our methodology encompasses the implementation of various retrieval methods, the setup of a consistent experimental environment, and the definition of evaluation metrics.

3.1. Experimental Procedure

Our experimental procedure consisted of the following steps:

1. Set up model settings and installing tools and libraries: This includes choosing language model for model response, embedding model for converting data into vector embeddings, chunking method to chunk documents. Furthermore, in this step I prepared my developing environment by installing Neo4j Desktop graph database for knowledge graph retrieval, creating virtual environment for development and installing Llamaindex framework and its libraries.
2. Document Loading: The research papers were loaded using the PyMuPDFReader library. A custom class was developed to facilitate structured loading of the data.
3. Data Preprocessing: A series of preprocessing steps were implemented to clean and refine the dataset.
4. Retrieval Implementation: I implemented each retrieval technique using LlamaIndex, integrating VectorStoreIndex for storing data in vector store. I also implemented Neo4j for knowledge graph retrieval.
5. Evaluation: I evaluated each model on a held-out test set, measuring performance across various metrics.

3.2. Experimental Setup

3.2.1. Dataset

The dataset used in this study is derived from the AI ArXiv collection, as described in the research paper "ARAGOG: Advanced RAG Output Grading" by Eibich et al (2024). It contains 10 research papers focused on Artificial Intelligence (AI) and Large Language Models (LLMs). These papers provide a rich and varied collection that reflects the latest developments and diverse research themes within these fields. It is an excellent resource for constructing databases to evaluate the effectiveness of Retrieval-Augmented Generation (RAG) methods, as it encapsulates wide range of discussions, findings, scientific formulas and noises such as references and appendixes, reflecting real-world scenarios that are crucial for our study's objectives. (Eibich, Nagpal, and Fred-Ojala, 2024, p. 5).

3.2.2. Language Model for Response Generation

For all retriever methods, the OpenAI's GPT-4o mini model was employed for response generation (OpenAI, 2024). This model is the smallest and most cost-effective option offered by OpenAI. Employing the same model across all retrieval methods ensures consistent performance of the language model, allowing for a fair comparison of the effectiveness of each retrieval technique. I selected the GPT-4o mini model primarily for its cost-effectiveness, which facilitates extensive experimental runs without excessive expenditure. Despite its smaller size, it retains the advanced capabilities of the GPT-4 architecture, ensuring it can handle complex language processing tasks efficiently. This choice allows us to maintain high computational performance standards while effectively managing resource utilization during our experiments.

3.2.3. Embedding Model

An embedding model is crucial for converting text and queries into vector embeddings, enabling the comparison of a query against stored indexed text in a database. This process facilitates the retrieval of relevant information based on the query, which is then synthesized with a language model to produce an answer. For my study, I utilized OpenAI's text-embedding-3-small model. I chose this model because it meets high computational performance standards and ensures task efficiency. The length of the embedding vector is 1536, which is optimal for handling diverse data efficiently. Its performance, as per the Massive Text Embedding Benchmark (MTEB) evaluation, is rated at 62.3%. This indicates a strong capability to generate meaningful and contextually relevant embeddings for our experiments (OpenAI Embeddings, n.d.).

3.2.4. Loading data

The research papers used in this study were loaded using the PyMuPDFReader library. It is Python tool designed for efficient reading and processing of PDF documents. To facilitate the structured loading of the data, a custom class was developed. This class was responsible for reading the papers from the project directory, organizing them into a dictionary structure. Each entry in the dictionary included both the full text of the paper and associated metadata, such as the title, authors, and abstract. This structured approach ensured that the data was readily accessible for subsequent processing and retrieval operations.

Each research paper in the dataset is treated as a distinct document within the database. The papers are formatted as plain text, allowing for straightforward indexing and retrieval during the implementation of various RAG techniques. The dataset includes not only the main body of text from each paper but also metadata such as titles, authors, and abstracts, which can be leveraged to enhance retrieval accuracy.

3.2.5. Data preprocessing

Data preprocessing is crucial to the success of Retrieval-Augmented Generation (RAG) models because it ensures the cleanliness and relevance of the data used for indexing and retrieval. By removing irrelevant sections like references and cleaning the text of problematic characters, the data is refined to better serve the retrieval process. This not only reduced the memory footprint of the database but also significantly enhanced the precision and relevance of the retrieved content.

For this study, I implemented several data cleaning tasks:

Removing References: One of the key preprocessing functions developed was designed to identify and remove the references section from each research paper. References, while valuable in the context of academic reading, often introduce noise when used in retrieval systems, as they do not contribute directly to the content being queried. Additionally, references can consume significant memory space in the database, potentially slowing down retrieval processes. By removing these sections, the data was made more relevant and streamlined, which in turn improved the accuracy and efficiency of the RAG outputs.

Replacing or Removing Problematic Characters: Another essential preprocessing step involved the development of a function to replace or remove problematic characters within the text. PDF documents often contain characters or formatting artifacts that can cause issues during text processing and analysis. These characters might include non-standard punctuation, encoding errors, or extraneous symbols. The function was designed to clean these elements from the text, ensuring that the data was in a consistent and usable format. This step was critical for preventing errors during indexing and retrieval and for enhancing the overall quality of the data fed into the RAG model.

3.2.6. Implementation of Base Retriever

I implemented base retrieval using traditional methods with default parameters and configurations. The process begins by dividing documents into smaller. In my study I used the SentenceSplitter() method, which prioritizes keeping sentences and paragraphs intact within each chunk. Unlike standard text splitting techniques, SentenceSplitter() ensures that chunks do not split mid-sentence or create fragmented text, preserving the natural flow of information. For vector store database I used LlamaIndex's default vector store – VectorStoreIndex. It is simple in-memory vector store, yet effective for similarity search and conduct experiments. Once the documents were chunked, each text segment was converted into a vector embedding. The embeddings then were stored in a VectorStoreIndex database and saved locally with vector_store.persist() method.

3.2.7. Implementation of Sentence-Window Retrieval

There are Document and Node objects in LlamaIndex. A Document acts as a versatile container for various data sources, such as a PDF, API output, or data retrieved from a database. Typically, a Document holds text or an image and additional attributes, including metadata. The metadata is a dictionary that contains key information about the document like its name, creation and update dates, authors, and more.

It is also important to understand the node concept. A node is a segment of a source Document, specifically a text chunk in our scenario. Nodes also store metadata and information about their relationships with other nodes (Llama Index Document and Nodes, n.d.). For building sentence window retrieval, I utilized the SentenceWindowNodeParser class, which is designed to break documents into nodes, each containing a single sentence. Furthermore, each node includes a "window" of sentences surrounding the main sentence. In my research, I set the window size to 6, meaning each node's window includes six sentences before and after the central sentence. I also implemented the MetadataReplacementNodePostProcessor class in sentence window retrieval. This tool is beneficial for large documents or indexes as it allows for the retrieval of more detailed information by replacing single sentences with a window of surrounding sentences before they are processed by the LLM (Llama Index Sentence Window, n.d.).

```
from llama_index.core.postprocessor import MetadataReplacementPostProcessor, SentenceTransformerRerank
# define postprocessors
post_proc = MetadataReplacementPostProcessor(target_metadata_key="window")
rerank = SentenceTransformerRerank(
    top_n=2, model="BAAI/bge-reranker-base"
)

sentence_query_engine_rerank = sentence_index.as_query_engine(
    text_qa_template=prompt_template, similarity_top_k=6, embed_model=embed_model,
    llm=llm, node_postprocessors=[post_proc, rerank]
)
sen_window_response = sentence_query_engine_rerank.query("What are the two main tasks BERT is pre-trained on?")
display(Markdown(f"<b>{sen_window_response}</b>"))
print("Sentence Window Retriever + Sentence Rerank Response: ", sen_window_response)
```

BERT is pre-trained on masked language modeling and next sentence prediction tasks.

```
Sentence Window Retriever + Sentence Rerank Response:  BERT is pre-trained on masked language modeling and next sentence prediction tasks.
```

I also utilized the SentenceTransformerRerank class for sentence window retrieval. Reranking can enhance the speed of a language model (LLM) query while maintaining, or even improving, accuracy. It achieves this by eliminating irrelevant nodes from the context, effectively streamlining the process and focusing the model's attention on the most pertinent information. Combining power of Sentence Reranking techniques with SentenceWindowNodeParser can show even better results than single use of sentence window node parser. It also boost scale of the computational resource and improves time constrains without losing accuracy.

```
window_response = sentence_query_engine_rerank.query(question)
print("Window Response: ")
display(Markdown(f"<b>{window_response}</b>"))
```

```
INFO:httpx:HTTP Request: POST https://api.openai.com/v1/embeddings "HTTP/1.1 200 OK"
Batches: 100% 1/1 [00:25<00:00, 25.55s/it]
INFO:httpx:HTTP Request: POST https://api.openai.com/v1/chat/completions "HTTP/1.1 200 OK"
Window Response:
```

BERT is pre-trained on masked language modeling and next sentence prediction tasks.

```
window = window_response.source_nodes[0].node.metadata["window"]
sentence = window_response.source_nodes[0].node.metadata["original_text"]

print(f"Window: {window}")
```

```
Window: BERT: Pre-training of Deep Bidirectional Transformers for
Language Understanding
Jacob Devlin
Ming-Wei Chang
Kenton Lee
Kristina Toutanova
Google AI Language
{jacobdevlin,mingweichang,kentonl,kristout}@google.com
Abstract
We introduce a new language representa-
tion model called BERT, which stands for
Bidirectional Encoder Representations from
Transformers.  Unlike recent language repre-
sentation models (Peters et al., 2018a; Rad-
ford et al., 2018), BERT is designed to pre-
train deep bidirectional representations from
unlabeled text by jointly conditioning on both
left and right context in all layers.  As a re-
sult, the pre-trained BERT model can be fine-
tuned with just one additional output layer
to create state-of-the-art models for a wide

range of tasks, such as question answering and
language inference, without substantial task-
specific architecture modifications.
 BERT is conceptually simple and empirically
powerful.
 It obtains new state-of-the-art re-
sults on eleven natural language processing
tasks, including pushing the GLUE score to
80.5% (7.7% point absolute improvement),
MultiNLI accuracy to 86.7% (4.6% absolute
improvement), SQuAD v1.1 question answer-
ing Test F1 to 93.2 (1.5 point absolute im-
provement) and SQuAD v2.0 Test F1 to 83.1
(5.1 point absolute improvement).
 1
Introduction
Language model pre-training has been shown to
be effective for improving many natural language
processing tasks (Dai and Le, 2015; Peters et al.,
2018a; Radford et al., 2018; Howard and Ruder,
2018).  These include sentence-level tasks such as
natural language inference (Bowman et al., 2015;
Williams et al., 2018) and paraphrasing (Dolan
and Brockett, 2005), which aim to predict the re-
lationships between sentences by analyzing them
holistically, as well as token-level tasks such as
```

```
named entity recognition and question answering,
where models are required to produce fine-grained
output at the token level (Tjong Kim Sang and
De Meulder, 2003; Rajpurkar et al., 2016).
 There are two existing strategies for apply-
ing pre-trained language representations to down-
stream tasks: feature-based and fine-tuning.  The
feature-based approach, such as ELMo (Peters
et al., 2018a), uses task-specific architectures that
include the pre-trained representations as addi-
tional features.  The fine-tuning approach, such as
the Generative Pre-trained Transformer (OpenAI
GPT) (Radford et al., 2018), introduces minimal
task-specific parameters, and is trained on the
downstream tasks by simply fine-tuning all pre-
trained parameters.  The two approaches share the
same objective function during pre-training, where
they use unidirectional language models to learn
general language representations.
 We argue that current techniques restrict the
power of the pre-trained representations, espe-
cially for the fine-tuning approaches.
 The ma-
jor limitation is that standard language models are
unidirectional, and this limits the choice of archi-
tectures that can be used during pre-training.
```

```
print(f"Original Sentence: {sentence}")
```

```
Original Sentence: These include sentence-level tasks such as
natural language inference (Bowman et al., 2015;
Williams et al., 2018) and paraphrasing (Dolan
and Brockett, 2005), which aim to predict the re-
lationships between sentences by analyzing them
holistically, as well as token-level tasks such as
named entity recognition and question answering,
where models are required to produce fine-grained
output at the token level (Tjong Kim Sang and
De Meulder, 2003; Rajpurkar et al., 2016).
```

The sentence window retrieval method represents a significant advancement over the base retrieval approach by providing more precise and contextually relevant results.

