**Enhancing the results generated from Large Language Models using Retrieval Augmented Generation**

Rudranil Gupta

M.S in Artificial Intelligence and Machine Learning

Liverpool John Moores University

Final Thesis Report

JULY 2024

# TABLE OF CONTENTS

DEDICATION v

ACKNOWLEDGEMENTS vi

ABSTRACT vii

LIST OF TABLES xii

LIST OF FIGURES xv

LIST OF ABBREVIATIONS xvi

CHAPTER 1: INTRODUCTION 1

1.1 Background of the Study 1

1.2 Problem Statement 3

1.3 Aim and Objectives 5

1.4 Research Questions 3

1.5 Scope of the Study 6

1.6 Significance of the Study 6

1.7 Structure of the Study 8

CHAPTER 2: LITERATURE REVIEW 11

2.1 Introduction

2.2 Complete overview of RAG Architecture

2.2.1 Query Construction

2.2.2 Query Translation

2.2.2.1 Text-to-SQL

2.2.2.2 Text-to-Cypher

2.2.2.3 Self Query Retriever

2.2.2.4 Multi-query Retriever

2.2.2.5 Query Decomposition

2.2.2.6 Stepback Prompting

2.2.2.7 Hyde

2.2.3 Query Routing

2.2.4 Indexing

2.2.4.1 Chunk Optimization

2.2.4.2 Multi-representation Indexing

2.2.4.3 Specialized Embeddings

2.2.4.4 Hierarchical Retrievals

2.2.5 Retrieval

2.2.6 Generation

2.3 Related Research Publications

2.4 Summary

CHAPTER 3: RESEARCH METHODOLOGY 30

3.1 Introduction

3.2 RAG Framework

3.2.1 Naive RAG

3.2.2 Advanced RAG

3.3 Advanced RAG Techniques

3.3.1 MultiQuery Retriever

3.3.2 Contextual Compression Retrieval Technique

3.3.3 Sentence Window Retrieval

3.3.4 Corrective Retrieval-Augmented Generation (CRAG)

3.4 RAG Evaluation

3.4.1 Evaluation Focus Areas

3.4.2 Retrieval Quality Assessment

3.4.3 Generation Quality Assessment

3.5 Evaluation of RAG (Retrieval-Augmented Generation ) using RAGAs - On Human annotated and Synthetic Test Data Generation

3.6 Summary

CHAPTER 4: ANALYSIS AND EXPERIMENTS 50

4.1 Introduction

4.2 Data ingestion

4.2.1 Process of Data Extraction

4.2.2 Creation of question-answers for fine tuning

4.3 Chunking, embedding and storing the chunks in a vector database

4.4 The Retrieval Augmented Generation Process

4.4.1 Creation of RAG Pipelines

4.4.2 Initializing various retrievers

4.4.3 Instantiating Reranking Model

4.4.4 Execution Of Multiple Retriever Chains

4.4.5 Evaluation and Comparison of Results using RAGAs

4.5 Sentence Window Retrieval Process

4.5 Finetuning llama2 model

4.6 Summary

CHAPTER 5: RESULTS AND DISCUSSIONS 114

5.1 Introduction

5.2 Metrics Comparison and analysis between Baseline and Advanced RAG Techniques

5.3 Baseline RAG(Vanilla RAG) vs MultiQuery Retriever

5.4 Baseline RAG(Vanilla RAG) vs Contextual Compressor Retriever

5.5 Baseline RAG(Vanilla RAG) vs Sentence Window Retriever

5.6 Fine-tuning versus RAG in Generative AI Applications Architecture

5.7 Fine-Tuning Llama 2 with QLoRA for Enhanced Efficiency

5.8 Sentence Window Retriever vs Fine tuned LLama2

5.9 Multi Query Performance vs FineTunedOnLlama2

5.10 ContextualCompressor Performance vs FineTunedOnLlama2

5.11 Baseline Performance vs FineTunedOnLlama2

5.12 Summary

CHAPTER 6: CONCLUSIONS AND RECOMMENDATIONS 151

6.1 Introduction 151

6.2 Discussion and Conclusion 151

6.3 Contribution to knowledge 153

6.4 Future Recommendations 153

REFERENCES 154

APPENDIX A: RESEARCH PROPOSAL 167

APPENDIX B: PYTHON CODE FOR RAG EVALUATION

**DEDICATION**

I would like to dedicate this research work to my family and to the research community who has made tremendous contribution in the field of Natural Language Processing and Large Language Models.

**ACKNOWLEDGEMENT**

I would like to thank my Research Supervisor Ankan Dutta for accepting me to be his student. He provided me with directions to proceed with and valuable inputs and feedback for making this research a success.

I would like to acknowledge learning platform upGrad and their staff for making this learning experience comfortable. I would like to express my heartfelt thanks to Dr. Ahmad Kaky from LJMU for his valuable sessions and timely guidelines.

I would like to express my sincere gratitude to my batch mates who have always supported me through their encouragement and suggestions.

I would like to extend my gratitude to my parents and God Almighty.

**ABSTRACT**

Retrieval Augmented Generation (RAG) is a technique in natural language processing that enhances generative AI models' accuracy and reliability by incorporating facts fetched from external sources like the internet, databases, open-domain datasets, or private information of individuals/organizations. Despite significant advancements, RAG faces challenges such as retrieval efficiency, context awareness, and relevance of generated answers to the question asked. In this study, we compared the performance of Baseline RAG or vanilla RAG and advanced RAG architectures like MultiQuery Retrieval, Contextual Compression Retrieval, and Sentence Window Retrieval. Additionally, we fine-tuned an LLM model (Llama2) and analyzed whether there is a change in performance across all the RAG architectures, including Baseline RAG and the advanced RAG architectures mentioned. To measure performance, we employed the RAGAs framework and used metrics such as Faithfulness, Answer Relevancy, Context Recall, Context Precision, and Answer Correctness. Our findings indicate that while the MultiQuery Retriever model showed a 5.39% improvement in faithfulness, it exhibited significant declines in context recall (86.18%) and context precision (88.59%). Similarly, Contextual Compression Retrieval improved faithfulness by 4.08% but saw a drastic reduction in context precision (89.24%) and answer correctness (30.20%). The Sentence Window Retriever model, although showing a 2.74% increase in answer relevancy, suffered from an 81.82% drop in context recall and a 34.45% decrease in answer correctness. Fine-tuning with Llama2 generally resulted in trade-offs; while some models demonstrated minor improvements in answer relevancy, they often incurred substantial reductions in faithfulness and correctness, particularly with a 48.82% decline in answer correctness in the fine-tuned baseline model. These comparisons highlight the nuanced effects of different RAG architectures and fine-tuning processes, emphasizing the importance of balancing various performance metrics to guide future improvements in RAG model development.

**LIST OF FIGURES**

Figure 1: A pre-trained retriever (Query Encoder + Document Index) is combined with a pre-trained sequence-to-sequence model (Generator), and they are fine-tuned end-to-end. The top-K documents, denoted as 'zi', are identified using Maximum Inner Product Search for a given query 'x'. For the final prediction 'y', 'z' is treated as a latent variable, and marginalization is performed over the predictions of the sequence-to-sequence model, considering the variations presented by different documents.

Figure 2: Retrieval Augmented Generation architecture

Figure 3: Gantt chart showing the research plan

**LIST OF ABBREVIATIONS**

LLM Large Language Models

NLP Natural Language Processing

RAG Retrieval Augmented Generation

BART Bidirectional and Auto-Regressive Transformers

DPR Dense Passage Retrieval

REALM Retrieval-Augmented Language Model

kNN-LM k-Nearest Neighbour Language Model

FiD Fusion-in-Decoder

ICT Information Retrieval with Contrastive Learning

BERT Bidirectional Encoder Representations from Transformers

RoBERTa Robustly Optimized BERT Approach

ALBERT A Lite BERT

ColBERT Columnar BERT

GPT-4 Generative Pre-trained Transformer 4

LLaMA Large Language Model Meta AI

API Application Programming Interface

AWS Amazon Web Services

FAISS Facebook AI Similarity Search

MLOps Machine Learning Operations

BLEU Bilingual Evaluation Understudy

ROUGE Recall-Oriented Understudy for Gisting Evaluation

METEOR Metric for Evaluation of Translation with Explicit ORdering

NLTK Natural Language Toolkit

**CHAPTER 1**

**INTRODUCTION**

**1.1 Background**

Despite the impressive capabilities of LLMs in generating human-quality text and performing a variety of NLP tasks, they often lack up-to-date information or specific domain knowledge. RAG emerged as a solution to bridge this gap, providing LLMs with access to external knowledge sources for more accurate and reliable responses. The evolution of RAG can be traced back to early information retrieval systems and question-answering systems developed in the 1970s. These systems used NLP to access text in specific domains, laying the groundwork for the text mining concepts that RAG builds upon.

The seminal paper (Lewis et al., 2020) that coined the term "RAG" was developed by Patrick Lewis and colleagues during his doctorate in NLP at University College London and while working for Meta at a new London AI lab. This work was inspired by a vision to integrate a retrieval index into a trained system, allowing it to learn and generate Contextually Rich Responses. The results showed how to make generative AI models more authoritative and trustworthy, leading to the widespread adoption and further development of RAG concepts.

RAG consists of three parts - a retriever, a generator and an indexed knowledge base of text documents like Wikipedia or CommonCrawl.

A retriever's primary role is to search for and retrieve relevant information from a vast array of external sources or databases. These sources could be anything from text documents, scientific papers, and websites to structured databases.

A generator is responsible for producing the final response or output that is presented to the user. It takes the information retrieved by the retriever and integrates it into the generation process. Once the retriever provides the relevant information, the generator analyses this data along with the original query. It then synthesizes this information, often combining it with its pre-existing knowledge (gained during its training phase) to create a comprehensive and contextually relevant response. The generator is usually a large language model capable of understanding and generating human-like text

RAG models combine parametric memory, which refers to the knowledge that is stored in the parameters (weights) of a neural network, specifically in the generative model part of the RAG model (like BART) and non-parametric memory in RAG models refers to the knowledge that is stored outside the model parameters, specifically in the external documents or databases that the model queries during the retrieval phase.

The retrieval model, typically a Dense Passage Retriever (DPR), functions as a bi-encoder architecture. It encodes both the input query and documents into vector representations. The likelihood of retrieving a document zgiven an input xis calculated based on the dot product of their vector representations. The DPR is a fine-tuned BERT model, which computes the document and query representations.

Where η is a parameter that returns top-k distributions over text passages, d(z) is a dense representation of a document produced by a BERTBASE *document encoder*, and q(x) a query representation produced by a *query encoder*, also based on BERTBASE.

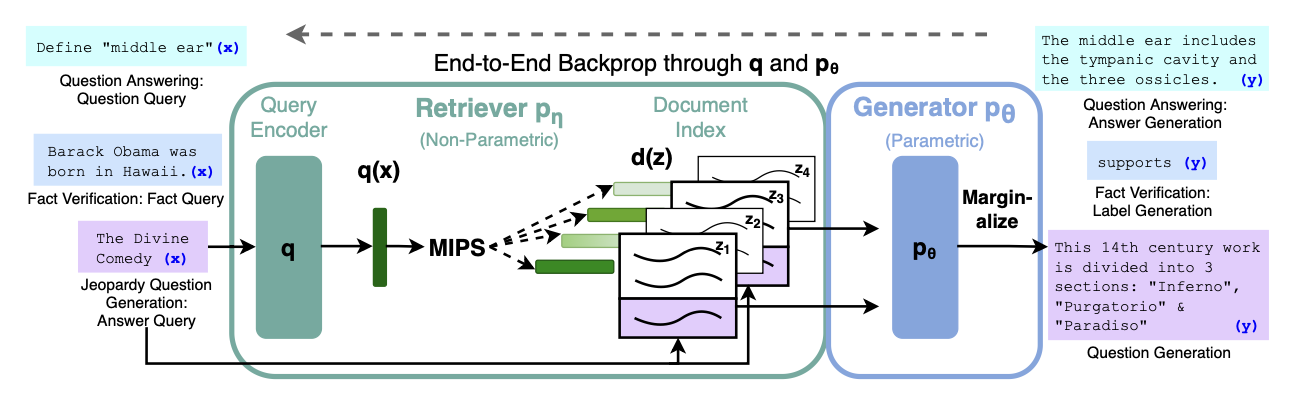


Figure 1: A pre-trained retriever (Query Encoder + Document Index) is combined with a pre-trained sequence-to-sequence model (Generator), and they are fine-tuned end-to-end. The top-K documents, denoted as 'zi', are identified using Maximum Inner Product Search for a given query 'x'. For the final prediction 'y', 'z' is treated as a latent variable, and marginalization is performed over the predictions of the sequence-to-sequence model, considering the variations presented by different documents. (Lewis et al., 2020)

The generator model can be any transformer model having encoder-decoder architecture with probability pθ(yi|x, z, y1:i−1) [parameterized by θ] of generating an output sequence y given the input sequence x, a current token based on a context of the previous i − 1 tokens y1:i−1 and the retrieved documents z.