3.2.8. Implementation of Auto-Merging Retriever

Another advanced retrieval method I explored in this study is the AutoMergingRetriever. In this technique, a set of leaf nodes is examined and then recursively "merges" subsets of these nodes that reference a common parent node based on established threshold. As a result, smaller chunks are merged into a broader context. This improves the synthesis process by creating a more comprehensive informational base from which the language model can generate responses (Llama Index Auto-merging retrieval, n.d.).

For parsing texts, I utilized LlamaIndex's HierarchicalNodeParser class which inputs all documents and returns an entire set of documents and segments the document into larger parent chunks and smaller child chunks**.** In my study documents are parsed into chunks of 1024, 512, and 256 tokens, with each smaller chunk linked to a corresponding parent chunk: Parent chunks: 1024 tokens; Intermediate chunks: 512 tokens; Child chunks: 256 tokens. This hierarchical

structure allows the system to maintain contextual relationships across different levels of the document.

3.2.9. Implementation of Knowledge Graph-Based Retrieval

For the knowledge graph-based retrieval, I utilized Neo4j, a graph database management system renowned for its efficient handling of highly interconnected data. The implementation involved constructing a knowledge graph from a set of documents, which were indexed using the KnowledgeGraphIndex.from_documents function. This method allowed for the systematic organization of text into structured data, forming nodes and relationships based on the content of the documents.

The code snippet below outlines the initialization of the knowledge graph index:

```
def build_knowledge_graph(documents, save_dir="kg_index"):
    # Neo4j Graph Store Setup
    graph_store = Neo4jGraphStore(username="neo4j",
                                  password="Ss123456$",
                                  url="bolt://localhost:7687",  # "bolt://7
                                  database="neo4j")

    storage_context = StorageContext.from_defaults(graph_store=graph_store)

    kg_index = KnowledgeGraphIndex.from_documents(
        documents,
        storage_context=storage_context,
        max_triplets_per_chunk=10,
        include_embeddings=True,
        show_progress=True
    )
    # save and load
    kg_index.storage_context.persist(persist_dir=save_dir)
    print("KNOWLEDGE GRAPH INDEX SAVED!!!")
    return kg_index
```

Figure 9: source code of building knowledge graph

Key parameters were set to optimize the indexing process:

- **documents**: A collection of texts that were converted into a graph format.

- **storage_context**: Defines the environment where the data is stored, ensuring efficient retrieval.
- **max_triplets_per_chunk**: Limits the number of triplets processed per chunk to 10, balancing detail with processing speed.
- **include_embeddings**: Embeddings were included to enhance the semantic understanding of the nodes.
- **show_progress**: Enabled to monitor the progress of the indexing operation, which is crucial for debugging and optimizing the setup.

The most challenging and crucial aspect of constructing a knowledge graph database is creating the graph schema, which involves extracting nodes and relationships from extensive text sources. Various methods exist for schema creation; however, due to time constraints in this study, I adopted the simplest approach by submitting the input data directly to the LLM (Language Learning Model) and allowing it to identify which nodes and relationships to extract. I instructed the LLM to output the extracted entities in a specified format, which includes a name, a type, and properties. This streamlined process facilitates the extraction of nodes and edges directly from the input text, making it efficient and manageable within the given constraints (Mayerhofer, 2024).

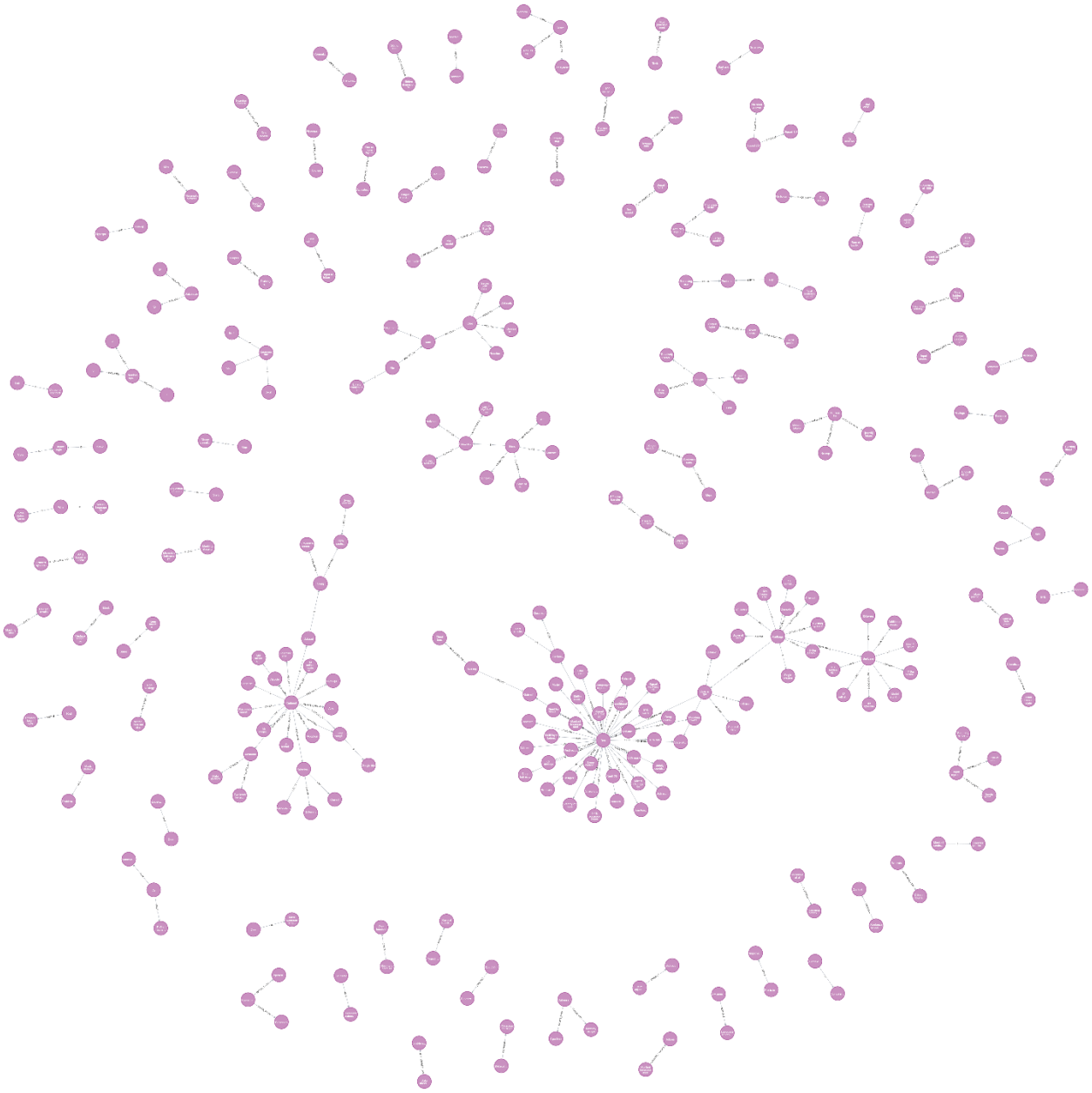

Figure 10: knowledge graph schema

The resulting graph, depicted in the provided screenshot, illustrates the nodes and their interconnections.

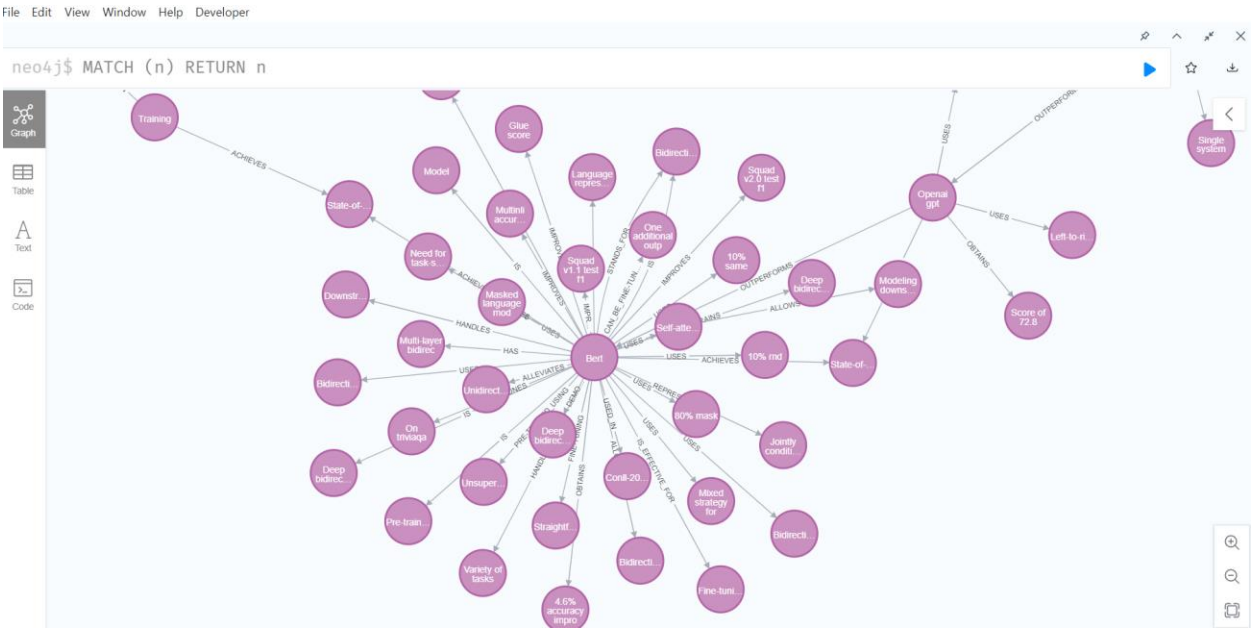

Figure 11: Knowledge graph focusing on central node – "Bert"

The second screenshot above is the result of zooming out the graph. It shows the central node, labeled 'Bert', represents a significant entity within the graph, showcasing its connectivity and the potential influence on retrieval outputs. This visualization aids in understanding the structural and relational aspects of the data as processed by the knowledge graph. "Bert" entity is connected directly with other nodes that describe its attributes (e.g., "Bidirectional", "Deep bidirectional", "Masked language mod") and capabilities (e.g., "Handles", "Uses", "Achieves").