The two types of RAG models are:

**RAG-Sequence Model:** The RAG-Sequence model utilizes a single document to generate the entire sequence. It views the chosen document as a crucial variable that helps calculate the probability of the sequence given the input (p(y|x)) through a top-K approximation method. Specifically, it first retrieves the top K documents. Then, for each of these documents, the model calculates the probability of the output sequence. These probabilities are combined (marginalized) to form the final sequence probability.

**RAG-Token Model:** Contrastingly, the RAG-Token model has the flexibility to select a different document for each token in the target sequence, allowing it to integrate information from multiple documents when formulating an answer. In practice, it retrieves the top K documents and then, for each token in the output sequence, the model predicts a distribution across these documents, which is then aggregated. This process repeats for each subsequent output token.

Both RAG Sequence and RAG Token models are trained by minimizing the negative log likelihood Σj − log p(yj |xj ) of generating the correct output sequences given the input sequences. The training process involves learning to retrieve the most relevant documents and use them effectively to generate accurate outputs.

To decode the output in the RAG-Token model, a standard beam decoder is employed. For the RAG-Sequence model, beam search is executed for each document *z*, with each potential answer (hypothesis) being evaluated using the probability function . This process generates a collection of potential answers, *Y*, among which certain answers might not be present in the beams of all documents. To calculate the probability of a specific hypothesis *y*, an extra forward pass is conducted for each document *z* that didn't initially include *y* in its beam. In this step, the probability given by the generator is combined with, and then these probabilities are aggregated across all beams to compute the marginal probabilities.

**1.2 Problem Statement:**

The existing challenges in Retrieval-Augmented Generation (RAG) are diverse and impact various aspects of its implementation and effectiveness. According to the paper titled ‘**A Survey on Retrieval-Augmented Text Generation**’ (Li et al., 2022), the current challenges are:

* **Retrieval Sensitivity**: The performance of RAG is highly sensitive to the quality of the retrieved information. Models can underperform when the retrieved examples are less like the query.
* **Retrieval Efficiency**: Expanding the retrieval memory for better accuracy can lead to reduced efficiency due to the increased computational overhead.
* **Local vs. Global Optimization**: The gap between retrieval metric optimization during training and inference poses challenges. This discrepancy affects the overall effectiveness of the model.
* **Diverse & Controllable Retrieval**: Current models often rely on universal metrics for retrieval, such as lexical similarities. There's a need for more customized retrieval metrics to enable more controlled and diverse text generation.
* **Multi-Modal Data Handling**: With advancements in multi-modal data retrieval, there's a need to efficiently integrate and process data from different modalities, like images and text, for enhanced RAG applications.
* **Scalability and Domain Adaptation**: Scaling RAG models to different domains and ensuring they adapt well to specific domain requirements remains a challenge.
* **Data Privacy and Security**: Ensuring the privacy and security of data used in retrieval processes, especially in sensitive applications.
* **Ethical Considerations and Bias**: Addressing potential biases in retrieved data and the ethical implications of using certain types of data for generation.

These challenges highlight the complex nature of RAG and the need for continued research and development to address these issues.

**1.3 Aim and Objectives:**

The main aim of this research is to propose the best RAG model after studying the performance of RAG across three distinct architectural approaches:

1. Using baseline RAG architecture
2. Using various Advanced RAG architectures
3. Fine-tuning the generation models to gain a deeper understanding of context

In our study, we will employ the advanced models outlined in the preceding section. A data pipeline based on the RAG framework will be constructed and fed with information from various external data sources for retrieval purposes. We will then evaluate the accuracy of the outcomes to determine if they align with the given context.

The research objectives formulated based on the aim of the study are as follows:

1. To enhance the quality and relevance of the input data by cleaning, normalizing, and structuring it in a way that maximizes the RAG model's ability to understand, interpret, and generate accurate and contextually appropriate responses.

3. To diversify training datasets and implement bias-correcting algorithms, aiming to mitigate biases during the retrieval phase.

4. To evaluate the precision and reliability of its outputs, ensuring that the generated responses are not only relevant and contextually appropriate but also grounded in accurate information derived from reliable external sources.

5. To evaluate the three different model architectures for implementing Retrieval Augmented Generation to determine which yields the highest accuracy.

6. To incorporate continuous learning and updating features in the RAG models, ensuring adaptability and relevance to evolving data landscapes.

7. To create and integrate feedback loops where the RAG system’s outputs are continually evaluated and used for ongoing system refinement, aiming for greater accuracy, reliability, and efficiency.

**1.4 Significance of the Study**

The contributions of this paper are multifaceted:

1. A detailed evaluation framework for RAG is established, highlighting the objectives and metrics for assessment. Through comparative analysis, the paper elucidates RAG's strengths and weaknesses relative to fine-tuning from different perspectives.
2. A comprehensive analysis and comparison between the Baseline RAG and various Advanced RAG techniques using RAGAs Framework
3. Comparison between Fine Tuning and RAG and their performances on different metrics.

**5. Scope of the Study**

Since the research is time-bound, we set out certain criteria boundaries for successful completion of our study.

Instead of using the complete dataset, we will take a sample of our data to train the RAG models which we will build.   
Due to resource and computation constraints, we will be using LLMs having 1 billion to 7 billion parameters instead of LLMs that have huge number of parameters like 34 billion or 70 billion. We shall also utilize the API keys for OpenAI models like GPT-3.5 and GPT-4 to fine tune our models.

Instead of hosting our models in cloud provider services like AWS and MS Azure, we will be using Google Colab and Kaggle notebooks.

We will be using vector databases like Pinecone, FAISS and Chromadb instead of using costlier alternatives used in production settings.

For deployment purposes, we shall use Streamlit and Flask.

**CHAPTER 2**

**LITERATURE REVIEW**

**2.1 Introduction**

Over the past few years, artificial intelligence and natural language processing have seen significant advancements, leading to the development of powerful large language models (LLMs) such as the Generative Pre-trained Transformer(GPT). The technology driving LLMs, including advanced deep learning techniques, large-scale transformers, and vast amounts of data, have propelled their rapid evolution. Models like GPT-4 and Llama 2 have demonstrated exceptional performance across numerous tasks and domains, often without specific prompts. These models surpass their predecessors and hold immense potential in various fields like coding, medicine, law, agriculture, and psychology, closely approaching human-level expertise. As LLM research continues, it is critical to identify their limitations and address the challenges of developing more comprehensive artificial general intelligence (AGI) systems. Large Language Models (LLMs) demonstrate significant capabilities but face challenges such as hallucination, outdated knowledge, and non- transparent, untraceable reasoning processes. Retrieval-Augmented Generation (RAG) has emerged as a promising solution by incorporating knowledge from external databases. This enhances the accuracy and credibility of the models, particularly for knowledge-intensive tasks, and allows for continuous knowledge updates and integration of domain-specific information. RAG synergistically merges LLMs’ intrinsic knowledge with the vast, dynamic repositories of external databases. This comprehensive review paper offers a detailed examination of the progression of RAG paradigms, encompassing the Naive RAG, the Advanced RAG, and the Modular RAG. It meticulously scrutinises the tripartite foundation of RAG frameworks, which includes the retrieval , the generation and the augmentation techniques. The paper highlights the state-of-the-art technologies embedded in each of these critical components, providing a profound understanding of the advancements in RAG systems. Furthermore, this paper introduces the metrics and benchmarks for assessing RAG models, along with the most up-to-date evaluation framework. In conclusion, the paper delineates prospective avenues for research, including the identification of challenges, the expansion of multi-modalities, and the progression of the RAG infrastructure and its ecosystem.

Large language models (LLMs) such as the GPT series and the LLama series , along with other models like Gemini, have achieved remarkable success in natural language processing, demonstrating supe-Large Language Models (LLMs) have demonstrated superior performance on various benchmarks, including SuperGLUE, MMLU, and BIG-bench. However, despite these advancements, LLMs face significant challenges, particularly when dealing with specialised or domain-specific queries. A notable problem is the generation of inaccurate content, also known as "hallucinations," particularly when queries go beyond the training data or require current information. This highlights the limitations of using LLMs in practical settings without proper checks. To address these issues, one effective strategy is the use of Retrieval-Augmented Generation (RAG), which combines external data retrieval with the generation process to improve the accuracy and relevance of the model's outputs.

Introduced by Lewis et al. in 2020, Retrieval-Augmented Generation (RAG) represents a significant advancement in the field of Large Language Models (LLMs), aimed at improving generative tasks. RAG incorporates a crucial initial step where LLMs access an external dataset to fetch pertinent information before engaging in text generation or answering queries. This methodology not only guides the subsequent generation process but also ensures that the outputs are substantiated by the retrieved data, considerably boosting the output's accuracy and relevance. By dynamically sourcing information from knowledge repositories during the inference stage, RAG effectively tackles the issue of generating factually incorrect content, often termed as "hallucinations." The adoption of RAG in enhancing LLMs has been swift, establishing it as a key technology in refining chatbot functionalities and making LLMs more applicable in real-world scenarios.

The field of Retrieval-Augmented Generation (RAG) has experienced rapid growth, yet it lacks a systematic framework that consolidates and abstracts its developments, presenting challenges in grasping the full spectrum of advancements within RAG. This survey aims to provide a comprehensive outline of the RAG process, capturing the current trends and future directions in RAG research through an in-depth exploration of retrieval augmentation within Large Language Models (LLMs).

This paper endeavours to succinctly summarise and categorise the fundamental principles, historical evolution, thematic content, and particularly the pertinent methodologies and applications post the advent of LLMs. It also aims to detail the evaluation strategies and practical use cases of RAG. The goal is to offer an extensive review and analysis of prevailing RAG technologies, drawing conclusions and envisioning future methodological advancements. The survey aspires to equip readers and practitioners with a deep and organised understanding of large-scale models and RAG, clarify the progression and principal technologies in retrieval augmentation, highlight the advantages and limitations of different techniques and their appropriate applications, and speculate on prospective future developments.

|  |  |
| --- | --- |
| **2.2** | **Complete overview of RAG Architecture** |

Latest RAG architecture consists of the given components: Query Construction, Query Translation, Routing, Indexing, Retrieval and Generation.

**2.2.1 Query Construction**

Query construction in Retrieval Augmented Generation (RAG) involves generating appropriate database queries based on natural language inputs to retrieve relevant information from various types of databases. This process is essential for effectively integrating structured and unstructured data sources in RAG systems.

1. **Relational Databases (RDBs)**: In RAG systems using relational databases, queries are constructed in SQL. This involves translating the natural language input into SQL queries that can retrieve data from tables based on predefined relationships and schema. The structured nature of relational databases makes SQL a powerful tool for precise data retrieval through SELECT statements, joins, and other SQL operations​​.
2. **Graph Databases (GDBs)**: For graph databases, queries are constructed using languages like Cypher. Graph databases excel in representing and querying complex relationships between entities, making them ideal for scenarios where the connections between data points are as important as the data itself. The query construction in this context focuses on navigating and exploiting these relationships to retrieve relevant information​.
3. **Vector Databases (VDBs)**: Vector databases store and query data based on vector embeddings, which are numerical representations of text or other unstructured data. In a RAG context, query construction for vector databases involves transforming natural language queries into vector space searches to find contextually similar data points. This method is particularly useful for semantic searches and handling large volumes of unstructured data​.

The overall process of query construction in RAG involves:

* **Understanding the user query**: Analyzing the natural language input to determine the information need.
* **Routing**: Deciding which database(s) to query based on the nature of the request and the type of data stored.
* **Generating appropriate queries**: Formulating the queries in the specific languages (SQL for RDBs, Cypher for GDBs, vector operations for VDBs) to retrieve relevant data.

This multi-step process ensures that the RAG system can effectively leverage different types of databases to provide comprehensive and accurate responses​.

A diagram of a software company

Description automatically generated with medium confidence  
  
  
  
  
**2.2.2 Query Translation**  
  
Query translation in Retrieval-Augmented Generation (RAG) systems is a process designed to improve the retrieval and generation of relevant information in response to user queries. This involves several techniques, including rephrasing the original query into multiple forms (multi-query translation), breaking down complex queries into simpler sub-queries (decomposition), and abstracting the query to a more general level (step-back prompting). Each of these methods enhances the retrieval accuracy and contextual relevance of the generated responses. For example, multi-query translation creates several variations of a user’s query to cover different aspects, thereby increasing the chances of retrieving pertinent documents​. Similarly, decomposition helps in handling complex queries by tackling simpler sub-queries, facilitating more detailed and accurate responses​​. These approaches ensure that RAG systems can provide precise, context-rich answers by leveraging a comprehensive retrieval process​​. Now let’s look at few approaches for query translation.  
  
**2.2.2.1 Text-to-SQL**  
  
Text-to-SQL in user query understanding involves translating natural language queries into structured SQL (Structured Query Language) commands that a database can execute. This process is essential for enabling non-technical users to interact with databases using everyday language rather than requiring them to learn complex SQL syntax.   
  