Here are some examples of triplets derived from the visualization:

1. **[Training]—[ACHIEVES]—[State-of-art]**
    - This triplet suggests that the entity "Training" achieves the state-of-the-art, implying a relationship where specific training methodologies or datasets lead to top performance metrics in the model domain.
2. **[BERT]—[USES]—[Self-attention]**
    - Here, "BERT" is linked to "Self-attention" through the "USES" relationship
3. **[BERT]—[ALLOWS]—[10% md]**
    - This triplet implies that using BERT is linked to "10% md" through "ALLOWS" relationship.
4. **[Openai gpt]—[OUTPERFORMS]—[Modeling downstream]**
    - In this triplet, the "Openai gpt" model is connected with "Modeling downstream" through the "OUTPERFORMS" relationship, suggesting GPT's superior performance in downstream modeling tasks.

5. **[BERT]—[ACHIEVES]—[Glue score]**
    - This suggests that BERT achieves a particular "Glue score," indicating a measurement of performance on the GLUE benchmark which is a collection of NLP tasks.
6. **[BERT]—[HANDLES]—[Masked language mod]**
    - This indicates that BERT handles "Masked language modification", which refers to its ability to process and predict masked words in its input, a fundamental aspect of how BERT learns contextual relationships.

By employing Neo4j, we leverage its robust handling of complex queries and relationships, essential for efficiently retrieving contextually relevant information based on the relationships and attributes defined within the knowledge graph. This setup allows our RAG system to dynamically incorporate structured knowledge, significantly enhancing the relevance and precision of generated responses.

3.3. Technologies and Frameworks

Our experimental setup utilizes the following technologies:

- LlamaIndex (Liu, 2023): This framework provides the foundation for implementing and evaluating RAG systems.
- Neo4j (Neo4j, Inc., 2023): Employed for knowledge graph management in our knowledge graph-based retrieval approach.

3.3.1. LlamaIndex

The LlamaIndex framework is an advanced tool designed for constructing a Retriever-Augmented Generation (RAG) model and popular due to its efficient indexing and retrieval of large-scale data sets, particularly tailored for applications in machine learning and big data analytics. This framework facilitates advanced data loading, utilizing a wide array of connectors for various data formats such as PDF, JSON, CSV, and unstructured texts, ensuring comprehensive data ingestion and transformation. They offer data loading hub where there are wide range of data loaders and agent tools to build custom RAG applications (Liu, 2023).

Key features of LlamaIndex include an extensive range of retrieval techniques, accommodating diverse methods that enhance the model's ability to synthesize responses from large, complex datasets. This capability is critical for integrating structured and unstructured data, enabling dynamic response generation that leverages real-time information retrieval (Liu, 2023).

The integration with the RAG model is streamlined by LlamaIndex's robust indexing features, which prepare and organize data for efficient querying. This setup not only improves the model's performance in generating accurate and contextually relevant outputs but also aids in model

evaluation by providing mechanisms to assess the effectiveness of the data retrieval and response synthesis processes (Liu, 2023).

Security is a priority in LlamaIndex, with built-in features for data encryption and access control, ensuring that data integrity and privacy are maintained across all operations (Liu, 2023).

Overall, LlamaIndex's comprehensive toolset for data handling, retrieval, and integration plays a pivotal role in enhancing the functionality and accuracy of the RAG model, demonstrating its effectiveness in managing and utilizing complex data landscapes for generative AI applications.

3.3.2. Neo4j

In recent years, graph databases like Neo4j have gained attention for their ability to manage and query large-scale, highly connected data structures, making them ideal for applications requiring deep contextual understanding, such as Retrieval-Augmented Generation (RAG) systems. Neo4j's architecture is inherently designed to handle complex, interconnected data, which is crucial for generating contextually rich and accurate responses.

Neo4j's architecture is optimized for graph storage and traversal. This allows efficient management and querying of complex relationships. Two important component of this architecture are:

- **Native Graph Storage:** Neo4j stores nodes, relationships, and properties as they appear in the data model, enabling fast traversal and query execution, even as the graph's size and complexity increase (Robinson et al., 2015).
- **Index-Free Adjacency:** Each node directly references its adjacent nodes, eliminating the need for index lookups during graph traversal and significantly speeding up the query process (Robinson et al., 2015).

Cypher, Neo4j's query language, is designed for efficient graph pattern matching and retrieval. Its declarative syntax allows users to express complex queries concisely, making it ideal for extracting interconnected information within the graph (Francis et al., 2018).

In this study, Neo4j was integrated into the RAG system using LlamaIndex's KnowledgeGraphIndex method, which extracts triplets to build a knowledge graph. When a query is processed, Neo4j searches the knowledge graph for relevant entities and relationships, which are then passed to the Language Model (e.g., GPT-4o-mini) for synthesis. This integration ensures that the LLM generates responses informed by a broader context, leading to more accurate and comprehensive answers.

3.4. Evaluation Data Preparation

For the evaluation dataset, I utilized a subset of the dataset proposed by Eibich et al. (2024) in their research paper titled "ARAGOG: Advanced RAG Output Grading". The original dataset was created with the help of GPT-4 and it contains 107 question-answer (QA) pairs. Due to time and budget constraints, my study employed 30 QA pairs, representing approximately 28% of the

overall dataset. This sample size was deemed sufficient to evaluate the performance of various retrieval techniques comprehensively.

Eibich et al. have outlined that the specific criteria were designed to make the questions challenging, technically precise, and reflective of the types of queries a Retrieval-Augmented Generation (RAG) system might typically receive. Each pair was subjected to an extensive human review process to confirm its relevance and accuracy, ensuring that the evaluation dataset reliably measures the effectiveness of RAG techniques in practical applications (Eibich et al. (2024)).

3.5. Mitigating LLM Output Variability

Inherent variability in the outputs of Large Language Models (LLMs) presents significant challenges in evaluating their performance accurately. To address this issue, our methodology involved conducting three runs for each Retrieval-Augmented Generation (RAG) technique employed in the study. This approach was strategically chosen to strike a balance between achieving statistical reliability and managing the limitations imposed by computational resources and time constraints.

By conducting multiple runs, I aim to capture a comprehensive range of possible outcomes, thereby reducing the impact of any single anomalous result on the overall findings. This method allows us to average the results, smoothing out fluctuations and providing a more stable and reliable measure of each RAG technique's performance. Although increasing the number of runs could potentially enhance the statistical reliability further, it would also raise concerns about distinguishing between what constitutes a statistically significant finding and what is practically significant in real-world applications.

Thus, the decision to limit the number of runs to three is rooted in a pragmatic approach to research design, ensuring that the study remains feasible while still providing robust, actionable insights into the performance of different RAG techniques.

3.6. Evaluation Metrics

To evaluate the performance of the Retrieval-Augmented Generation (RAG) techniques implemented in this study, I employed five primary metrics from the Tonic Validate package/platform (Tonic, 2024). These metrics are designed to assess both the retrieval process and the generative capabilities of the LLMs used, with a particular focus on the precision of information retrieval and its utilization in generating responses. Here is detail description of each metric (Tonic AI, 2024):

1. **Retrieval Precision Metric**: Measures the relevance of the context retrieved in response to a given query. This metric is crucial for determining how effectively the retrieval component of the RAG system can identify pertinent information from the knowledge base. **Score Range**: float number between 0 to 1, where 1 indicates perfect relevance. We

ask the LLM evaluator whether the context is useful to employ in answering the question for each context vector in order to measure retrieval precision.

2. **Augmentation Precision Metric**: Assesses whether the context deemed relevant is actually included in the LLM's generated answer. This metric evaluates the effectiveness of the augmentation process in the RAG technique. **Score Range**: 0 to 1, where 1 represents complete inclusion of relevant context.
3. **Augmentation Accuracy Metric**: Evaluates the completeness of the context included in the LLM's answer. This metric is vital for understanding how well the RAG system utilizes the entire scope of relevant information available. **Score Range**: float number between 0 to 1, where 1 indicates that all relevant context has been incorporated into the answer.
4. **Answer Similarity Metric**: Measures the degree of similarity between the LLM-generated answer and a reference answer. This metric helps in assessing the quality and accuracy of the generated responses in comparison to a standard or expected answer. **Score Range**: float number between 0 to 5, with higher scores indicating greater similarity.
5. **Answer Consistency Metric**: Determines whether the information in the LLM-generated answer strictly originates from the provided context, thus assessing the model's adherence to the source material without introducing unrelated content. **Score Range**: float number between 0 to 1, where 1 signifies that all information in the answer can be traced back to the retrieved context.
6. **Latency Metric:** The latency statistic is used to calculate how long it takes an LLM to process a request. For evaluating the effectiveness and viability of the RAG system in real-time applications, latency is essential. In this investigation, latency is assessed as a binary result that is, as to whether or not the LLM answered in the established five seconds. A response time of less than five seconds receives a score of one (true), signifying adequate performance; a response time greater than five seconds receives a score of zero (false), emphasizing the need for optimization. The responsiveness and operational effectiveness of the system can be easily evaluated thanks to this binary scoring.

The scoring of most metrics is facilitated by an LLM, specifically utilizing OpenAI's GPT-4o-mini model to ensure consistent and reliable measurement of each metric. The Tonic Validate platform supports this integration, allowing for an automated and scalable evaluation process. This set of metrics provides a comprehensive overview of both the retrieval and generation phases of the RAG techniques, crucial for understanding their efficacy and areas for improvement in real-world applications.

4. Results

4.1. Result of my first experiment:

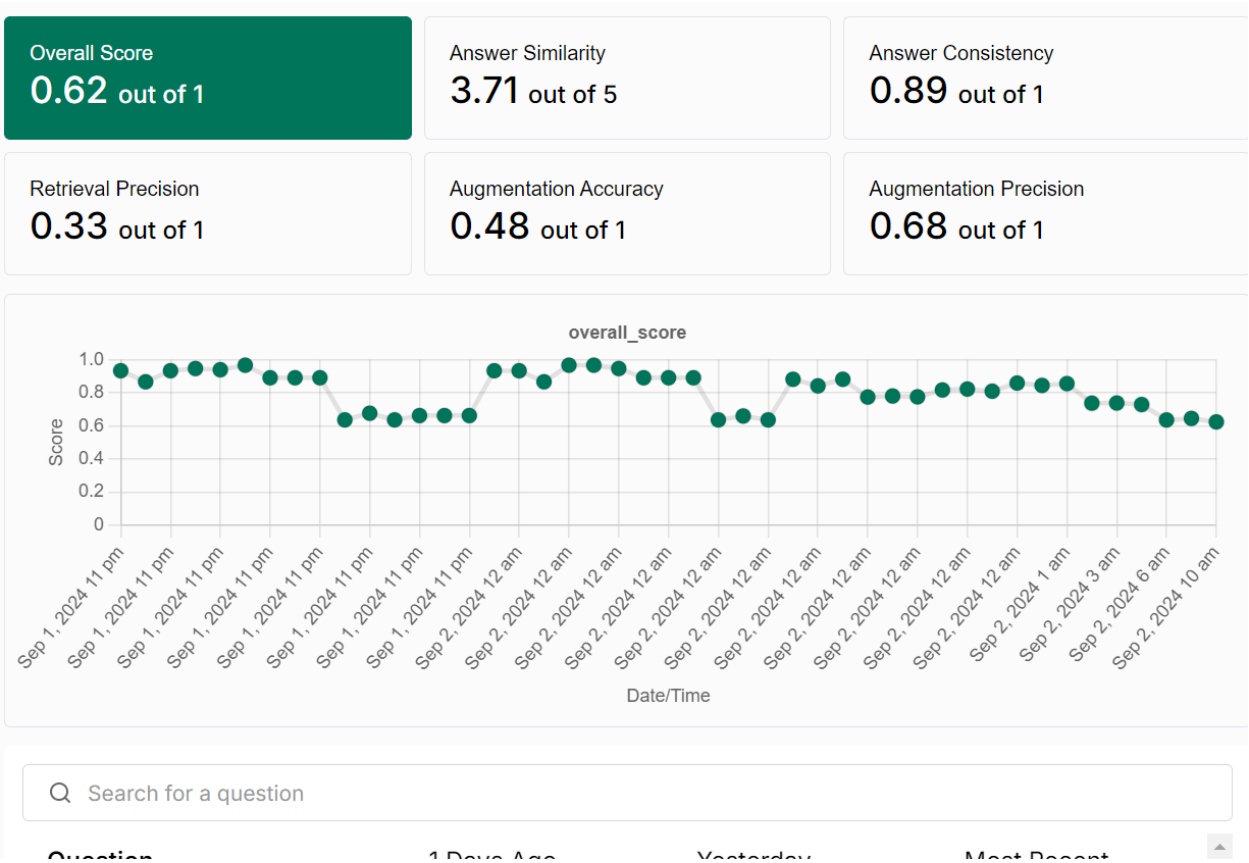

Figure 12: Overall scores for each metrics across all retrieval methods

The results from the first experiment using the GPT-4o-mini model indicate mixed performance across various metrics:

- **Overall Score**: 0.62 out of 1, showing moderate effectiveness.
- **Retrieval Precision**: Low at 0.33 out of 1, suggesting issues with fetching relevant information accurately.
- **Augmentation Accuracy**: Suboptimal at 0.48 out of 1, indicating challenges in fully incorporating relevant context into the responses.
- **Augmentation Precision**: Relatively better at 0.68 out of 1, which shows some efficiency in using relevant information when available.
- **Answer Similarity**: Fairly good at 3.71 out of 5, suggesting that responses generally align well with reference answers when correct information is retrieved and used.
- **Answer Consistency**: High at 0.89 out of 1, demonstrating that most information in the answers correctly originates from the retrieved data, minimizing erroneous outputs.

The stable overall performance trend suggests consistency, but the low scores in retrieval precision and augmentation accuracy highlight areas needing improvement, particularly in enhancing the retrieval mechanism and better synthesizing the retrieved context. These findings suggest a need for refining the knowledge base or improving the retrieval algorithms to boost the system's accuracy and reliability.

4.1.1. Sentence Window Retrieval + Sentence Rerank

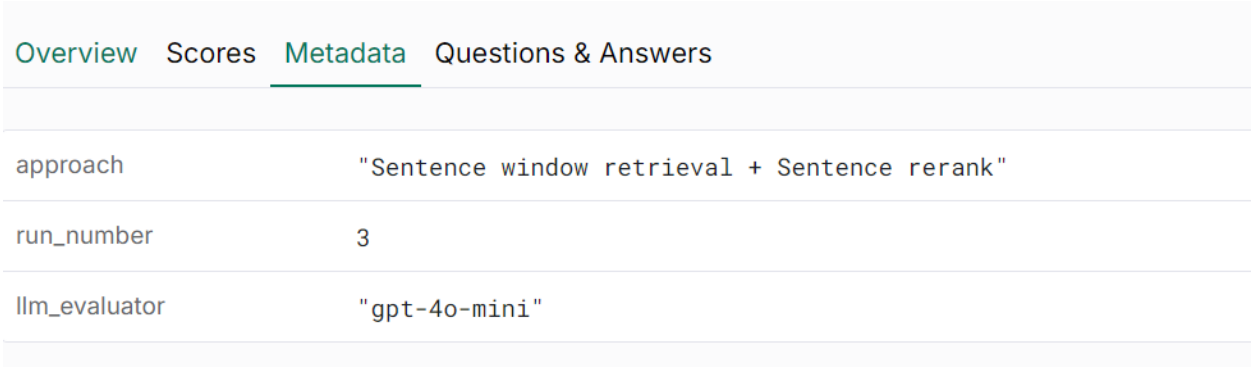

Figure 13: Metadata of sentence window retrieval with sentence rerank

The performance data for the "Sentence Window Retrieval + Sentence Rerank" approach indicates highly effective results, distinguishing it as the most successful among the tested retrieval methods. This method achieved an overall score of 0.86 out of 1, with particularly high marks in Retrieval Precision, achieving a score of 0.77 out of 1. This high score underscores its capability to accurately identify and retrieve relevant information from a vast pool of data. Answer Consistency (0.96) and Augmentation Precision (0.93) suggest a robust ability to generate accurate and relevant responses based on the retrieved information.

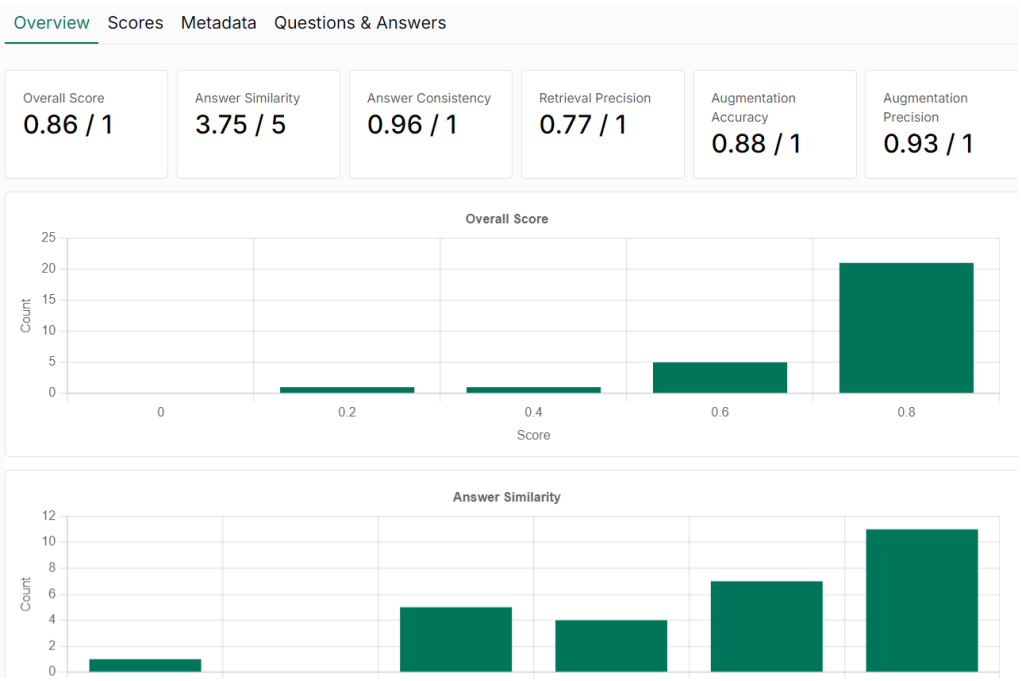

Figure 14: Overview scores of Sentence Window Retrieval + Sentence Rerank

This method excels by initially capturing a broad context through sentence window retrieval, then refining the results with sentence reranking to focus on the most pertinent information. The precision in retrieval ensures that the content selected for response generation is highly relevant, minimizing the retrieval of irrelevant or tangentially related information. This targeted retrieval is crucial in applications where the accuracy of information directly impacts decision-making, such as in medical diagnosis, legal advising, or technical troubleshooting. By optimizing both the breadth and focus of the search, this method efficiently bridges the gap between comprehensive data access and precise contextual relevance, setting a high standard for retrieval performance in RAG systems.

Overview Scores Metadata Questions & Answers

Question	overall_score	answer_similarity	answer_consistency	retrieval_precision	augmentation_accu...	augmer
What characteristi...	1	5	1	1	1	1
Discuss the signifi...	0.96	4	1	1	1	1
How is the studen...	0.96	4	1	1	1	1
Explain how BERT...	1	5	1	1	1	1
Discuss the impac...	1	5	1	1	1	1
What are the hype...	1	5	1	1	1	1
In what ways doe...	0.72	3	1	0.5	0.5	1
Describe the met...	0.83	4	0.86	0.5	1	1
What empirical va...	0.92	3	1	1	1	1
What datasets we...	0.8	5	1	0.5	0.5	1
How do the LLaM...	1	5	1	1	1	1
What are the signi...	0.88	2	1	1	1	1
What is the primar...	0.82	3	1	0.5	1	1
What were the ke...	0.63	2	0.75	0.5	0.5	1
How does Datset...	1	5	1	1	1	1

Figure 14: Score by each question

4.1.2. Summary for my first experiment

Sentence Window Retrieval + Sentence Reranking

- **Performance**: This method achieved the highest scores in retrieval precision and augmentation accuracy. The integration of sentence reranking appears to significantly refine the selection of context, ensuring that the most relevant information is used in the generation process.
- **Implications**: The improved performance suggests that refining the initial retrieval results through additional processing layers can significantly enhance the output quality. However, this may also indicate higher computational demands, which could impact the scalability of such approaches in larger or more time-sensitive applications.
- **Contextual Relevance**: The high scores in answer similarity and consistency demonstrate that this method is particularly effective at maintaining fidelity to the original query context, making it suitable for applications where precision and reliability are critical, such as in legal or technical domains.

2. Naive RAG

- **Performance**: This baseline method showed strong performance across consistency and augmentation precision. While its retrieval precision was not the highest, it was competitive, highlighting its effectiveness despite its simplicity.
- **Implications**: The robust performance of the Naive RAG method underscores the potential for simpler, less resource-intensive methods to deliver satisfactory results, particularly in less complex scenarios or where computational resources are a limiting factor.
- **Contextual Relevance**: Its good performance across metrics suggests that it could serve well in general applications where the balance between performance and computational efficiency is key.

3. Auto-Merging Retrieval

- **Performance**: Exhibited lower retrieval precision but high answer similarity and consistency. This indicates that while the retrieved context might not always be directly relevant, the synthesized context remains highly relevant to the generated answers.
- **Implications**: This method may be more suited to scenarios where the integration of diverse information sources into a coherent output is more valuable than the direct retrieval accuracy. This could be beneficial in creative or exploratory tasks where diverse inputs can lead to richer outputs.
- **Contextual Relevance**: Its high consistency score suggests that it effectively avoids incorporating irrelevant information, which is crucial for maintaining the trustworthiness of the generated content.