**Process of Text-to-SQL in User Query Understanding**

1. **Natural Language Understanding (NLU)**:
   * **Intent Recognition**: Identifying the user's intent from the query. For example, a user query like "Show me the total sales for last month" has the intent of retrieving sales data.
   * **Entity Recognition**: Extracting specific entities from the query such as "total sales" and "last month."
2. **Schema Linking**:
   * **Mapping Entities to Database Schema**: Linking the identified entities and intent to the corresponding elements in the database schema. For example, "total sales" might map to a column named total\_sales in a sales table.
3. **Query Formation**:
   * **Generating SQL Query**: Formulating a SQL query based on the mapped entities and intent. For the example query, this could translate to SELECT SUM(total\_sales) FROM sales WHERE date BETWEEN '2023-06-01' AND '2023-06-30'.
4. **Execution and Retrieval**:
   * **Running the SQL Query**: The generated SQL query is executed against the database to retrieve the desired information.
   * **Presenting Results**: The results are then formatted and presented to the user in a comprehensible manner.

**2.2.2.2 Text-to-Cypher**  
  
Text-to-Cypher in user query understanding involves translating natural language queries into Cypher, the query language used for interacting with graph databases such as Neo4j. This process allows users to interact with graph databases without needing to know the complex syntax of Cypher.   
  
**Process of Text-to-Cypher in User Query Understanding:**

1. **Natural Language Understanding (NLU)**:
   * **Intent Recognition**: Identifying what the user wants to achieve with their query. For example, "Find all friends of John who live in New York" implies the user wants to retrieve information about John's friends residing in New York.
   * **Entity Recognition**: Extracting specific entities and attributes such as "John," "friends," and "New York."
2. **Graph Schema Mapping**:
   * **Mapping Entities to Graph Schema**: Linking the identified entities to the appropriate nodes and relationships in the graph database schema. For example, mapping "John" to a node labeled Person, "friends" to a FRIEND relationship, and "New York" to a property of the Location node.
3. **Query Generation**:
   * **Formulating Cypher Query**: Creating a Cypher query that reflects the user’s intent. The query for the above example might be   
     MATCH (john:Person {name: "John"})-[:FRIEND]->(friend:Person)-[:LIVES\_IN]->(location:Location {city: "New York"}) RETURN friend.
4. **Execution and Result Presentation**:
   * **Running the Cypher Query**: Executing the generated Cypher query against the graph database.
   * **Presenting Results**: Displaying the query results in a user-friendly format.

**2.2.2.3 Self Query Retriever**

A self-querying retriever is one that can query itself. Specifically, it uses a query-constructing LLM chain to create a structured query, which it then applies to its underlying VectorStore. [This allows the retriever not only to compare user-input queries with stored document contents but also to extract and execute filters based on metadata from the user query](https://python.langchain.com/v0.1/docs/modules/data_connection/retrievers/self_query/).  
Let’s consider a scenario where we have a collection of documents related to programming languages. Our self-querying retriever can construct queries based on user input. For instance, if the user asks, “Which programming languages are statically typed?” the retriever can generate a structured query like this:

Query:"programming languages" AND "statically typed"Query:"programming languages" AND "statically typed"

The retriever then applies this query to the document collection, retrieving relevant documents that match both criteria. In this case, it would return information about statically typed programming languages.

**2.2.2.4 Multi-query Retriever**  
  
Distance-based vector database retrieval involves embedding queries into a high-dimensional space and identifying similar documents based on their proximity. However, slight variations in query wording or inadequate semantic capture by embeddings can yield disparate results. Manual engineering or tuning is often necessary to mitigate these issues, though it can be labor-intensive.

The MultiQueryRetriever automates prompt tuning using a language model to generate diverse queries from different angles for a given user query. Each query retrieves relevant documents, and their unique combination across all queries expands the pool of potentially pertinent documents. By offering multiple perspectives on the same query, the MultiQueryRetriever aims to enhance results beyond the limitations of distance-based retrieval, achieving a more comprehensive outcome.

**2.2.2.5 Query Decomposition**

In the context of Retrieval-Augmented Generation (RAG), query decomposition plays a crucial role in enhancing model performance. Let’s delve into it:

**Query Decomposition** involves breaking down complex queries into smaller sub-questions, each of which can be addressed independently. This approach simplifies the retrieval process and enables detailed responses. Specifically, when dealing with a multifaceted query, RAG decomposes it into narrower retrieval challenges. Here’s how it works:

1. **Input Query**: Suppose we have an input query like: *“Is it legal for a licensed child driving a Mercedes-Benz to be employed in the US?”*
2. **Decomposition**: The query is deconstructed into sub-questions, such as:
   * *“Is it legal for a licensed child to drive a Mercedes-Benz?”*
   * *“Can a licensed child be employed in the US?”*
3. **Sub-Question Retrieval**: Each sub-question is treated as an independent retrieval task. The RAG system retrieves relevant documents for each sub-question.
4. **Fusion**: The retrieved documents from all sub-questions are combined (usually using a union operation), and the RAG model ranks them to determine the most relevant information.

By breaking down complex queries, RAG ensures that the model can handle diverse input scenarios effectively.

**2.2.2.6 Stepback Prompting**When faced with a specific query, step-back prompting involves using the Large Language Model (LLM) to generate a more general query. This broader query aims to retrieve high-level context or information related to the original query. Essentially, step-back prompting allows us to ground the answer by considering both the specific and general contexts.

During retrieval, the system performs two steps:1. Retrieves information for the original specific query.2. Retrieves information for the more general query generated through step-back prompting.  
  
Both the specific and general contexts are fed into the LLM during the final answer generation step. [This approach helps ensure that the response is well-informed and contextually relevant](https://chatgen.ai/blog/the-ultimate-guide-on-retrieval-strategies-rag-part-4/).  
[By abstracting specific queries into broader questions, step-back prompting facilitates the retrieval of a wider range of related information, ultimately improving the RAG system’s performance](https://blog.rebyte.ai/ragtechniques).

**2.2.2.7 Hyde**

**Hypothetical Document Embeddings (HyDE)** is a transformative solution in the context of **Retrieval-Augmented Generation (RAG)**. Let’s dive into the details:

1. **Concept**: HyDE generates hypothetical documents based on the user’s query. Instead of directly seeking in the vector database using the query and its computed vector, HyDE creates a theoretical document using a Language Learning Model (LLM), such as ChatGPT. This theoretical document is then transformed into an embedding vector.
2. **Unsupervised Encoder**: HyDE goes a step further by incorporating an unsupervised encoder learned through contrastive methods. This encoder refines the theoretical document’s embedding, making it more suitable for locating similar documents in the vector database.
3. **Answer-to-Answer Similarity**: Unlike traditional RAG approaches that focus on query-to-answer embedding similarity, HyDE emphasizes answer-to-answer embedding similarity. By doing so, it enhances retrieval performance across various tasks, including web search, question-answering, and fact verification.
4. **Advantages and Drawbacks**:  
     
   **Advantages**: HyDE’s performance is robust, matching well-tuned retrievers. It excels in scenarios where answers can be more open-ended, such as qualitative research, medical transcripts, and case studies.

[**Drawbacks**: HyDE may not consistently produce good results, especially when the subject is unfamiliar to the language model, potentially leading to incorrect information generation](https://medium.com/pondhouse-data/advanced-rag-improving-retrieval-augmented-generation-with-hypothetical-document-embeddings-hyde-db2b50ddb981).

[HyDE leverages hypothetical documents and answer embeddings to enhance retrieval, bridging the semantic gap between queries and relevant information in specialized domains](https://medium.com/pondhouse-data/advanced-rag-improving-retrieval-augmented-generation-with-hypothetical-document-embeddings-hyde-db2b50ddb981).

**2.2.3 Query Routing**

**In Retrieval-Augmented Generation (RAG), query routing** is a crucial step that determines the next course of action based on the user query. Let’s explore two key approaches:

1. **Logical Routing**:
   * In logical routing, the system intelligently routes user questions to the most relevant data sources or processes.
   * It involves predefined routes that guide decision-making. For example, if the user query pertains to summarization, logical routing directs the system to summarize relevant information.
   * [Logical routing creates non-deterministic chains, where the output of one step informs the next, leading to a synthesized response](https://pub.towardsai.net/advanced-rag-techniques-an-illustrated-overview-04d193d8fec6).
2. **Semantic Routing**:
   * Semantic routing leverages vector spaces to make decisions based on semantic meaning.
   * When a user query arrives, the system routes it to the most appropriate task-driven prompt or data source.
   * [By using semantic similarity, semantic routing ensures that the response aligns with the intended scope of the RAG system, avoiding answers outside its domain](https://bing.com/search?q=semantic+routing+in+RAG).

**2.2.4 Indexing**

In the context of RAG, indexing involves organizing and structuring the information within a knowledge base to facilitate quick and accurate retrieval of relevant content when needed. The process typically entails breaking down documents into smaller, manageable chunks and creating vector representations, or embeddings, of these chunks. These embeddings are then stored in a searchable format, often using specialized vector databases optimized for this purpose. When a query is made, the system can swiftly locate the most relevant information by comparing the query's embedding to those stored in the index. This similarity search enables the RAG system to pinpoint and retrieve the most pertinent information, which is then used to augment the generation process. The index can be continuously updated as new information is added to the knowledge base, ensuring that the RAG system remains current and effective. By enabling fast and accurate information retrieval, indexing directly impacts the quality and relevance of the generated responses, making it a crucial element in the overall performance of RAG systems.  
Now, let us discuss few of the components used in indexing in RAG.

**2.2.4.1 Chunk Optimization**

Chunk optimization in Retrieval-Augmented Generation (RAG) systems refers to the process of dividing documents or large bodies of text into smaller, more manageable pieces to improve the retrieval and generation processes. This technique is crucial for enhancing the overall performance and effectiveness of RAG systems.

The primary goal of chunk optimization is to create text segments that are large enough to contain meaningful context but small enough to be specific and relevant to particular queries. This balance is essential because overly large chunks may include irrelevant information, while excessively small chunks might lack necessary context.

Various strategies are employed for chunk optimization, including using natural language processing techniques to identify semantic boundaries, splitting text based on fixed token or character counts, or utilizing document structure such as paragraphs or sections. Advanced methods might consider the coherence and completeness of information within each chunk.

Effective chunk optimization can significantly improve retrieval accuracy, as it allows the system to pinpoint and return more precise, relevant information. It also enhances the efficiency of the embedding process and reduces computational overhead during retrieval operations. Furthermore, well-optimized chunks can lead to more coherent and contextually appropriate responses in the generation phase of RAG systems.

1. Fixed-size chunking: This is the simplest and most common method. It involves dividing text into chunks of a predetermined size, often with some overlap to maintain context. It's computationally efficient and suitable for most situations.
2. Content-aware chunking: This approach takes into account the nature of the content being chunked. It includes:
   * Sentence splitting: Using tools like NLTK or spaCy to divide text into meaningful sentence-level chunks.
   * Recursive chunking: An iterative method that uses multiple separators to create chunks of similar sizes while respecting content structure.
3. Specialized chunking: This is used for structured formats like Markdown or LaTeX, preserving the original document structure during the chunking process.
4. Semantic chunking: An experimental technique that uses embeddings to create chunks based on semantic similarity. It aims to group sentences with similar themes or topics together, potentially offering more contextually relevant chunks.

Each method has its strengths and is suited to different types of content or retrieval needs. The choice of chunking method can significantly impact the effectiveness of information retrieval and the quality of generated responses in a RAG system.

**2.2.4.2 Multi-representation Indexing**

Multi-representation indexing is an advanced technique in Retrieval-Augmented Generation (RAG) systems that involves creating and storing multiple representations of the same content to enhance retrieval accuracy and flexibility.  
This approach recognizes that different query types or contexts may benefit from different representations of the same information. In a multi-representation index, a single piece of content might be represented by its full text, a summary, key phrases, entity extractions, or various types of embeddings (e.g., semantic, contextual, or domain-specific). Each representation captures different aspects of the content, allowing the retrieval system to match queries more effectively across various dimensions.  
For instance, a semantic embedding might capture overall topic relevance, while a key phrase index could match specific terminology. By leveraging these multiple representations during the retrieval process, the system can more accurately identify and rank relevant information, potentially combining signals from different representations to improve overall retrieval performance. This approach can be particularly beneficial in domains where information needs are diverse or where the same content may be relevant in different contexts or for different types of queries.  
Let us discuss few approaches:

Parent Document Representation:

* The "parent document" refers to the original, complete piece of content from which various representations are derived. This could be a full article, report, or any other coherent unit of information. Now if we’re dealing with a large document, such as an e-book or a research paper, instead of treating the entire document as a monolithic unit, multi-representation indexing breaks it down into smaller, more manageable pieces.
* These smaller chunks are represented in various ways, capturing different aspects of the content. For example:
  + Sentence Embeddings: For shorter texts, sentence embeddings (such as those generated by sentence transformers) may suffice.
  + Token-Based Embeddings: For larger documents, chunking by tokens (using models like “text-embedding-ada-002”) provides more context.
  + Summary-Like Representations: Creating summary-like representations helps maintain context while optimizing for retrieval efficiency.
* **Dense X Retrieval Technique**:
  + Dense X is a retrieval technique used in RAG systems like Langchain and LlamaIndex.
  + Instead of chunking the entire document and retrieving the top-k results based on semantic similarity, Dense X converts the document into compact retrieval units.
  + These retrieval units are represented by real-valued vectors (embeddings). The similarity between a user query and these embeddings is computed using efficient nearest neighbor techniques (such as those provided by the FAISS toolkit).
  + [The goal is to optimize retrieval accuracy while maintaining efficiency and relevance](https://towardsai.net/p/machine-learning/dense-x-retrieval-technique-in-langchain-and-llamaindex).

Multi-representation indexing plays a crucial role in RAG by ensuring that relevant information is accessible for both retrieval and generation

**2.2.4.3 Specialized Embeddings**

**Specialized embeddings** refer to context-aware vector representations of words, phrases, or sentences that capture specific semantic nuances. Unlike generic word embeddings (such as Word2Vec or GloVe), specialized embeddings are tailored for particular tasks or domains. Here are some examples:

1. **BioBERT Embeddings**:
   * BioBERT, pre-trained on biomedical literature, provides embeddings specifically suited for biomedical text analysis.
   * Researchers use BioBERT to enhance performance in tasks like named entity recognition, relation extraction, and biomedical question answering.
2. **ELMo (Embeddings from Language Models)**:
   * ELMo generates contextualized word embeddings by considering the entire sentence context.
   * It captures polysemy (multiple meanings) and context-dependent word senses, making it valuable for downstream NLP tasks.
3. **ClinicalBERT**:
   * ClinicalBERT is fine-tuned on clinical notes and electronic health records.
   * It excels in medical text understanding, including tasks like diagnosis prediction, symptom extraction, and patient outcome prediction.
4. **Domain-Specific Embeddings**:
   * Researchers often train embeddings on domain-specific corpora (e.g., legal texts, financial reports, or scientific articles).
   * These embeddings enhance performance in specialized applications within those domains.

Specialized embeddings empower NLP models to handle domain-specific jargon, context, and nuances effectively.

One instance of using specialized embeddings is fine-tuning of LLMs. LLM fine-tuning involves taking a pre-trained model and further training it on a domain-specific dataset. These pre-trained models, such as GPT (Generative Pre-trained Transformer), are initially trained on vast amounts of text data, enabling them to understand natural language and generate human-like text. [However, fine-tuning tailors these models to specific tasks or domains, enhancing their performance and broadening their applicability across various fields1](https://www.datacamp.com/tutorial/fine-tuning-large-language-models). [The process typically includes steps like choosing a pre-trained model, loading data, using tokenizers, initializing the base model, evaluating performance, and fine-tuning using specialized methods1](https://www.datacamp.com/tutorial/fine-tuning-large-language-models). [By fine-tuning, we can leverage cutting-edge models without building them from scratch, making NLP tasks more accessible and efficient](https://www.datacamp.com/tutorial/fine-tuning-large-language-models).

**ColBERT (Contextualized Late Interaction over BERT)** is a state-of-the-art neural search model developed by Stanford University. [It combines the deep language understanding of BERT with a novel interaction mechanism, allowing efficient and accurate retrieval over large text collections1](https://github.com/stanford-futuredata/ColBERT). Here are the key points about ColBERT:

1. **Late Interaction Approach:**
   * Unlike traditional embedding models like BERT, which consolidate token vectors into a single representation, ColBERT maintains per-token representations. This fine-grained approach enables more precise similarity calculations.
   * [At search time, ColBERT embeds each query into a matrix and efficiently finds passages that contextually match the query using scalable vector-similarity operators1](https://github.com/stanford-futuredata/ColBERT).
2. **How Specialized Embeddings Are Used:**
   * ColBERT encodes both queries and documents separately.
   * For documents, instead of creating a single embedding for the entire passage, ColBERT generates contextualized embeddings for each token. These token-level embeddings capture more nuanced information about the passage.
   * [The rich interactions between query and passage embeddings allow ColBERT to surpass the quality of single-vector representation models while efficiently scaling to large corpora](https://zilliz.com/learn/explore-colbert-token-level-embedding-and-ranking-model-for-similarity-search).

**2.2.4.4 Hierarchical Retrievals**

**Hierarchical retrieval** is a strategy that narrows down data to relevant chunks in a structured manner. It executes the retrieval process in a hierarchical order, with multiple levels of organization. [The number of levels in the hierarchy can vary based on the specific context of usage](https://pixion.co/blog/rag-strategies-hierarchical-index-retrieval). Essentially, it involves breaking down the retrieval process into steps, where each step refines the search space by considering progressively more specific information. [This approach can be particularly useful for complex tasks like open-domain question answering, where efficient and accurate retrieval is crucial](https://www.amazon.science/publications/hybrid-hierarchical-retrieval-for-open-domain-question-answering)

RAPTOR

**RAPTOR** (Recursive Abstractive Processing for Tree-Organized Retrieval) introduces a novel approach to retrieval-augmented language models by constructing a recursive tree structure from documents. [This allows for more efficient and context-aware information retrieval across large texts, addressing common limitations in traditional language models](https://github.com/profintegra/raptor-rag). By employing recursive clustering and summarization techniques, RAPTOR creates a hierarchical tree structure capable of synthesizing information across various sections of retrieval corpora. [During the query phase, RAPTOR leverages this tree structure for more effective retrieval](https://www.langchain.ca/blog/efficient-information-retrieval-from-complex-pdfs-using-raptor-rag/). [The architecture enhances document understanding by recursively clustering related text chunks and summarizing them, capturing both meaning and structural hierarchy across various levels of abstraction](https://web.stanford.edu/class/cs224n/final-reports/256925521.pdf). [Introduced in 2024, RAPTOR embeds, clusters, and summarizes chunks of text to create a tree representing different levels of abstraction4](https://becomingahacker.org/comparing-rag-rag-fusion-with-raptor-different-ai-retrieval-augmented-implementations-1aa76fce6a5c).

**2.2.5 Retrieval**

In the context of Retrieval-Augmented Generation (RAG), retrieval refers to the process of fetching relevant information from a large corpus of data to support the generation of accurate and contextually relevant text by a language model. This process begins with formulating a search query based on the user's input. The retrieval mechanism then searches the corpus, identifying the most pertinent documents through various search algorithms and indexing techniques. Once these documents are retrieved, they are integrated into the language model's input, enabling it to generate responses that are grounded in specific, relevant data. This integration significantly enhances the model's ability to provide precise and factually correct answers, making RAG particularly effective for applications requiring high accuracy, such as automated question-answering, research assistance and content creation​.

Reranking

Reranking, also known as rank refinement or re-scoring, is a technique used in information retrieval to enhance the quality of search results generated by a primary ranking model. Initially, a primary ranking model retrieves and ranks a set of documents or items based on certain criteria, often using simple algorithms or basic feature sets. However, the initial ranking might not fully capture all the relevant aspects needed to deliver the most pertinent results.

To address this, a secondary ranking model is applied to reorder the initially ranked items. This secondary model can incorporate more complex features, user preferences, contextual information, or additional data that were not considered during the primary ranking phase. Techniques for reranking include machine learning algorithms, ensemble methods, and deep learning models like BERT or other transformer-based models. For instance, in the context of question-answering systems, reranking can prioritize the most relevant and accurate answers by refining the initial list of retrieved documents.

Reranking plays a crucial role in applications such as search engines, recommender systems, and natural language processing tasks. By refining search results or recommendations, reranking ensures that users receive the most relevant, high-quality information, thereby improving overall user experience and satisfaction.

Re-Rank

Rerank 3, developed by Cohere, is a foundation model designed to enhance enterprise search and retrieval efficiency. It significantly improves search relevance and accuracy by leveraging state-of-the-art machine learning techniques to rerank search results based on various sophisticated features. This model aims to deliver highly relevant results quickly, catering to the complex needs of enterprise search applications.  
Rerank 3 enhances enterprise search with cutting-edge features:

* 4,000 token context length, boosting search quality for extensive documents
* Versatile search across complex data types including emails, invoices, JSON, code, and tabular information
* Support for over 100 languages
* Enhanced speed and cost-efficiency

While generative models with extended context can perform Retrieval-Augmented Generation (RAG), optimal RAG solutions combine generative and Rerank models. Rerank 3's precise semantic reranking ensures only the most pertinent information reaches the generative model. This approach improves response accuracy while minimizing latency and costs, particularly when retrieving data from vast document collections.

RankGPT

This method aims to evaluate their performance and introduces a novel instructional permutation generation approach where LLMs generate permutations of passages based on query relevance instead of intermediate relevance scores. Findings indicate that properly instructed LLMs can outperform state-of-the-art supervised methods on benchmarks like TREC, BEIR, and Mr.TyDi, with GPT-4 achieving particularly high performance on a new test set called NovelEval. This model also introduces a permutation distillation method to transfer these ranking capabilities into smaller, specialized models, demonstrating that a distilled 440M model can surpass a 3B supervised model on BEIR. This comprehensive evaluation underscores the potential of LLMs in IR and provides a practical pathway for their deployment and optimization. (Sun et al., 2023)  
It has been implemented by llamaindex.  
  
RAG Fusion

RAG-Fusion is a novel chatbot model that combines Retrieval-Augmented Generation (RAG) with Reciprocal Rank Fusion (RRF)**.**(Rackauckas, 2024)

 **Process**:

* RAG-Fusion involves sending the original query to a large language model to generate multiple new search queries based on the original query.
* These generated queries are then reranked using reciprocal scores and fused with documents to provide a final list of relevant information.

 **Purpose**:

* The aim of RAG-Fusion is to enhance the accuracy and comprehensiveness of answers provided by chatbots.

 **Advantages**:

* By generating multiple queries and reranking documents, RAG-Fusion can offer more contextualized responses to user queries.
* The model's integration of RRF as a reranker has shown improved performance in document reranking compared to other methods.

 **Challenges**:

* One of the main challenges of RAG-Fusion is the slower response time compared to traditional RAG chatbots due to complex API calls and multiple queries generation.
* Evaluation frameworks like ROUGE and BLEU may not be fully effective in assessing the performance of RAG-Fusion in certain tasks.

Refinement

Refinement in the context of retrieval in RAG (Retrieval-Augmented Generation) systems refers to the process of improving the quality and relevance of the retrieved information before it's passed to the language model for generation. Here's a more detailed explanation:

1. Basic concept: Refinement is an additional step between the initial retrieval of documents and the final generation phase. It aims to enhance the relevance and quality of the context provided to the language model.
2. Purpose:
   * To reduce noise in the retrieved information
   * To focus on the most relevant parts of retrieved documents
   * To improve the coherence and accuracy of the generated response
3. Common refinement techniques: a) Re-ranking:
   * Involves scoring and reordering the initially retrieved documents based on their relevance to the query
   * Often uses more sophisticated models than the initial retrieval step

b) Passage selection:

* + Identifying and extracting the most relevant passages from retrieved documents
  + Can help in reducing the context length and focusing on key information

c) Query expansion:

* + Enhancing the original query with additional relevant terms
  + Can help in retrieving more pertinent information in subsequent retrieval steps

d) Iterative retrieval:

* + Performing multiple rounds of retrieval, each time refining the query based on previously retrieved information

e) Summarization:

* + Condensing retrieved documents to extract key information
  + Useful when dealing with long documents or large amounts of retrieved text

1. Benefits:
   * Improved response quality
   * Reduced hallucinations in generated text
   * Better handling of complex or ambiguous queries
   * More efficient use of context window in language models
2. Challenges:
   * Balancing refinement quality with system latency
   * Ensuring that important information isn't lost during refinement
   * Adapting refinement techniques to different types of queries and document formats

Refinement is a crucial step in advanced RAG systems, as it helps bridge the gap between raw retrieval results and the high-quality context needed for effective generation. It's an active area of research and development in the field of information retrieval and natural language processing.

Active Retrieval

Active retrieval in the context of RAG (Retrieval-Augmented Generation) systems refers to a dynamic and iterative approach to information retrieval. Instead of a single, static retrieval step, active retrieval involves multiple rounds of querying and refining the search based on intermediate results or the ongoing generation process. Here's a more detailed explanation:

1. Basic concept: Active retrieval dynamically adjusts and performs multiple retrieval operations during the process of answering a query or generating content.
2. Process: a) Initial retrieval based on the user's query b) Analysis of retrieved information c) Formulation of follow-up queries or refinements d) Additional retrieval rounds based on these new queries e) Integration of newly retrieved information into the context
3. Key features:
   * Adaptive: Adjusts retrieval strategy based on intermediate results
   * Iterative: Performs multiple rounds of retrieval
   * Context-aware: Uses the developing context to guide further retrieval
4. Benefits:
   * Improved accuracy: Can find more relevant information that might be missed in a single retrieval step
   * Handling complex queries: Better equipped to address multi-part or nuanced questions
   * Reduced hallucinations: By actively seeking more information, it can fill gaps in knowledge
   * Improved coherence: Can retrieve additional context to ensure consistent and well-informed responses
5. Challenges:
   * Increased latency: Multiple retrieval rounds can increase response time
   * Computational cost: More intensive in terms of processing and potential API calls
   * Balancing depth vs. breadth: Deciding when to dig deeper on a topic vs. exploring new areas
   * Query formulation: Generating effective follow-up queries can be complex
6. Applications:
   * Question-answering systems
   * Research assistants
   * Dialogue systems requiring in-depth knowledge
   * Content generation tools needing diverse, accurate information
7. Variations:
   * Agent-based active retrieval: Using multiple specialized agents for different aspects of retrieval
   * Hybrid approaches: Combining active retrieval with other techniques like re-ranking or summarization

Active retrieval represents a more sophisticated approach to information gathering in RAG systems, aiming to mimic the way humans might research a topic – starting with a broad query and then diving deeper based on initial findings. It's particularly useful for complex tasks that require comprehensive and accurate information gathering.