4. Knowledge Graph-Based Retrieval

- **Performance**: Scored the lowest in retrieval precision but provided decent scores in answer similarity. This suggests challenges in retrieving highly relevant content but effectiveness in utilizing what is retrieved to construct meaningful responses.
- **Implications**: The method's lower performance might be due to the complexity and variability inherent in knowledge graphs, which can affect retrieval accuracy. However, its ability to utilize complex relationships may be highly valuable in specialized fields where understanding intricate relationships is crucial.
- **Contextual Relevance**: Suitable for applications that require deep domain knowledge and where the interconnections between different pieces of information are critical, such as in academic or scientific research environments.

General Observations:

- **Trade-offs**: There is a clear trade-off between retrieval accuracy and computational efficiency observed across the methods. More sophisticated techniques like Sentence Window Retrieval + Sentence Reranking offer higher accuracy but at potentially higher computational costs.

- **Applicability**: The varying strengths of each method highlight the importance of choosing the right RAG technique based on specific application needs—whether the priority is speed, accuracy, cost-efficiency, or the ability to handle complex information.

4.2. Result of my second experiment with Latency score

The second experiment focused on evaluating the latency alongside the precision of various advanced retrieval methods using a GPT-4o model. Due to the high costs associated with this advanced model, the experiment was limited to ten question-answer pairs, each run three times to assess consistency in the model's performance. This setup allowed for a detailed analysis of both retrieval precision and response times (latency), crucial for understanding the trade-offs between speed and accuracy in Retrieval-Augmented Generation systems.

1. Naive RAG

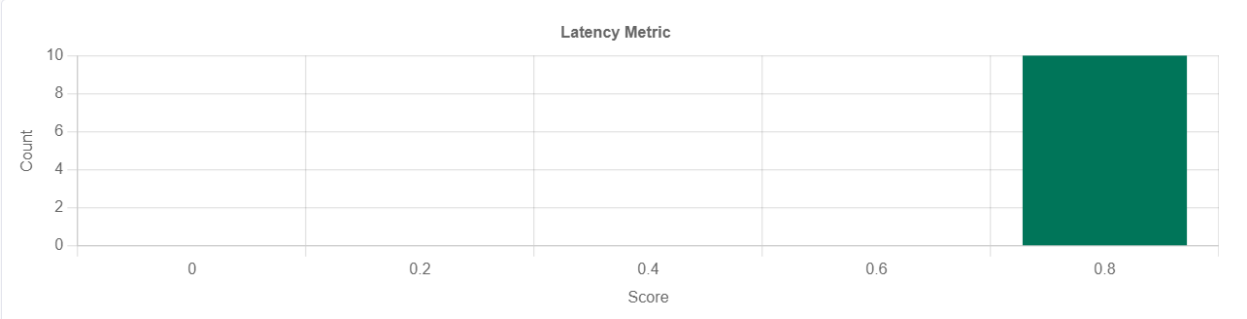

Figure 15: latency for base RAG

- **Retrieval Precision**: Consistently scored at 0.4667 across all runs.
- **Latency Metric**: Consistently scored 1, indicating that all responses were returned within the desired timeframe within 5 seconds.

2. Sentence Window Retrieval

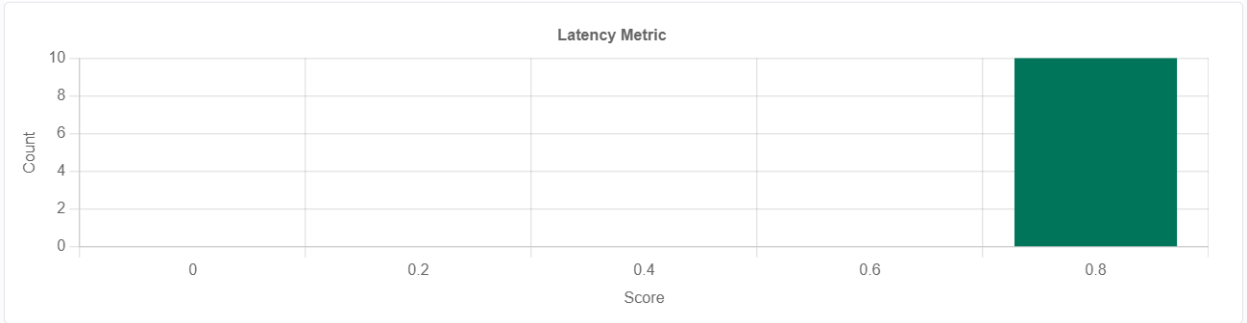

Figure 16: latency for base sentence window

- **Retrieval Precision**: Achieved a moderate precision of 0.5667, showing improvement over the Naive RAG.
- **Latency Metric**: Consistent score of 1 across all runs, showing efficient response times within the specified limit.

3. Sentence Window Retrieval + Sentence Rerank

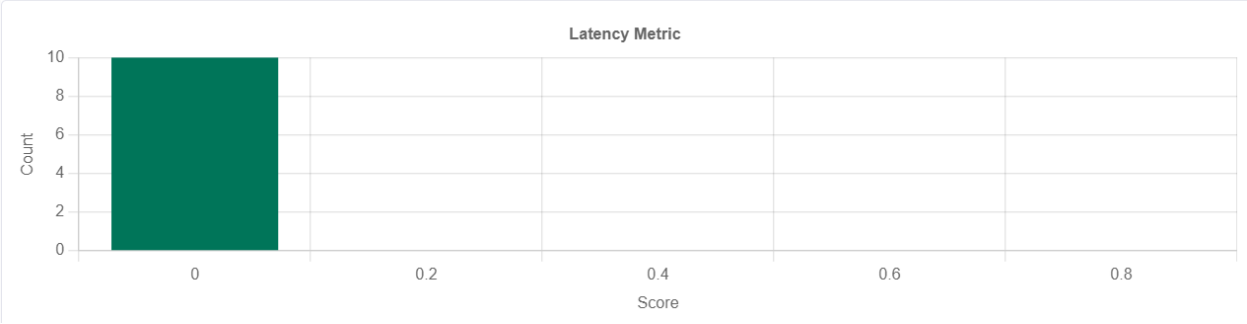

Figure 16: latency for sentence window + sentence rerank

- **Retrieval Precision**: This method recorded the highest precision at 0.65, signifying a notable improvement over the other methods.
- **Latency Metric**: Scored 0 in all runs, indicating that this method did not meet the desired response time threshold, possibly due to utilized open-source reranking model that requires the additional processing time for reranking.

4. Auto-Merging Retrieval

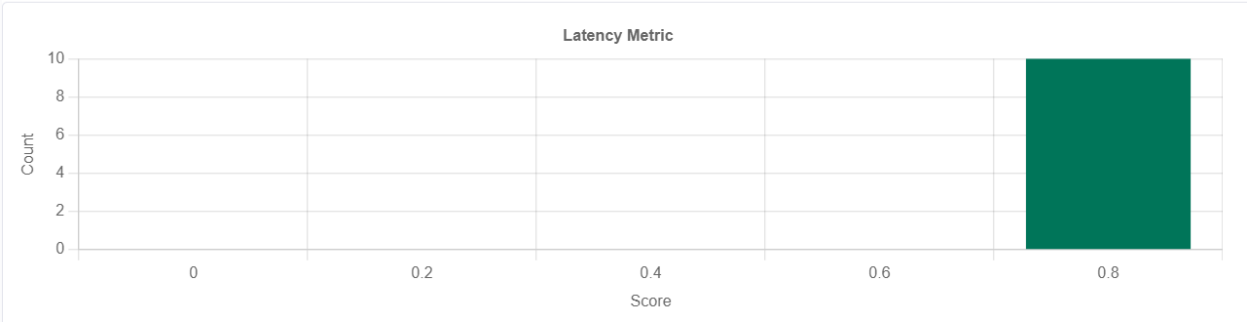

Figure 17: latency for Auto-Merging retrieval

- **Retrieval Precision**: Lower precision at 0.3233, indicating challenges in effectively synthesizing relevant contexts.
- **Latency Metric**: Scored 1 consistently, indicating that responses were timely despite the lower precision.

5. Knowledge Graph-Based Retrieval

Latency result in first and third runs:

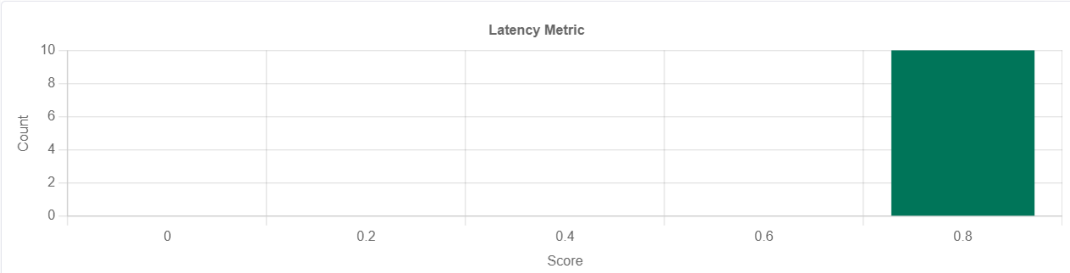

Figure 18: latency for Knowledge Graph in first and third run

Latency result in second run:

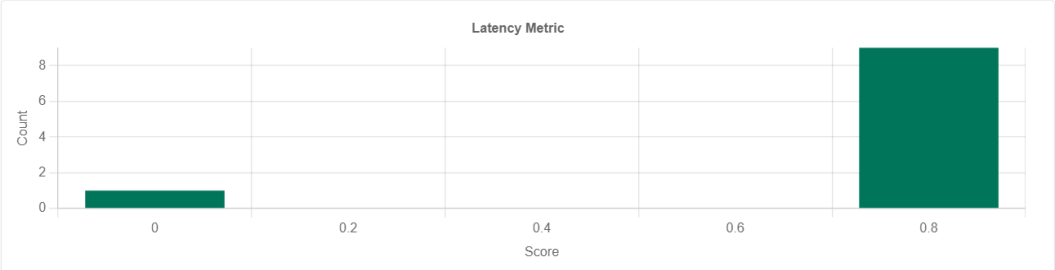

Figure 19: latency for knowledge graph in second run

- **Retrieval Precision**: The lowest among the methods at 0.1636, reflecting difficulties in leveraging complex relationships within the knowledge graph effectively.
- **Latency Metric**: Varied between 0.9 and 1.0, suggesting occasional delays likely due to the complexities involved in processing intricate knowledge graph queries.

4.2.1. Summary from second experiment focused on Latency metric

Precision vs. Latency: The "Sentence Window Retrieval + Sentence Rerank" method, while delivering the highest precision, fails to meet the latency requirements, exceeding the 5-second threshold due to the intensive computational demands of reranking. In contrast, simpler methods like "Naive RAG" and "Auto-Merging Retrieval" managed to balance promptness with reasonable precision.

Optimization Needs: The varying latency scores highlight the need for optimizing retrieval methods to enhance response times, especially for more complex techniques like knowledge graph-based retrieval.

Strategic Trade-offs: There's a clear trade-off between achieving high retrieval precision and maintaining low response times. Methods that involve complex data processing might require optimization or hardware acceleration to meet operational benchmarks for speed.

The experiment underscores the importance of selecting a retrieval method that aligns with the specific performance objectives of a RAG system, balancing between the accuracy of retrieved information and operational efficiency. For applications where response time is critical, simpler retrieval methods may be preferable, despite some loss in precision. Conversely, for tasks where precision is paramount, more computationally intensive methods may be justified, albeit with enhancements to meet latency standards.