Let us discuss one architecture CRAG (Corrective RAG) helps in Active Retrieval and Refinement

CRAG

Corrective Retrieval Augmented Generation (CRAG) is a method designed to enhance the robustness of retrieval-augmented generation (RAG) systems when faced with inaccurate or irrelevant retrieval results. CRAG employs a lightweight retrieval evaluator to assess document relevance and triggers one of three actions: Correct, Incorrect, or Ambiguous. For relevant documents, it refines key information into knowledge strips, while for irrelevant results, it turns to web search as an additional knowledge source. The method was evaluated on four datasets covering short-form and long-form generation tasks, demonstrating significant improvements over standard RAG and advanced methods like Self-RAG. (Yan et al., 2024)

The methodology of Corrective Retrieval Augmented Generation (CRAG) involves several key steps:

1. Retrieval Evaluation:
   * A lightweight T5-based retrieval evaluator assesses the relevance of retrieved documents to the input query.
   * It calculates a confidence score for each retrieved document.
2. Action Triggering:
   * Based on the confidence scores, one of three actions is triggered: a) Correct: When at least one document is highly relevant b) Incorrect: When all documents are deemed irrelevant c) Ambiguous: When the evaluator cannot confidently determine relevance
3. Knowledge Refinement (for "Correct" action):
   * Relevant documents are decomposed into fine-grained knowledge strips.
   * These strips are filtered and recomposed to focus on key information.
4. Web Search Integration (for "Incorrect" action):
   * When retrieved documents are irrelevant, web search is used as an alternative.
   * The input query is rewritten into search keywords.
   * Web search results are processed and refined similarly to internal documents.
5. Combination (for "Ambiguous" action):
   * Both refined internal documents and web search results are used.
6. Generation:
   * The processed knowledge (internal, external, or combined) is fed to a language model for final text generation.

This methodology allows CRAG to adapt to different retrieval qualities and provide more reliable knowledge for generation. The paper demonstrates its implementation with various RAG-based approaches and language models, showing its flexibility and effectiveness across different tasks and scenarios.

**2.2.6 Generation**

In the Retrieval-Augmented Generation (RAG) architecture, the generation step is a crucial component that follows the retrieval process. Here's an explanation of the generation step in RAG:

1. Purpose: The generation step is responsible for producing the final output text based on the input query and the retrieved information.
2. Process:
   * It takes as input: a) The original query or prompt b) The relevant documents or information retrieved in the earlier step
   * A language model (often a large language model like GPT or LLaMA) is used to generate the response.
   * The model considers both the query and the retrieved information to produce contextually relevant and informative text.
3. Key aspects:
   * The language model is typically pre-trained and may be fine-tuned for the specific task.
   * It needs to effectively integrate the retrieved information with the query to generate accurate and coherent responses.
   * The model should be able to synthesize information from multiple retrieved documents if necessary.
4. Variations:
   * Some RAG systems may use a separate encoder for the retrieved documents and a decoder for generation.
   * Others might use a single model for both understanding the retrieved content and generating the response.
5. Challenges:
   * Balancing the use of retrieved information with the model's own knowledge.
   * Avoiding hallucinations or generating false information not present in the retrieved documents.
   * Maintaining coherence and relevance to the original query.

Generation quality is an important indicator of how accurate the answers are. To improve the generation quality, approaches like Self RAG and RRR (Rewrite-Retrieve-Read) has been introduced. We will go through them one-by-one.

Self RAG

SELF-RAG is a framework that enhances the quality and factuality of large language models through retrieval and self-reflection. It improves overall generation quality, outperforming non-proprietary and proprietary models in various tasks. The inference process involves generating tokens and reflection tokens to guide the model in generating text. Experimental details include training setup, evaluation metrics, datasets used, results analysis, human evaluations, and qualitative examples.(Asai et al., 2023)  
  
Key Insights

* SELF-RAG improves generation quality by incorporating retrieval and self-reflection mechanisms.
* It outperforms baseline models in fact verification, question answering, and long-form generation tasks.
* The framework shows significant gains in improving factuality, citation accuracy, and overall performance.
* Training setup involves using a Cloud TPU v4, Adam optimizer, and multiple evaluation metrics.
* Human evaluations confirm the reliability and accuracy of SELF-RAG in generating supported responses.

RRR Algorithm

The proposed RRR algorithm iteratively improves retrieval and rewrite stages using pre-trained language models to enhance document retrieval, refinement, and query rewrite. It outperforms existing techniques in benchmarks, achieving up to a 17% relative gain in nDCG metrics. The feedback loop between retrieval and rewrite stages boosts performance, with the final re-ranker playing a crucial role in fine-grained ranking. By combining GAR and RAG paradigms, RRR excels in zero-shot information retrieval tasks, utilizing GPT models for rewriting and retrieval.(Sun et al., 2023)  
  
Key Insights

* The RRR algorithm leverages pre-trained language models for effective document retrieval and query rewrite.
* Performance evaluation on benchmarks demonstrates up to a 17% relative gain in nDCG metrics compared to state-of-the-art techniques.
* The feedback loop between retrieval and rewrite stages significantly enhances system performance.
* The final re-ranker is essential for fine-grained ranking, with relevance assessment based on the original query improving performance.
* RRR combines GAR and RAG paradigms to achieve superior performance in zero-shot information retrieval tasks.

**2.3 Related Research Publications**

The Probabilistic Relevance Framework (PRF) provided a structured approach to document retrieval, originating from research during the 1970s and 1980s. This framework laid the foundation for the development of BM25, one of the most effective text-retrieval algorithms. Recent advancements in PRF research have introduced new retrieval models that incorporate document meta-data, particularly structural and link-graph information. These developments have further enhanced algorithms such as BM25F, making them highly successful in web-search and corporate-search applications. (Robertson and Zaragoza, 2009)

Large Language Models (LLMs) as we know them today have evolved over time, but their foundations can be traced back to the early 2010s. Here's a brief timeline:

1. 2013-2014: Word2vec and GloVe models were introduced, laying groundwork for representing words as vectors.
2. 2017: The "Attention Is All You Need" paper introduced the Transformer architecture, a key innovation for modern LLMs.
3. 2018: OpenAI released GPT (Generative Pre-trained Transformer), one of the first large-scale language models using the Transformer architecture.
4. 2019: OpenAI released GPT-2, which was significantly larger and more capable than its predecessor.
5. 2020: OpenAI released GPT-3, which marked a major leap in scale and capabilities.

Since then, many other companies and organizations have developed their own LLMs, including Google's BERT and LaMDA, Facebook's RoBERTa, Microsoft and NVIDIA's Megatron-Turing NLG, and Anthropic's Claude models.

On the other hand, RAG, which stands for Retrieval-Augmented Generation, is a relatively recent development in the field of natural language processing and AI. RAG was introduced in 2020 by researchers from Facebook AI Research (now Meta AI) and University College London. Since its introduction, RAG has influenced the development of many AI systems and has been adopted or adapted by various organizations to improve their language models and chatbots.  
The concept of RAG continues to evolve, with researchers exploring various improvements and applications in different domains.  
RAG represents an important step in making AI language models more reliable and capable of leveraging external knowledge. It's part of a broader trend in AI research towards creating systems that can effectively combine pre-trained knowledge with the ability to retrieve and use new information.

A lot of research has taken place since the introduction of RAG which are:  
  
1. REALM (Retrieval-Augmented Language Model Pre-Training) is an innovative approach introduced by Google Research in 2020, designed to enhance language models by integrating external knowledge during the pre-training process. Unlike RAG, which applies retrieval at inference time, REALM incorporates a neural retriever during pre-training, allowing the model to learn how to effectively use external information. This method involves training the model to retrieve relevant documents from a large corpus and use them to inform its predictions on masked language modeling tasks. REALM offers several advantages, including improved performance on knowledge-intensive tasks, better interpretability, and the ability to update knowledge by modifying the retrieval corpus without retraining the entire model. This approach has significantly influenced subsequent research in retrieval-augmented language models and contributes to the ongoing trend of developing more knowledgeable and adaptable AI systems. REALM represents a crucial step in the evolution of language models, aiming to create AI that can effectively combine deep learning capabilities with structured knowledge retrieval.  
  
2. Dense Passage Retriever (DPR) is a significant advancement in information retrieval systems, introduced in 2020 by researchers from Facebook AI Research (now Meta AI).

DPR uses dense representations for both queries and passages, moving away from traditional sparse retrieval methods like TF-IDF or BM25. It employs two BERT-based neural networks: one for encoding queries and another for encoding passages. These encoders map queries and passages into a shared, dense vector space where relevance is measured by vector similarity.

The system is trained on question-answer pairs, learning to encode questions and their corresponding answer passages close together in the vector space. This allows DPR to effectively retrieve relevant passages for a given query, even when there's little lexical overlap between the query and the relevant passage.

DPR's key advantages include its ability to capture semantic relationships beyond exact keyword matching, its scalability to large document collections, and its strong performance on various question-answering and information retrieval tasks. It has become a cornerstone in many modern retrieval systems and has influenced subsequent research in dense retrieval methods.  
  
3. "Leveraging Passage Retrieval with Generative Models for Open Domain Question Answering" is a research paper that made significant contributions to the field of open-domain question answering. Here's an overview:

This paper, published in 2020, introduced the Fusion-in-Decoder (FiD) approach, which combines retrieval and generation for more effective question answering. The authors proposed a method that retrieves multiple relevant passages for a given question and then uses a generative model to process these passages together with the question to produce an answer.

Key aspects of the FiD approach include:

1. Retrieval: It uses a retriever (such as DPR) to fetch multiple relevant passages for a given question.
2. Encoding: Each retrieved passage is independently encoded along with the question.
3. Fusion: The encoded representations are concatenated and processed together in the decoder of a sequence-to-sequence model.
4. Generation: The model generates the final answer based on the fused information from all retrieved passages.

This approach showed significant improvements over previous methods, particularly in handling complex questions that require information from multiple sources. It demonstrated the power of combining retrieval and generation in a way that allows the model to consider multiple pieces of evidence simultaneously.

The FiD method has been influential in subsequent research on open-domain question answering and has contributed to the broader trend of integrating retrieval mechanisms with large language models for more accurate and informative responses.

4. ColBERT v2, introduced in 2021, is an advanced neural information retrieval model that builds upon the original ColBERT architecture. It maintains ColBERT's core strengths of late interaction and token-level encodings while introducing several key improvements. Notably, ColBERT v2 enhances efficiency through reduced dimensionality and quantization, allowing for more compact index sizes and faster query processing. It also employs advanced training techniques to improve effectiveness, resulting in state-of-the-art performance on various retrieval benchmarks. Unlike its predecessor, ColBERT v2 offers better scalability for large-scale retrieval tasks, making it more suitable for practical applications like web search and large-scale document retrieval. These enhancements enable ColBERT v2 to strike a balance between the deep semantic understanding of neural models and the efficiency required for real-world information retrieval systems.

5. "Atlas: Few-shot Learning with Retrieval Augmented Language Models" is a research paper presenting a novel approach to few-shot learning in large language models by integrating retrieval mechanisms, which came out in 2022. The model, named Atlas, is designed to effectively tackle knowledge-intensive tasks using significantly fewer parameters than traditional large models. It combines a retriever based on dense embeddings with a language model built on the T5 architecture, employing a Fusion-in-Decoder technique for efficiency.

Atlas demonstrates strong performance in few-shot settings, notably achieving a 42% accuracy on the Natural Questions dataset with only 64 examples, outperforming a much larger 540 billion parameter model by 3% and requiring 50 times fewer parameters. The model also excels in other tasks like TriviaQA and FEVER, proving its adaptability and efficiency. Atlas's architecture allows for efficient updating and fine-tuning, making it suitable for dynamic knowledge tasks without the need for extensive retraining or large-scale annotations.

6. The paper "Hierarchical Representations in Dense Passage Retrieval for Question-Answering," published in May 2023, proposes a novel method to enhance the performance of question-answering systems by leveraging hierarchical representations within transformer networks. Traditional dense passage retrieval techniques often rely on the final layer of a transformer to generate queries. However, this paper suggests that valuable information can be extracted from every layer of the transformer, not just the final one. By integrating retrieval mechanisms at multiple layers, the approach aims to utilize different levels of abstraction to improve the relevance of retrieved documents.

This hierarchical retrieval method shows significant improvements in downstream tasks, such as question answering, by combining the strengths of multiple layers within the transformer network. The system effectively aligns the hierarchical structures of the input questions with the hierarchical nature of the transformer model, resulting in better document retrieval and subsequently more accurate answers.