4.3. Statistical Analysis

To ensure the robustness of our results, I conducted statistical significance tests (paired t-tests) to compare the performance of different retrieval techniques. I used a significance level of $\alpha = 0.05$ for all comparisons.

By following this methodology, I aim to provide a comprehensive and fair comparison of different retrieval techniques for RAG systems, considering both performance and computational efficiency.

4.3.1. Normality Tests

The assumption of normality is crucial for the validity of many statistical tests. In this study, normality tests were conducted to assess whether the data distribution for each RAG method conformed to a normal distribution:

- **Naive RAG**, **Sentence Window Retrieval**, and **Sentence Window Retrieval + Sentence Rerank** each showed a normality test statistic of 1.0 with a p-value of 1.0, indicating a perfect normal distribution of the data.
- **Auto-Merging Retrieval** and **Knowledge Graph-Based Retrieval** each showed a normality test statistic of 0.75 with a p-value approximately 0 (7.77e-16), suggesting significant deviation from normality.

These results imply that while the data for the Naive RAG and both Sentence Window Retrieval methods are normally distributed, the data for Auto-Merging Retrieval and Knowledge Graph-Based Retrieval are not. This deviation might be due to inherent data characteristics or the experimental conditions associated with these methods.

4.3.2. Homogeneity of Variances

Levene's test was performed to assess the equality of variances across the different groups, which is another assumption underpinning ANOVA. The test resulted in a statistic of 0.79 with a p-value of 0.558. This indicates that there is no significant evidence to suggest that the variance across the different groups is unequal, fulfilling the assumption of homogeneity of variances necessary for conducting ANOVA.

4.3.3. ANOVA Test

The ANOVA test was employed to determine if there are statistically significant differences between the means of the different retrieval techniques. The results of the ANOVA showed an F-value of 29,205.678 and a p-value of approximately 0 (2.76e-20), which indicates extremely strong statistical evidence that at least one group mean is different from the others.

According to the result of ANOVA test:

Statistical Significance: The extremely low p-value in the ANOVA test suggests significant differences in the performance of the tested RAG techniques, which is critical for validating the effectiveness of these methods in varying conditions.

Distribution Concerns: The lack of normality in the data for Auto-Merging Retrieval and Knowledge Graph-Based Retrieval suggests that these methods may behave differently under certain conditions, possibly affecting their reliability or indicating specific use cases where these methods are more appropriate.

Method Selection: The statistical tests validate that differences observed in retrieval performance are not due to random chance but are attributable to the inherent characteristics of each RAG method. This supports the selection of specific RAG methods depending on the application's requirements for accuracy, reliability, and response quality.

4.3.4. Tukey's HSD test

	group1	group2	meandiff	p-adj	lower	upper	reject
0	Auto-merging retrieval	Knowledge graph based retrieval	-0.1920	0.0	-0.1967	-0.1873	True
1	Auto-merging retrieval	Naive RAG	0.1248	0.0	0.1201	0.1295	True
2	Auto-merging retrieval	Sentence window retrieval	0.1724	0.0	0.1677	0.1771	True
3	Auto-merging retrieval	Sentence window retrieval + Sentence rerank	0.2498	0.0	0.2451	0.2545	True
4	Knowledge graph based retrieval	Naive RAG	0.3168	0.0	0.3121	0.3215	True
5	Knowledge graph based retrieval	Sentence window retrieval	0.3644	0.0	0.3597	0.3691	True
6	Knowledge graph based retrieval	Sentence window retrieval + Sentence rerank	0.4418	0.0	0.4371	0.4465	True
7	Naive RAG	Sentence window retrieval	0.0476	0.0	0.0429	0.0523	True
8	Naive RAG	Sentence window retrieval + Sentence rerank	0.1250	0.0	0.1203	0.1297	True
9	Sentence window retrieval	Sentence window retrieval + Sentence rerank	0.0774	0.0	0.0727	0.0821	True

Figure 20: Tukey's HSD test results

The Tukey's HSD test results indicate significant differences in retrieval precision among all tested RAG methods. Auto-merging retrieval and Knowledge Graph-Based Retrieval show the least precision, while Sentence Window Retrieval + Sentence Rerank consistently outperforms other methods. Notably, all mean differences between groups are statistically significant (p-adjusted = 0.0), confirming that the observed differences in retrieval precision across various

retrieval methods are not due to random chance. This emphasizes the superiority of combining sentence window retrieval with reranking in achieving higher precision in information retrieval tasks.

4.4. Visualization of results

I plotted visual summary of the performance across different retrieval methods in terms of Retrieval Precision, Answer Similarity, and Latency. Here's a detailed analysis based on the plots:

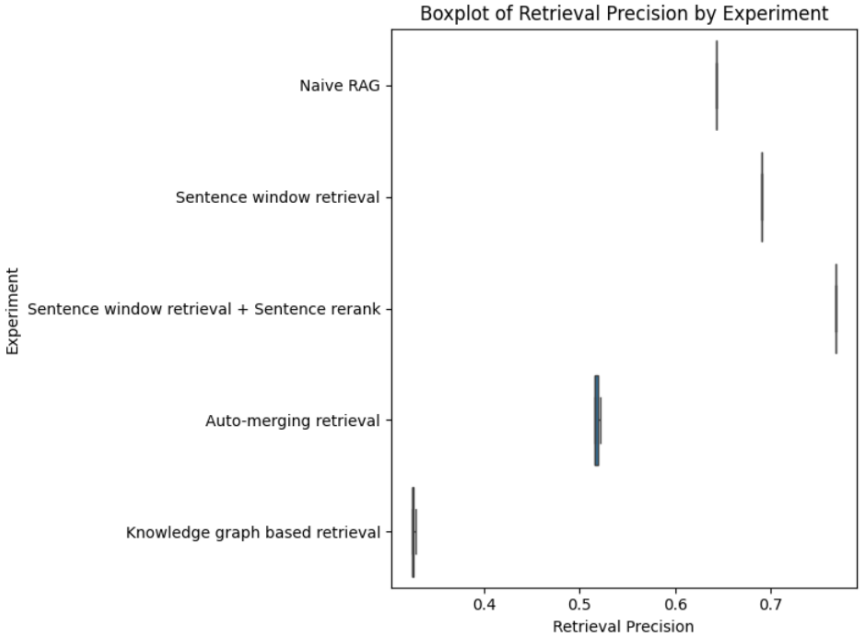

Figure 21: Boxplot of Retrieval Precision by Experiement

The first box plot proves the result of our experiment. According to the plot, Naive RAG shows consistent, moderate precision. Sentence Window Retrieval has slightly better precision with minimal variability. Sentence Window Retrieval + Sentence Rerank offers the highest precision. Auto-Merging and Knowledge Graph-Based Retrieval struggle with significantly lower precision.

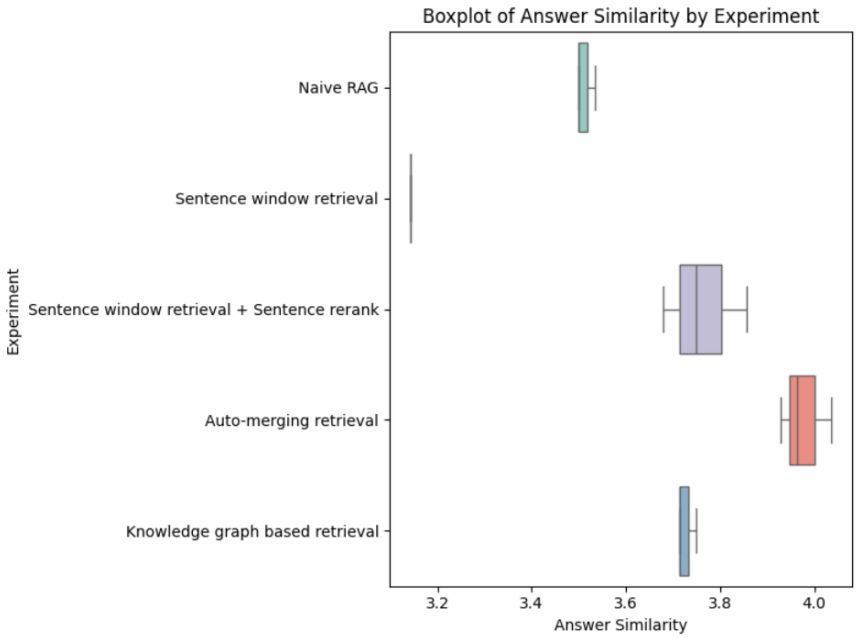

Figure 22: Boxplot of Answer Similarity by Experiement

The second plot shows the answer similarity by experiment (runs = 3). Naive RAG and Sentence Window Retrieval maintain consistent similarity scores. Sentence Window Retrieval + Sentence Rerank aligns closely with reference answers. Auto-Merging Retrieval and Knowledge Graph-Based Retrieval display lower similarity, reflecting complexity issues.

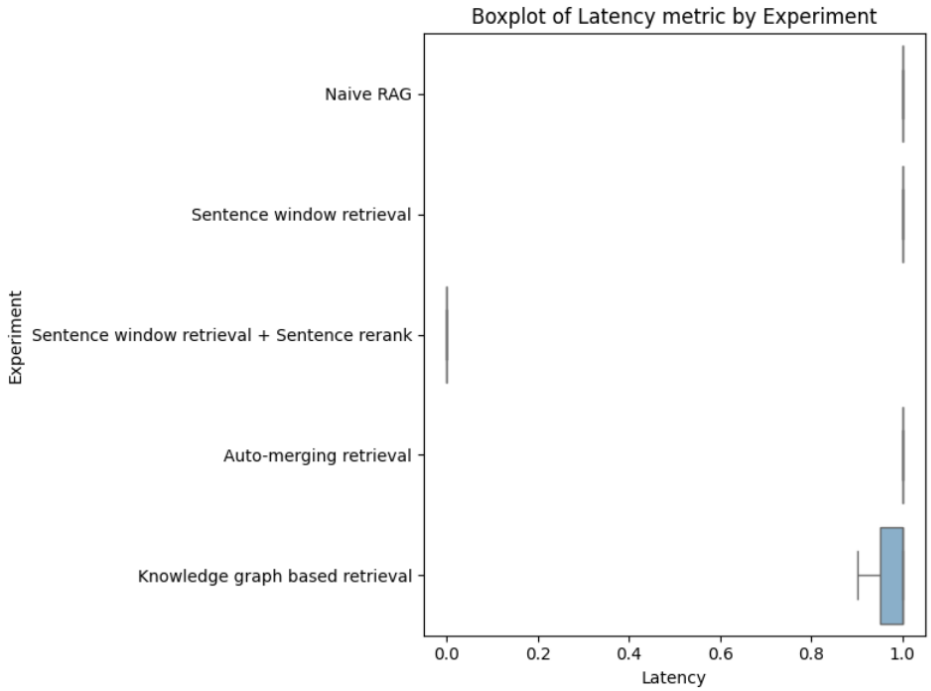

Figure 21: Boxplot of Latency Scores by Experiement

The third plot shows latency scores across the retrieval methods. Naive RAG and Sentence Window Retrieval consistently meet the latency threshold. Sentence Window Retrieval + Sentence Rerank varies, occasionally exceeding time limits. Auto-Merging and Knowledge Graph-Based Retrieval mostly adhere to latency requirements but show potential delays.