7. "Retrieve What You Need: A Mutual Learning Framework for Open-domain Question Answering" introduces a novel approach to enhance the performance of open-domain QA systems, which traditionally follow a retrieve-then-read paradigm (2024). This framework incorporates an intermediate module called the knowledge selector, trained with reinforcement learning, to improve the interaction between the retriever and the reader components. The benefits of this mutual learning framework include eliminating the need for additional annotated question-passage pairs, enhancing both retrieval and QA performance, and improving computational efficiency compared to previous models. Furthermore, it significantly boosts the zero-shot performance of large-scale pre-trained language models like ChatGPT by efficiently encapsulating relevant knowledge without exceeding input length constraints​.  
  
8. RAGAS (Retrieval Augmented Generation Assessment) is a framework for automated evaluation of Retrieval Augmented Generation (RAG) systems without requiring reference answers. It was introduced in 2023.   
  
RAGAS proposes metrics for three key dimensions:

* + - * Faithfulness: How well the answer is grounded in the retrieved context
      * Answer Relevance: How directly the answer addresses the question
      * Context Relevance: How focused and relevant the retrieved context is
* The metrics are implemented using prompts to large language models (LLMs) and do not require human-annotated references.
* To evaluate RAGAS, the authors created WikiEval, a new dataset with human judgments on the three quality dimensions for question-context-answer triples.
* Experiments on WikiEval show RAGAS metrics align well with human judgments, outperforming baseline LLM-based evaluation methods.
* RAGAS is integrated with popular RAG frameworks like LlamaIndex and LangChain for easy adoption.
* The paper argues that such automated evaluation is crucial for faster development cycles of RAG systems, especially given the rapid adoption of large language models.
* The authors note that context relevance was the most challenging dimension to evaluate automatically.

The paper concludes that RAGAS provides valuable insights for RAG system developers, even without ground truth data, and demonstrates strong alignment with human assessments, particularly for faithfulness and answer relevance.

**2.4 Summary**

In this chapter, we presented a comprehensive overview of Retrieval-Augmented Generation (RAG) systems, focusing on their architecture and recent advancements. We explored the key components of RAG, including query construction, query translation, routing, indexing, retrieval, and generation. We delved into various techniques within each component, such as specialized embeddings, hierarchical retrievals, and reranking methods. We also examined recent developments like SELF-RAG and the RRR algorithm, which aim to improve generation quality. We concluded with a chronological review of related research publications, tracing the evolution of language models and retrieval techniques from early word embedding models to more sophisticated approaches like REALM, Dense Passage Retriever, and Atlas. Through this comprehensive review, we highlighted the rapid progress and ongoing challenges in the field of RAG, emphasizing its importance in enhancing the performance and reliability of AI language models. Our analysis underscores the dynamic nature of this field and sets the stage for future research directions in RAG systems.

**CHAPTER 3**

**RESEARCH METHODOLOGY**

**3.1 Introduction**The concept of Retrieval-Augmented Generation (RAG) is effectively encapsulated within its operational flow, as illustrated in Figure 2, which outlines a standard RAG application process.

ChatGPT, renowned as a leading Large Language Model (LLM), is limited by the scope of its pretraining data and, as such, does not possess information on recent occurrences. RAG steps in to bridge this knowledge gap by fetching timely document excerpts from external databases. In the described scenario, RAG retrieves a collection of relevant news articles concerning the query. These articles, combined with the original question, form an enhanced prompt that empowers ChatGPT to generate a well-informed response. This illustration underscores how RAG can augment a model's output by integrating real-time data retrieval, showcasing the method's ability to enrich responses with current information.

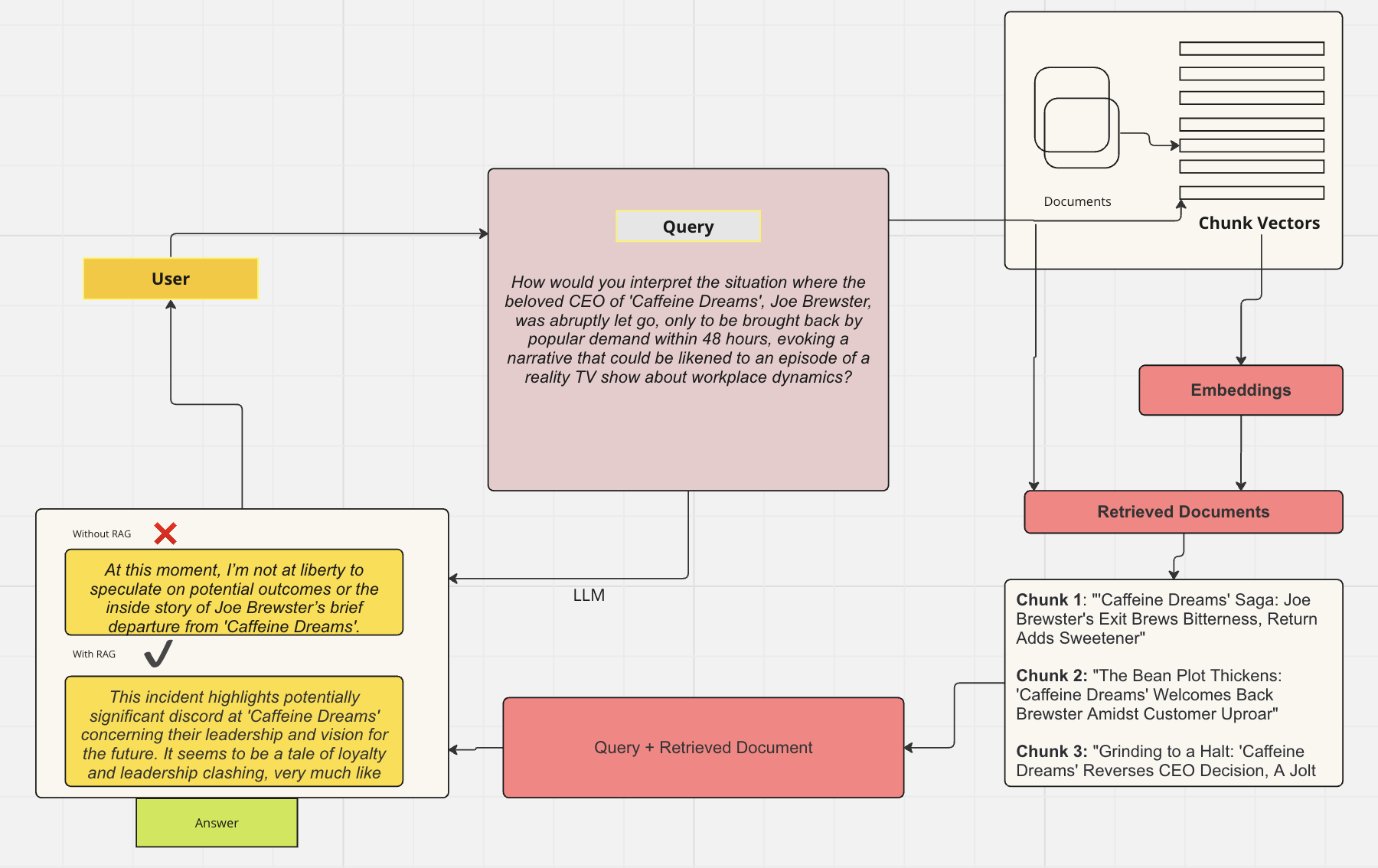


Figure 2: A representative instance of the RAG process applied to question answering

The RAG workflow comprises three key steps. First, the corpus is partitioned into discrete chunks, upon which vector indices are constructed utilising an encoder model. Second, RAG identifies and retrieves chunks based on their vector similarity to the query and indexed chunks. Finally, the model synthesises a response conditioned on the contextual information gleaned from the retrieved chunks. These steps form the fundamental framework of the RAG process, underpinning its information retrieval and context-aware generation capabilities. Next, we will provide an introduction to the RAG research framework.

**3.2 RAG Framework**

The field of Retrieval-Augmented Generation (RAG) is in a state of constant advancement, and this section aims to outline its development trajectory. We classify RAG into three distinct types: Naive RAG, Advanced RAG, and Modular RAG. Although the initial RAG models were economical and outperformed the capabilities of traditional Large Language Models (LLMs), they had their share of drawbacks. The emergence of Advanced RAG and Modular RAG came as a solution to address these specific limitations found in Naive RAG

**3.2.1 Naive RAG**

The Naive RAG approach marks the initial methodology that rose to prominence following the broad uptake of ChatGPT. This early model adheres to a conventional process encompassing indexing, retrieval, and generation steps, and is often referred to as the "Retrieve-Read" framework.

In this section, we describe a novel method to enhance the response generation of large language models (LLMs) by incorporating an information retrieval system based on vector embeddings. This approach leverages the power of Transformer Encoder models to encode textual data into high-dimensional vectors, which are then indexed in a vector database for efficient retrieval. By integrating this retrieval mechanism with an LLM, the system can provide more informed and contextually relevant responses to user queries.

**Indexing**

The indexing procedure is an essential preliminary step in the preparation of data, carried out offline, and consists of multiple phases. Initially, it involves the indexing of data where the raw data is purified and distilled. Various document formats, such as PDF, HTML, Word, and Markdown, are standardised into plain text format. Due to the context size constraints of language models, this text is further divided into smaller segments, a process known as chunking. These segments are then encoded into vector forms using an embedding model selected to optimise the trade-off between inference speed and model size, aiding in similarity assessments during the retrieval stage. Subsequently, an index is constructed to efficiently store these textual segments and their corresponding vector embeddings in a key-value structure, ensuring scalable and effective search functionality. In our research, we employed both Langchain and Llamaindex for the chunking process, utilising a recursive character splitter with a chunk size of 1500 and chunk overlap of 150.

**Retrieval**

When a user submits a query, the system applies the encoding model previously used in the indexing phase to convert the input into a vector form. This enables the system to evaluate the similarity between the query's vector and those of the chunks in the corpus index. The system identifies and retrieves the top K chunks that most closely match the query. These selected chunks then serve as the extended context to generate a response to the user's inquiry.

**Generation**

The user's query and the chosen document excerpts are merged into a unified prompt that the large language model uses as a basis for crafting an answer. Depending on the specifics of the task at hand, the model can either utilise the knowledge built into its parameters or limit its responses to facts explicitly found in the supplied documents. For continued conversations, the model can also incorporate prior dialogue exchanges into the prompt, allowing for coherent and contextually aware multi-turn dialogues.

**Drawbacks in Naive RAG**

Naive RAG grapples with substantial obstacles across three fundamental dimensions: "Retrieval," "Generation," and "Augmentation."

The retrieval phase often confronts varied difficulties, chiefly low precision that can lead to the selection of irrelevant chunks, which in turn may cause issues like hallucinations or incomplete information capture. Another common retrieval problem is low recall, which prevents the model from capturing all pertinent chunks, thus limiting the Large Language Models' (LLMs) capacity to form well-rounded responses. Complicating these challenges further is the retrieval of outdated information that could lead to inaccuracies.

The quality of the generated responses also poses significant challenges, most notably hallucinations where the model produces content that's not based on the provided context. Additionally, the generation phase may yield responses that are irrelevant or exhibit potential toxicity or bias.

In the augmentation stage, effectively merging context from the retrieved passages with the ongoing generative task presents its own set of hurdles, which may lead to disjointed or inconsistent outputs. The issue of redundancy and repetition arises when multiple retrieved passages overlap in content, leading to a repetitive narrative in the generated response.

Bad Retrieval:

* Retrieval issues mainly stem from the quality and relevance of the data retrieved by the system, which subsequently impacts the response generation. Key retrieval-related challenges include:
  + Low Precision: This occurs when the retrieved set of documents or data chunks contains irrelevant information. The presence of irrelevant data can lead to several issues in response generation, such as:
    - Hallucinations: The model may generate responses based on irrelevant or incorrect information, leading to fabricated content that does not align with accurate data or the user's query.
    - Lost in the Middle Problems: The model might produce responses that start relevantly but veer off-topic, losing coherence and relevance to the initial query.
  + Low Recall: If the retrieval system fails to fetch all relevant data chunks, it may not provide the model with sufficient context to generate accurate and comprehensive responses. This lack of context can hinder the model's ability to synthesise a well-informed answer.
  + Outdated Information: Retrieving outdated or redundant information can lead to responses that are no longer accurate or relevant, reducing the utility and reliability of the system.

Bad Response Generation:

* Once the data is retrieved, the generation component is responsible for synthesising an answer. Challenges in this phase include:
  + Hallucination: The model may create answers that are not supported by the retrieved data or are entirely fabricated, leading to misinformation.
  + Irrelevance: The generated response might not address the user's query effectively, providing information that, while possibly accurate, is unrelated to the question posed.
  + Toxicity/Bias: The response generation process might introduce harmful, offensive, or biassed content, which can undermine the credibility and ethical standing of the system.

Determining the significance and relevance of each retrieved passage to the generative task is another challenge, necessitating an equitable evaluation of each passage's contribution. Moreover, harmonising varied writing styles and tones to achieve a uniform output is paramount.

A final consideration is the overreliance on augmented information by generative models, which risks producing outputs that lack originality or synthesised insights, merely echoing the retrieved content.