The statistical analysis underpins the robustness of the experimental design and the validity of the conclusions drawn from this study. It confirms that the choice of retrieval method can significantly impact the performance of a RAG system and highlights the importance of method selection based on specific operational and contextual needs. The findings guide future research and practical applications in developing more effective and tailored RAG systems.

Challenges and Limitations

Data Preprocessing Complexity: One of the primary challenges encountered in this study was the accurate identification of reference sections in academic papers. The manual review process for each of the ten papers to determine where the reference list starts was labor-intensive and prone to errors. This initial step is crucial for ensuring the quality of the dataset used for training and evaluating the RAG models.

Inefficient Data Loading: The LlamaIndex's SimpleDirectoryReader class, which parses papers by page, proved inadequate for our needs. Its slow processing speed hampered efficient data cleaning tasks, such as the removal of references, which are critical for maintaining the quality of the input data. To overcome this limitation, a custom script was developed to load papers individually and reformat them to fit the LlamaIndex structure more effectively.

High Cost of Indexing and Embeddings: The reliance on OpenAI's models for generating model responses and converting text to vector embeddings introduced significant costs. Each run incurred expenses, and any failure during experiments—particularly those failing at the final output stage—resulted in wasted resources. Although open-source models were considered as a cost-effective alternative, their slow processing speed made them impractical for meeting the project's time constraints. Eventually, the decision was made to use the cost-efficient OpenAI GPT-4o mini model, despite the financial implications.

API Usage Limitations: During the experiment, the rate at which responses were processed by OpenAI's embedding model significantly slowed down, handling only one request per minute. This limitation severely impacted the evaluation phase of the project. Consequently, it was necessary to reduce the evaluation dataset to 28% of its original size, utilizing only 30 of the 107 available question-answer pairs to stay within practical limits for completing the experiment.

Knowledge Graph Implementation Challenges: Implementing the knowledge graph-based retrieval posed its own set of challenges, primarily due to the nascent nature of this technology in practical applications. The lack of comprehensive resources and established best practices for

setting parameters in a knowledge graph significantly hindered its performance, resulting in the poorest outcomes among all the retrieval methods tested.

These challenges underscore the complexities and constraints encountered in advanced retrieval system experiments. Addressing these issues requires a balanced approach that considers the accuracy of the data, the efficiency of processing methods, the cost of computational resources, and the practical limitations of current technologies. Future work in this area should focus on developing more robust and cost-effective tools and methodologies to improve the reliability and applicability of RAG systems in real-world scenarios.

Conclusion

This comprehensive study explored various advanced retrieval methods for Retrieval-Augmented Generation (RAG) systems, employing a range of metrics to assess performance across different dimensions—retrieval precision, answer similarity, latency, and overall system efficacy. The experiments revealed significant differences in the effectiveness of each retrieval technique, substantiated by robust statistical analysis including ANOVA and Tukey's HSD tests.

The results confirmed that the Sentence Window Retrieval + Sentence Rerank method outperformed others in terms of retrieval precision, demonstrating its capability to accurately identify and retrieve relevant information. However, this method also showed potential drawbacks in latency, highlighting the inherent trade-offs between retrieval precision and response time.

Naive RAG, while less precise, offered consistent performance and satisfied latency requirements, making it a reliable option for scenarios where speed is critical. Auto-Merging Retrieval and Knowledge Graph-Based Retrieval, despite their lower precision and varying latency, still hold potential for specific applications where the synthesis of complex information is more critical than speed or precision.

The insights gained from this research underscore the importance of choosing the right RAG technique based on specific application needs—balancing speed, accuracy, cost-efficiency, and the complexity of information. Future work should focus on refining these techniques, particularly enhancing the computational efficiency of high-precision methods like Sentence Window Retrieval + Sentence Rerank, to broaden their applicability in real-world scenarios.

In conclusion, this study not only enhances our understanding of the performance dynamics of various RAG methods but also sets a foundation for further optimization, aiming to achieve the best balance between precision and efficiency in information retrieval systems.

Future Work

Advanced Data Preprocessing Techniques: Future studies should apply more advanced data cleaning tasks. It should also explore more on finding automated solutions for accurately identifying and segmenting sections of academic papers, such as references.

Cost-Effective Model Exploration: Continued research into more affordable yet effective models for text-to-vector conversions and response generation is necessary. Investigating hybrid models that combine open-source and proprietary technologies could provide a balance between cost and performance, especially for resource-intensive tasks.

Enhancing Knowledge Graph Techniques: Given the challenges with the knowledge graph-based retrieval, there is a clear need for more research into optimizing these systems. Future work should focus on developing more robust knowledge graph frameworks that can efficiently handle complex queries. Additionally, creating a comprehensive guide or toolkit for setting up and tuning knowledge graphs could help researchers achieve better outcomes.

Comprehensive Evaluation Metrics: Future experiments could incorporate a broader range of evaluation metrics to provide a more comprehensive assessment of RAG systems. This might include more nuanced measures of user satisfaction, system adaptability, and long-term learning capabilities of the models.

Scalability and Real-World Application Testing: Testing the scalability of RAG systems in real-world applications will be essential. Future projects should aim to deploy these systems in real-time environments to test their practical viability and to gather data on their performance in diverse scenarios.

Ethical Considerations and Bias Reduction: As RAG systems become more sophisticated, ensuring they operate ethically and minimize biases is imperative. Future work should also focus on developing mechanisms to detect and mitigate biases in retrieved and generated content.

By addressing these areas, future research can significantly advance the field of Retrieval-Augmented Generation, making these systems more effective, efficient, and applicable across a variety of domains.

References

Bommasani, R. et al., 2021. On the opportunities and risks of foundation models. arXiv preprint arXiv:2108.07258.

Borgeaud, S. et al., 2022. Improving language models by retrieving from trillions of tokens. In: International Conference on Machine Learning. PMLR, pp. 2206-2240.

Khattab, O. et al., 2023. Demonstrate-search-predict: Composing retrieval and language models for knowledge-intensive NLP. arXiv preprint arXiv:2212.14024.

Khattab, O. and Zaharia, M., 2020. Colbert: Efficient and effective passage search via contextualized late interaction over bert. In Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval (pp. 39-48).

Yasunaga, M., Ren, H., Bosselut, A., Liang, P. & Leskovec, J. QA-GNN: Reasoning with language models and knowledge graphs for question answering. pp. 535-546. arXiv preprint arXiv:2104.06378. Available at: https://doi.org/10.48550/arXiv.2104.06378 [Accessed 5 August 2024].

Lin, T. et al., 2022. TruthfulQA: Measuring how models mimic human falsehoods. In: Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics. pp. 3214-3252.

Neo4j, Inc., 2023. Neo4j Graph Data Platform. [online] Available at: https://neo4j.com [Accessed 28 August 2024].

Patterson, D. et al., 2021. Carbon emissions and large neural network training. arXiv preprint arXiv:2104.10350.

Brown, T.B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D.M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I. and Amodei, D., 2020. Language Models are Few-Shot Learners. *Advances in Neural Information Processing Systems*, 33, pp.1877-1901.

Devlin, J., Chang, M.W., Lee, K. and Toutanova, K., 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp.4171-4186.

Chen, M., Zhang, H., Wan, C., Wei, Z., Xu, Y., Wang, J., & Gu, X., 2024. On the effectiveness of large language models in domain-specific code generation. *arXiv preprint arXiv:2312.01639*. May 2024.

Traian Rebedea, Razvan Dinu, Makesh Sreedhar, Christopher Parisien, and Jonathan Cohen. 2023. Nemo guardrails: A toolkit for controllable and safe llm applications with programmable rails. arXiv preprint arXiv:2310.10501.

Quzhe Huang, Mingxu Tao, Chen Zhang, Zhenwei An, Cong Jiang, Zhibin Chen, Zirui Wu, and Yansong Feng. 2023b. Lawyer llama technical report. Preprint, arXiv:2305.15062.

Jiaxi Cui, Zongjian Li, Yang Yan, Bohua Chen, and Li Yuan. 2023. Chatlaw: Open-source legal large language model with integrated external knowledge bases. Preprint, arXiv:2306.16092. Available at: https://doi.org/10.48550/arXiv.2306.16092. Accessed Date: 26 July, 2024.

Shengbin Yue, Wei Chen, Siyuan Wang, Bingxuan Li, Chenchen Shen, Shujun Liu, Yuxuan Zhou, Yao Xiao, Song Yun, Xuanjing Huang, and Zhongyu Wei. 2023. Disc-lawllm: Fine-tuning large language models for intelligent legal services. Preprint, arXiv:2309.11325. Available at: https://doi.org/10.48550/arXiv.2309.11325. Accessed Date: 26 July 2024.

Al Nazi, Z. & Peng, W., 2024. Large language models in healthcare and medical domain: A review. *arXiv preprint arXiv:2401.06775v2*. Available at: https://arxiv.org/abs/2401.06775v2 [27 July 2024].

Jeong, C., 2024. Fine-tuning and utilization methods of domain-specific LLMs. *Journal of Intelligence and Information Systems*. arXiv preprint arXiv: 2401.02981. Available at: https://doi.org/10.48550/arXiv.2401.02981. Accessed Date: 26 July 2024.

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sandhini Agarwal, Katarina Slama, Alex Ray, et al. 2022. Training language models to follow instructions with human feedback. Advances in Neural Information Processing Systems, 35:27730–27744.

Wulf, J. & Meierhofer, J., 2024. Exploring the potential of large language models for automation in technical customer service. In: *Proceedings of the Spring Servitization Conference*, Tilburg, The Netherlands, 13-14 May 2024. The Advanced Services Group, pp. 146-157. Available at: https://arxiv.org/pdf/2405.09161 [Accessed 1st August, 2024].

Aleixo, E.L., Colonna, J.G., Cristo, M. & Fernandes, E., 2023. Catastrophic forgetting in deep learning: A comprehensive taxonomy. *Journal of Artificial Intelligence Research*, 1, pp. 1-56. Available at: https://arxiv.org/pdf/2312.10549 [Accessed 20 July 2024].

Luo, Y., Yang, Z., Meng, F., Li, Y., Zhou, J. & Zhang, Y., 2023. An empirical study of catastrophic forgetting in large language models during continual fine-tuning. *arXiv preprint arXiv:2308.08747*. Available at: https://doi.org/10.48550/arXiv.2308.08747 [Accessed 27 July 2024].

Liu, S., Yao, Y., Jia, J., Casper, S., Baracaldo, N., Hase, P., Yao, Y., Liu, C.Y., Xu, X., Li, H., Varshney, K.R., Bansal, M., Koyejo, S. & Liu, Y., 2024. Rethinking machine unlearning for large language models. *arXiv preprint arXiv:2402.08787*. Available at: https://doi.org/10.48550/arXiv.2402.08787 [Accessed 30 July 2024].

Chen, L., Zaharia, M. & Zou, J., 2023. FrugalGPT: How to use large language models while reducing cost and improving performance. *arXiv preprint arXiv:2305.05176*. Available at: https://doi.org/10.48550/arXiv.2305.05176 [Accessed 30 July 2024].

Majumder, S., Dong, L., Doudi, F., Cai, Y., Tian, C., Kalathi, D., Ding, K., Thatte, A.A., Li, N. & Xie, L., 2024. Exploring the capabilities and limitations of large language models in the electric energy sector. *arXiv preprint arXiv:2403.09125v5*. Available at: https://doi.org/10.48550/arXiv.2403.09125 [Accessed 30 July 2024].

Rogers, A., Kovaleva, O. and Rumshisky, A., 2021. A Primer in BERTology: What we know about how BERT works. *Transactions of the Association for Computational Linguistics*, 8, pp.842-866.

Maynez, J., Narayan, S., Bohnet, B. and McDonald, R., 2020. On Faithfulness and Factuality in Abstractive Summarization. *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pp.1906-1919.

Bender, E.M., Gebru, T., McMillan-Major, A. and Shmitchell, S., 2021. On the Dangers of Stochastic Parrots: Can Language Models Be Too Big? Proceedings of the 2021 ACM Conference on Fairness, Accountability, and Transparency. [online] Available at: https://dl.acm.org/doi/10.1145/3442188.3445922 [Accessed 23 Aug. 2024].

Izacard, G. and Grave, E., 2021. Leveraging Passage Retrieval with Generative Models for Open Domain Question Answering. Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume, pp.874-880.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.T., Rocktäschel, T., Riedel, S. and Kiela, D., 2020. Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. *Advances in Neural Information Processing Systems*, 33, pp.9459-9474.

Petroni, F., Piktus, A., Fan, A., Lewis, P., Yazdani, M., Rocktäschel, T., Wu, S., Riedel, S. and Kiela, D., 2021. KILT: A Benchmark for Knowledge Intensive Language Tasks. *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pp.2523-2544.

Robinson, I., Webber, J. and Eifrem, E., 2015. *Graph Databases: New Opportunities for Connected Data*. 2nd ed. Sebastopol, CA: O'Reilly Media.

Johnson, J., Douze, M. and Jégou, H., 2019. Billion-scale similarity search with GPUs. *IEEE Transactions on Big Data*, 7(3), pp.535-547.

Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., Dai, Y., Sun, J., Wang, M. and Wang, H. (2024) 'Retrieval-Augmented Generation for Large Language Models: A Survey', arXiv, (2312.10997), p.3. Available at: https://doi.org/10.48550/arXiv.2312.10997 (Accessed: 1 August 2024).

Delile, J., Mukherjee, S., Van Pamel, A. & Zhukov, L., 2024. Graph-based retriever captures the long tail of biomedical knowledge. *arXiv preprint arXiv:2402.12352*. Available at: https://doi.org/10.48550/arXiv.2402.12352 [Accessed 1 August 2024].

Ji, S., Pan, S., Cambria, E., Marttinen, P., & Yu, P. S. (2022) 'A Survey on Knowledge Graphs: Representation, Acquisition, and Applications', IEEE Transactions on Neural Networks and Learning Systems, 33(2), pp. 494-514.

Bruckhaus, T., 2024. RAG does not work for enterprises. *arXiv preprint arXiv:2406.04369*. Available at: https://doi.org/10.48550/arXiv.2406.04369 [Accessed 1 August 2024].

Guu, K., Lee, K., Tung, Z., Pasupat, P., & Chang, M. W. (2020) 'REALM: Retrieval-Augmented Language Model Pre-Training', *Proceedings of the 37th International Conference on Machine Learning (ICML)*, pp. 3929-3938.

Khandelwal, U., Fan, A., Jurafsky, D., & Zettlemoyer, L. (2020) 'Generalization through Memorization: Nearest Neighbor Language Models', *Proceedings of the 8th International Conference on Learning Representations (ICLR)*.

Llama Index, 2024. Stages within RAG. Available at: https://docs.llamaindex.ai/en/stable/getting_started/concepts/ [Accessed 2 August 2024].

Karpukhin, V., Oguz, B., Min, S., Lewis, P., Wu, L., Edunov, S., ... & Yih, W.T. (2020). Dense passage retrieval for open-domain question answering. arXiv preprint arXiv:2004.04906. https://arxiv.org/abs/2004.04906

Hogan, A., Blomqvist, E., Cochez, M., d'Amato, C., de Melo, G., Gutiérrez, C., & Janowicz, K. (2021) 'Knowledge Graphs', *Synthesis Lectures on Data, Semantics, and Knowledge*, 12(2), pp. 1-260.

Llama Index, 2024. Metadata replacement + node sentence window. Available at: https://docs.llamaindex.ai/en/stable/examples/node_postprocessor/MetadataReplacementDemo/ [Accessed 2 August 2024].

S. Yang. Advanced rag 01: Small to big retrieval. https://towardsdatascience.com/advanced-rag-01-small-to-big-retrieval-172181b396d4, 2023. [Accessed 2 August 2024].

Reimers, N. & Gurevych, I. (2019) 'Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks', *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing (EMNLP-IJCNLP)*, pp. 3973-3983.

J. Liu, "LlamaIndex," 11 2022. [Online]. Available: https://github. com/jerryjliu/llama index

Pandit, V., 2024. Retrieval augmented generation (RAG) with knowledge graphs. Available at: https://vivekpandit.medium.com/knowledge-graphs-enhancing-retrieval-augmented-generation-rag-db8878e8a2b9 [Accessed 2 August 2024].

Chen, R., Zhang, X., Wu, J., Fan, W., Wei, X.-Y. & Li, Q., 2024. Multi-level querying using a knowledge pyramid. arXiv preprint arXiv:2407.21276. Available at: https://doi.org/10.48550/arXiv.2407.21276 [Accessed 2 August 2024].

Eibich, M., Nagpal, S. & Fred-Ojala, A., 2024. ARAGOG: Advanced RAG output grading. *arXiv preprint arXiv:2404.01037*, p. 5. Available at: https://doi.org/10.48550/arXiv.2404.01037 [Accessed 4 August 2024].

James Calam. Ai arxiv dataset. https://huggingface.co/datasets/jamescalam/ai-arxiv, 2023. Accessed: 2024-03-24.

Aman.ai (2024) 'Retrieval-Augmented Generation (RAG)', Aman AI, available at: https://aman.ai/primers/ai/RAG/ [Accessed 3 August 2024].

Chadha, A. & Jain, V., 2020. Retrieval augmented generation. Available at: https://aman.ai/primers/ai/RAG/ [Accessed 3 August 2024].

DeepLearning.AI (2024) 'Building and Evaluating Advanced RAG', DeepLearning.AI, available at: https://learn.deeplearning.ai/courses/building-evaluating-advanced-rag/lesson/2/advanced-rag-pipeline [Accessed 3 August 2024].

Benveniste, D., 2023. [LinkedIn post]. Available at: https://www.linkedin.com/posts/damienbenveniste_machinelearning-datascience-artificialintelligence-activity-7119708674868051969-5HA1/?utm_source=share&utm_medium=member_desktop [Accessed 4 August 2024].

Francis, P., Robinson, I., and Webber, J., 2018. Graph Databases: New Opportunities for Connected Data. 2nd ed. O'Reilly Media.

Robinson, I., Webber, J., and Eifrem, E., 2015. Graph Databases: New Opportunities for Connected Data. 2nd ed. O'Reilly Media.

Roller, S., Dinan, E., Goyal, N., Ju, D., Williamson, M., Liu, Y., Xu, J., Ott, M., Smith, E.M., Boureau, Y.L. and Weston, J., 2021. Recipes for building an open-domain chatbot. In Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume (pp. 300-325).

Zhang, Z., Liu, Y., Lin, H., Wang, M., Wu, F., Yu, S. and Chen, X., 2022. Knowledge-graph augmented language model pre-training. In 2022 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP) (pp. 8332-8336). IEEE.

Manning, C.D., Raghavan, P. and Schütze, H., 2008. Introduction to information retrieval. Cambridge university press.

Rashkin, H., Celikyilmaz, A., Choi, Y. and Gao, J., 2021. PlotMachines: Outline-conditioned generation with dynamic plot state tracking. In Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (pp. 4274-4295).

Xu, H., Liu, W., Pu, P. and Chua, T.S., 2022. KG-FiD: Infusing Knowledge Graph in Fusion-in-Decoder for Open-Domain Question Answering. In Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers) (pp. 4932-4945).

Facebook AI Research, 2024. *FAISS: A library for efficient similarity search and clustering of dense vectors*. Available at: https://github.com/facebookresearch/faiss [Accessed 25 August 2024].

Jegou, H., Douze, M., and Schmid, C., 2011. Product quantization for nearest neighbor search. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 33(1), pp.117-128.

Guo, R., Sun, X., Yu, H., and Zhou, Y., 2020. Accelerating large-scale similarity search using dynamic inverted indexing. *Proceedings of the 29th ACM International Conference on Information & Knowledge Management (CIKM)*, pp.2161-2164.

Elastic, n.d. What is vector embedding? Available at: https://www.elastic.co/what-is/vector-embedding [Accessed 5 August 2024].

Rojo-Echeburúa, A., 2024. Using a knowledge graph to implement a RAG application. DataCamp. Available at: https://www.datacamp.com/tutorial/knowledge-graph-rag [Accessed 5 August 2024].

Narayana, L.U., 2024. Implementing Neo4j knowledge graphs with LlamaIndex: A guide using Indian spiritual texts. Stackademic. Available at: https://blog.stackademic.com/implementing-neo4j-knowledge-graphs-with-llamaindex-a-guide-using-indian-spiritual-texts-9e5860e15c65 [Accessed 5 August 2024].

OpenAI, 2024. GPT-4o Mini: Advancing cost-efficient intelligence. Available at: https://openai.com/index/gpt-4o-mini-advancing-cost-efficient-intelligence/ [Accessed 5 August 2024].

OpenAI, n.d. Guides: Embeddings. Available at: https://platform.openai.com/docs/guides/embeddings [Accessed 6 August 2024].

Llama Index, n.d. Loading documents and nodes. Available at: https://docs.llamaindex.ai/en/stable/module_guides/loading/documents_and_nodes/ [Accessed 16 August 2024].

Llama Index, n.d. Metadata Replacement Demo. Available at: https://docs.llamaindex.ai/en/stable/examples/node_postprocessor/MetadataReplacementDemo/ [Accessed 16 August 2024].

Llama Index, n.d. Auto Merging Retriever example. Available at: https://docs.llamaindex.ai/en/stable/examples/retrievers/auto_merging_retriever/ [Accessed 20 August 2024].

Mayerhofer, N., 2024. Construct knowledge graphs from unstructured text. Neo4j Developer Blog. Available at: https://neo4j.com/developer-blog/construct-knowledge-graphs-unstructured-

text/#:~:text=We%20take%20the%20simplest%20possible,edges%20from%20the%20input%20text [Accessed 19 August 2024].

Tonic, 2024. About RAG metrics. Available at: https://docs.tonic.ai/validate/about-rag-metrics/tonic-validate-rag-metrics-how-to-use [Accessed 30 August 2024].

TonicAI, 2024. Tonic Validate README. Available at: https://github.com/TonicAI/tonic_validate?tab=readme-ov-file [Accessed 30 August 2024].