**3.2.2 Advanced RAG**

Advanced RAG has been developed with targeted enhancements to address the shortcomings of Naive RAG. In terms of retrieval quality, Advanced RAG implements pre-retrieval and post-retrieval strategies. To address the indexing challenges experienced by Naive RAG, Advanced RAG has refined its indexing approach using techniques such as sliding window, fine-grained segmentation, and metadata. It has also introduced various methods to optimise the retrieval process

In the context of Naive RAG (Retrieval-Augmented Generation) systems, several challenges can impact the quality of responses generated. These challenges can be broadly categorised into issues related to retrieval quality and response generation quality. Here is a structured breakdown of these challenges for inclusion in the research paper.

**Data Indexing Enhancement for RAG Systems(Pre-Retrieval)**

The enhancement of data indexing is critical for improving the substance of indexed content. Five key strategies are employed to achieve this goal: improving data granularity, refining index structures, enriching with metadata, optimising alignment, and utilising mixed retrieval methods.

Firstly, improving data granularity is fundamental for the RAG system's efficiency. It focuses on the standardisation and consistency of text, verification of factual details, retention of rich context, and removal of extraneous information. By clarifying entities and terminology, ensuring accuracy, and updating obsolete documents, the system's ability to perform is significantly elevated.

Refinement of index structures is another vital strategy. It entails adjusting the size of document chunks to better capture context, enabling queries over diverse indexing paths, and exploiting the graph structure's potential to seize pertinent context by utilising the connections between graph nodes.

The incorporation of metadata also plays a significant role. By embedding metadata, such as dates and purposes, into the chunks, filtering becomes more targeted, and the inclusion of metadata like chapter and subsection references further bolsters retrieval efficacy.

Alignment optimization tackles the challenges of document consistency and congruence. It involves embedding "hypothetical questions" into documents, a novel technique to resolve alignment discrepancies and ensure coherence among varied document sources.

**Fine-Tuning Retrieval with Advanced Embeddings**

At the retrieval stage, the primary objective is to pinpoint the most pertinent context by assessing the query's similarity with document chunks, a task hinged on the embedding model's efficacy. In advanced RAG models, there are avenues for optimising these embedding models to heighten retrieval precision.

The process of fine-tuning embeddings is pivotal in sharpening the retrieval relevance. Customising models like the high-performance BGE embedding model can tailor retrieval for specific domains, addressing unique and evolving terminologies. Training data for this fine-tuning is generated by leveraging language models to create contextually grounded questions, subsequently used in fine-tuning.

Dynamic embeddings, in contrast to static ones, adjust to the usage context of words. For example, transformer models like BERT produce different embeddings for the same word based on its contextual usage. OpenAI's dynamic embedding models, based on LLM principles, offer a nuanced contextual understanding, though they may not match the contextual sensitivity of more recent comprehensive models like GPT-4.

**Post-Retrieval Process**

Upon securing the pertinent context from the database, integrating it effectively with the user query is vital for input into Large Language Models (LLMs), while navigating the constraints of context window limitations. Directly presenting all relevant documents to the LLM may surpass the context window limit, introduce extraneous information, and detract from focusing on essential details. Hence, further refinement of the retrieved content is imperative.

***Context Prioritization and Refinement***: A crucial step involves re-ranking the retrieved content to ensure that the most pertinent information is positioned prominently within the prompt. This methodology is employed in systems like LlamaIndex, LangChain, and HayStack. For instance, the Diversity Ranker emphasises sequence reorganisation based on the diversity of documents, while LostInTheMiddleRanker strategically places the most significant document at both the beginning and end of the context window. Furthermore, technologies such as cohereAI rerank, bge-rerank, and LongLLMLingua refine the process by recalculating the semantic proximity between the text and the query, tackling the challenge posed by vector-based approximations of semantic similarity.

***Enhancing Prompt Clarity:*** Studies have shown that the presence of irrelevant information in retrieved documents can negatively impact the efficacy of RAG. The focus in subsequent processing lies in condensing extraneous context, accentuating key paragraphs, and shortening the overall context length. Techniques like Selective Context and LLMLingua employ smaller language models to assess prompt mutual information or perplexity to gauge the significance of each component. Recomp advances this by training compressors across various levels of detail, whereas Long Context and “Walking in the Memory Maze” devise summarization strategies to bolster the LLM's recognition of critical information, especially when dealing with lengthy contexts**.**

**3.3 Advanced RAG Techniques**

**3.3.1 MultiQuery Retriever**

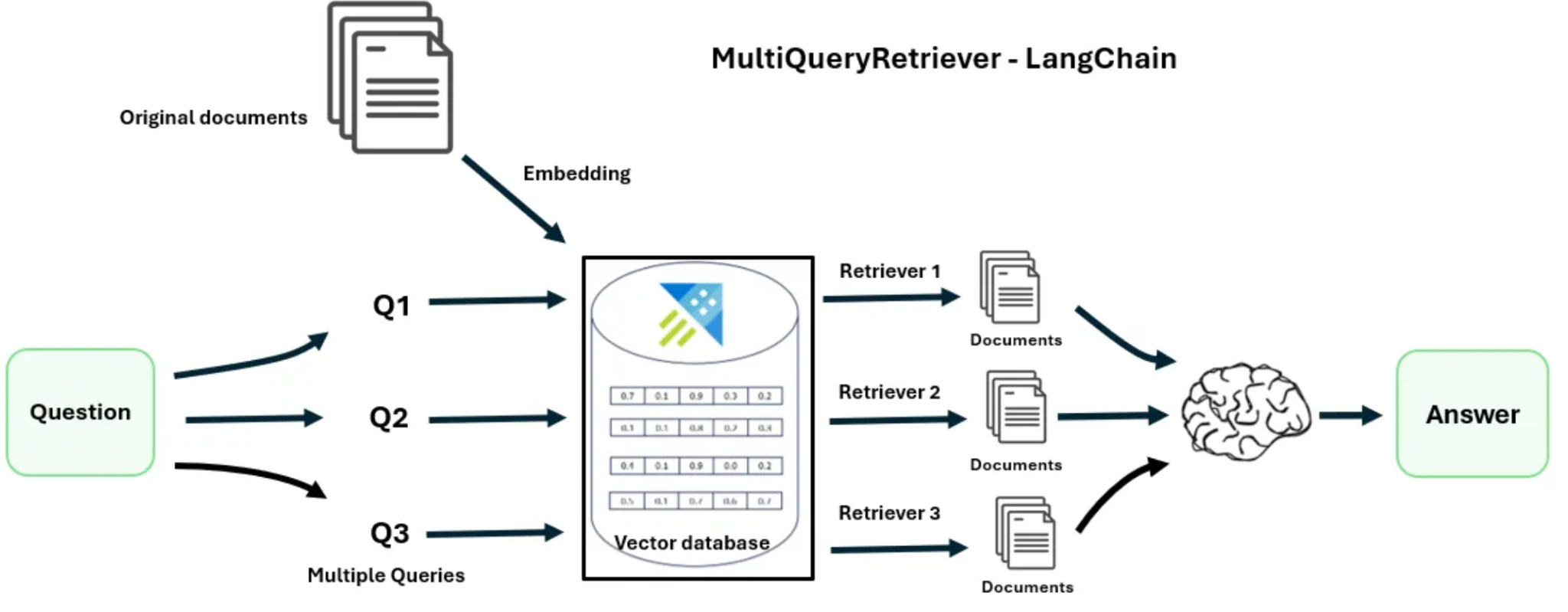
*The final outcome of the simple RAG method depends on how the query is written*. Even minor variations in the query phrasing sometimes can lead to different outcomes. To mitigate this strong query dependency and enhance result consistency, the *Multi Query Retriever method* emerges as an improved solution. This method doesn’t rely on a singular set of documents retrieved for an initial query to produce the final output. Instead, it harnesses the power of diversity by retrieving multiple sets of documents based on varied interpretations of the original query. This is particularly advantageous when dealing with queries that are vague or imprecisely formulated. By casting a wider net through multiple queries, this method markedly increases the likelihood of pinpointing the most relevant and accurate answers from the vast ocean of available documents.

In my research, I've explored the MultiQueryRetriever, an innovative approach designed to enhance the effectiveness of information retrieval processes. This technique automates prompt tuning by employing a Large Language Model (LLM) to craft multiple queries from distinct perspectives based on a single user input query.

For each generated query, the MultiQueryRetriever fetches a set of relevant documents. It then aggregates these documents, ensuring uniqueness across all queries, to assemble a more comprehensive set of potentially pertinent documents. This methodology is particularly beneficial because it generates various interpretations or perspectives on the same question, potentially addressing and overcoming some of the inherent limitations found in distance-based retrieval methods, thereby yielding a richer and more diverse set of results.

To illustrate, consider a scenario where the input query is, "What does the course say about regression?" The MultiQueryRetriever expands this into several nuanced queries, including:

* **"What is the course's perspective on regression?"**
* **"Can you provide information on regression as discussed in the course?"**
* **"How does the course cover the topic of regression?"**
* **"What are the course's teachings on regression?"**
* **"In relation to the course, what is mentioned about regression?"**

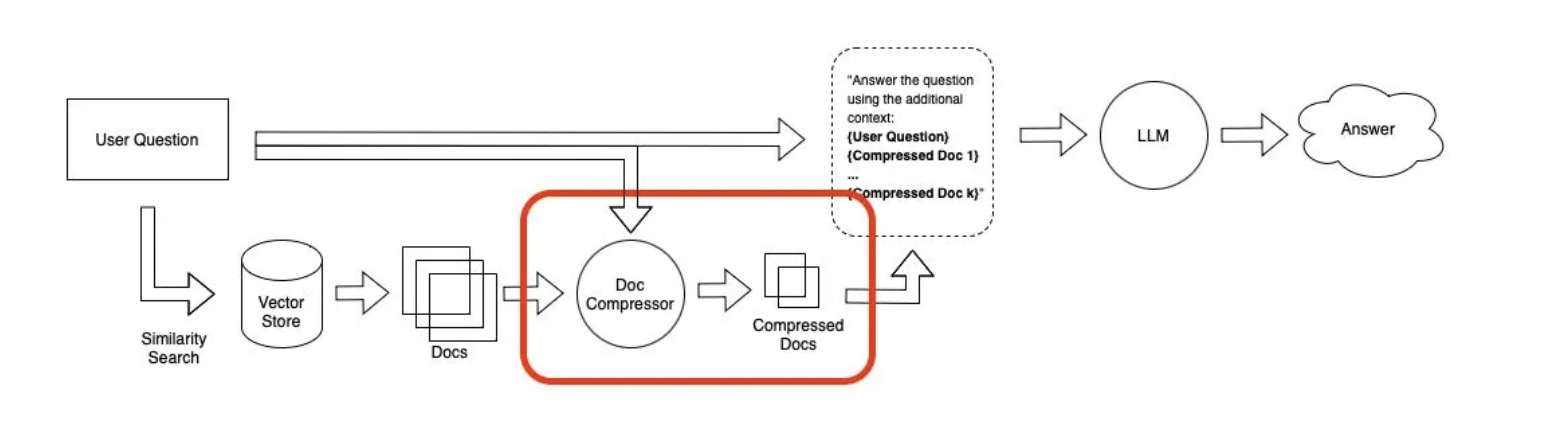
****

**3.3.2 Contextual Compression Retrieval Technique**

Context overflow is one of the biggest challenges in any RAG based application.The challenge with retrieval is that usually you don’t know the specific queries your document storage system will face when you ingest data into the system. This means that the information most relevant to a query may be buried in a document with a lot of irrelevant text. Passing that full document through your application can lead to more expensive LLM calls and poorer responses.

Contextual compression is meant to fix this. The idea is simple: instead of immediately returning retrieved documents as-is, you can compress them using the context of the given query, so that only the relevant information is returned. “Compressing” here refers to both compressing the contents of an individual document and filtering out documents wholesale.

**How does it work?**



In this research, we have used the contextual Compressor Retrieval Technique in a bit different way rather than going with the prebuilt compression techniques aided in Langchain and LLamaIndex. Instead we have used the sophisticated document retrieval pipeline using various components using Langchain library, specially tailored for handling large sets of documents and ensuring that the retrieved documents are relevant, non-redundant, and well ordered.

This process includes two main parts: **Rerranker** and **Contextual Compression Pipeline.**

**Let’s Breakdown the Rerranker part:**

We instantiate a class **BgeRerank** that extends **BaseDocumentCompressor** and is designed to rerank a sequence of documents based on their relevance to a given query using a cross-encoder model from the sentence-transformers library.

It has three main attributes :

* model\_name(model used for reranking): **BAAI/bge-reranker-large by default**
* top\_n: number of top documents to return after reranking(set as **3** in our use case)
* model: An instance of CrossEncoder initialised with model\_name. **This model is used to compute relevance scores between the query and each document.**

**This class has two main methods:**

* **bge\_rerank Method**: This method takes a query and a list of documents (docs) as input. It constructs pairs of the query and each document to create a list of model inputs.The model predicts relevance scores for each query-document pair.The documents are then sorted based on their scores in descending order.The method returns the top n documents as specified by the top\_n attribute.
* **compress\_documents Method:** This method overrides a method from *BaseDocumentCompressor*, which rerank a sequence of documents based on their relevance to a given query.It accepts a sequence of **Document objects**, a **query string**, and an **optional Callbacks instance**.It first checks if the document list isempty and returns an empty list if true, to avoid unnecessary API calls.It then extracts the page content from each Document object and passes it along with the query to the bge\_rerank method.The method updates each document's metadata with a **"relevance\_score"** based on the reranking results and returns a list of the updated documents.

The Contextual Compression Pipeline we have 7 main components:

* **​​Vector Store Retriever (vs\_retriever)(Chroma db has been used):**

This creates a retriever from a vector store with a specified search parameter k, which defines the number of documents to retrieve. **The vectorstore.as\_retriever()** method converts a vector store into a retriever object, enabling vector-based search. Basically it is called Naive RAG.

* **Ensemble Retriever (ensemble\_retriever):**

In our finding, EnsembleRetriever which combines multiple retrievers**, bm25\_retriever** and **vs\_retriever**, using a weighted scheme.Both retriever contributes to the final retrieval results, with weights specified as **[0.5, 0.5]**, where both retrievers have equal influence on the outcome.

* **Redundant Filter (redundant\_filter):**

We have employed **EmbeddingsRedundantFilter** from Langchain library to filter out redundant documents based on their embeddings. This step enhances the diversity of the retrieved documents by removing those that are too similar to each other.

* **Reordering (reordering):**

We have used **LongContextReorder** whichreorders documents based on certain criteria to ensure that the most relevant information is presented first. This is particularly useful when dealing with long documents or contexts.

* **Reranker (reranker):**

In this step we have used **BgeRerank**, a cross-encoder model reranker. It refines the order of documents based on their relevance to the query, improving the accuracy of the retrieval.

* **Document Compressor Pipeline (pipeline\_compressor):**

We have used this to chain **multiple document transformers** and a **rerranker** to process the retrieved documents. Here, it applies **the redundant filter**, **reordering**,

and **reranking** in sequence to the retrieved documents, optimising them for relevance and coherence.

* **Contextual Compression Retriever (compression\_pipeline):**

Finally, ContextualCompressionRetriever combines the **document compressor pipeline with the base retriever.**

The ensemble\_retriever(**bm25\_retriever,vs\_retriever**) fetches the initial set of documents, which are then passed through the **pipeline\_compressor** for filtering, reordering, and reranking.

**3.3.3 Sentence Window Retrieval**

In our investigation into enhancing retrieval-based models, we've delved into the Sentence-Window Retrieval method, an approach that significantly augments the contextual understanding capabilities of models by incorporating an extended range of adjacent sentences rather than limiting the focus to a single sentence. This method is particularly effective in augmenting the Retriever-Augmented Generation (RAG) model's comprehension by providing it with a more extensive contextual framework, thereby facilitating the generation of responses that are not only accurate but also contextually nuanced.

To elucidate, let's consider the application of this methodology within a chatbot aimed at offering technical troubleshooting assistance. When a user encounters a specific error, such as **"Error 404: File Not Found"**, and seeks help from the chatbot, the Sentence-Window Retrieval technique comes into play. It enables the chatbot to draw on a wide array of contextually relevant information sourced from documents like user manuals and forums, ensuring that the response is both pertinent and informed.

The operational flow of Sentence-Window Retrieval unfolds through several key stages:

* **Tokenization**: Initially, the corpus or documents are segmented into discrete sentences or relevant units, setting the foundation for the subsequent retrieval and response generation phases.
* **Query Formation**: The user's input or query serves as a beacon, guiding the search for pertinent information within the expansive corpus, directing the retrieval process.
* **Window Selection:** A pivotal step involves selecting a 'window' of sentences around the query's locus. The size of this window is adaptable, allowing for customization based on the specific needs of the retrieval task, thus ensuring the capture of relevant context.
* **Scoring and Ranking:** The relevance of sentences within the selected window is evaluated using the RAG model's scoring mechanisms, potentially leveraging and fine-tuning pre-existing language models to enhance the retrieval accuracy.
* **Retrieval and Generation:** In the final stage, the highest-ranked sentences are extracted and serve as the context for the generation phase. The RAG model synthesises this context to produce responses or summaries, ensuring that the output is both relevant and coherent.

We have implemented this advanced retrieval technique using **Llamaindex** library and used **weaviatevectorstore** available in the **Llamaindex as** vector store for the embedding of the top 50 files from the **Covid19** dataset available on Kaggle on different variety of questions(total 50 questions) on different difficulty levels. This retriever was evaluated on **30%** simple, **30%** multi contextual questions and **40 %** reasoning questions.

This Retriever was evaluated using RAGAs(Open source library to evaluate the RAG pipeline using different metrics

For Generator Component(**faithfulness**, **answer\_relevancy**)

For Retriever Component **(context\_recall**, **context\_precion**)

We can use this technique to improve the retrieval process of the RAG(Retrieval Component). We have used gpt-4 as the critic model to evaluate this metric with an underlying technique called GPT-4 Eval.

**3.3.4 Corrective Retrieval-Augmented Generation (CRAG)**In our exploration, we delved into the Corrective Retrieval-Augmented Generation (CRAG) technique, a cutting-edge approach in natural language processing designed to rectify factual inconsistencies and errors in text generation. CRAG ingeniously combines generative and retrieval-based capabilities to forge outputs that are more factually aligned.

Background and Motivation: Despite the impressive ability of text generation LLMs to produce fluent and coherent narratives, they often generate content with factual inaccuracies, termed as "hallucinations." These inaccuracies mainly stem from the training objective of these models, which focuses on predicting subsequent words based on the prior context, occasionally leading to the generation of plausible yet factually incorrect information. In addressing this challenge, retrieval-augmented generation techniques emerge, leveraging relevant context passages from a knowledge source to steer the text generation process. Nevertheless, mere retrieval augmentation doesn't inherently ensure adherence to the retrieved factual context, prompting the need for a more robust solution.

**High-Level Architecture:** The architecture of CRAG encompasses three pivotal components:

* **Generative Model:** This element generates initial text sequences in an auto-regressive manner, using advanced models like GPT-4 or T5, fine-tuned to enhance text fluency.
* **Retrieval Model:** This component fetches factually relevant passages from a knowledge base to underpin the generation process with accurate information.
* **Orchestrator:** Serving as the command centre, the orchestrator facilitates the interaction between the generative and retrieval models, orchestrating the evaluation and selection of text candidates based on their likelihood and factual alignment.

**Model Orchestration:** Orchestration is crucial, steering the interplay between text generation and knowledge retrieval. It meticulously monitors the generation progress, triggers the retrieval mechanism judiciously, evaluates the factual consistency of generation candidates, and incrementally constructs the final text, ensuring both coherence and factual accuracy.

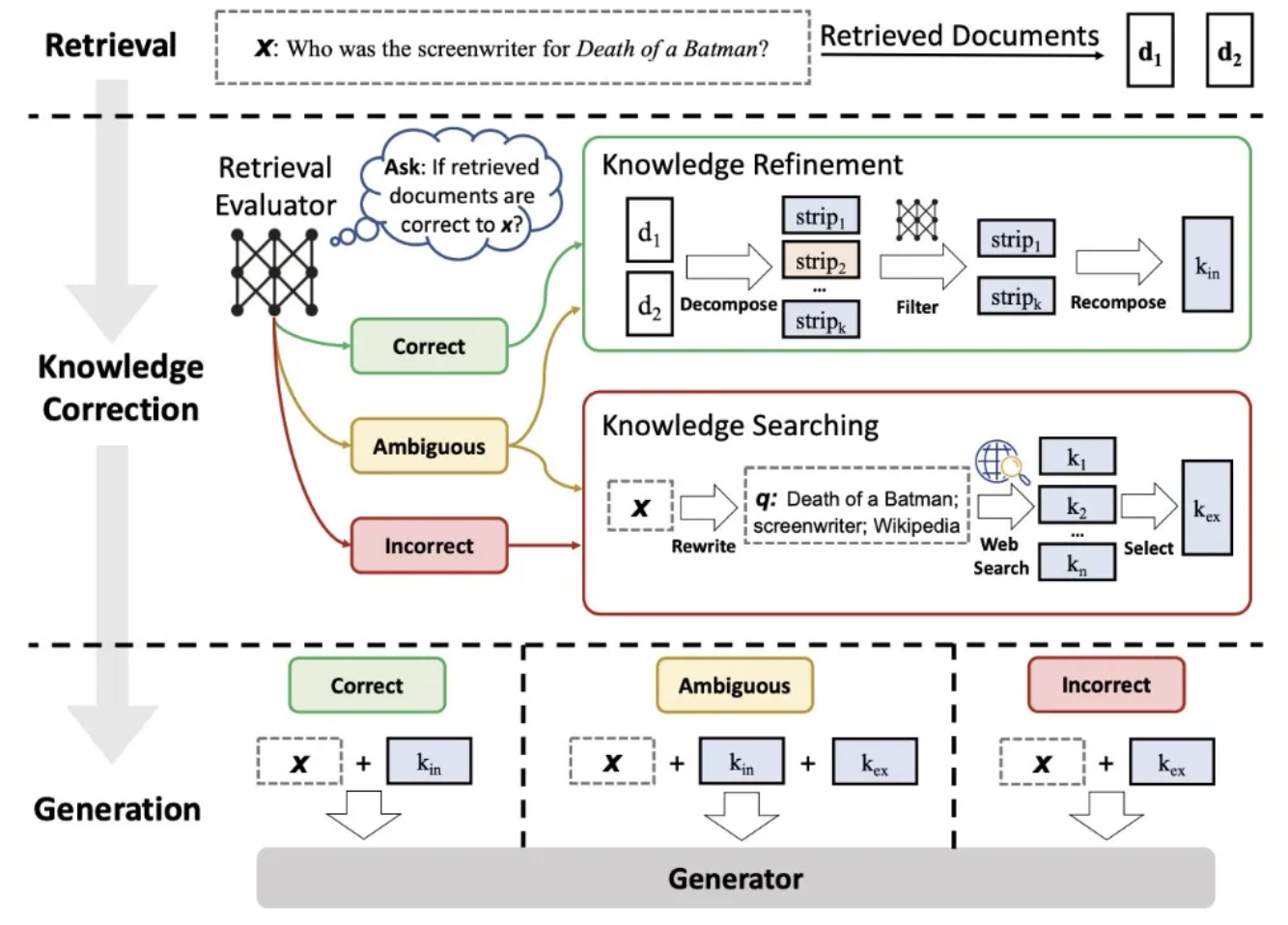
**Training Objectives:** Training CRAG focuses on dual goals: enhancing text generation fluency and ensuring factual correctness. A balanced scoring mechanism assesses candidates on both linguistic likelihood and factual alignment.

**Inference Procedure:** During inference, CRAG dynamically retrieves contextually pertinent information, utilising it to scrutinise and adjust the generative model's output, refining the text progressively.

**Strengths and Advantages**: CRAG's primary advantage lies in its capacity to amend factual inaccuracies in the generated text, marking a significant advancement for applications demanding high factual accuracy, such as technical support or academic content generation.

**Current Implementations and Results**: In our exploration, we can apply CRAG to scenarios like RAG on Covid19 dataset in case there are new innovations happening on a daily basis and the vector database increases everyday accordingly, where it markedly enhances the factual accuracy of responses, underscoring its practical utility and potential.

**Challenges and Future Work:** Despite its strengths, CRAG faces challenges like optimising retrieval efficiency and further enhancing the model's ability to discern and rectify nuanced factual inaccuracies. This is the reason we do not apply it in our current research. Future explorations will aim to refine these aspects, broaden application domains, and continuously improve the model's adherence to factual content.



**3.4 RAG Evaluation**

The swift progress and increasing utilisation of Retrieval-Augmented Generation (RAG) within Natural Language Processing (NLP) have elevated the assessment of RAG models to a critical area of focus within the Large Language Models (LLMs) community. This evaluation's main goal is to thoroughly understand and enhance the effectiveness of RAG models in various application settings.

**3.4.1 Evaluation Focus Areas**

The appraisal of RAG models centres on two pivotal elements: the retrieval and generation components. This bifurcation is essential for a comprehensive examination of the context's quality provided by the retrieval module and the content's quality generated by the model.

**3.4.2 Retrieval Quality Assessment**

Assessing the quality of retrieval is fundamental to gauge the effectiveness of the context retrieval mechanism within the RAG framework. Metrics traditionally used in search engines, recommendation systems, and information retrieval are applied to evaluate the retrieval module's performance in RAG models. For assessing retriever component we have two main metrics in RAGAs i.e context recall and context precision.

**3.4.3 Generation Quality Assessment**

Evaluating the generation aspect focuses on the generator's ability to create coherent and pertinent responses based on the provided context. This evaluation is bifurcated into assessments for unlabeled and labelled content. For unlabeled content, aspects such as the answers' faithfulness, relevance, and non-harmfulness are scrutinised. For assessing the generator component of the RAG module we have two main metrics: faithfulness and answer relevancy from RAGAs framework.

Moreover, evaluations of both the retrieval and generation components can be executed through manual or automated methods through the synthetic test data generation feature of RAGAs.

**3.5 Evaluation of RAG (Retrieval-Augmented Generation ) using RAGAs - On Human annotated and Synthetic Test Data Generation**

In our exploration of the Retrieval-Augmented Generation (RAG) application, we've unveiled its potential to leverage external knowledge for answering complex questions. The journey from a proof of concept to a robust real-world application, however, presents a unique set of challenges, particularly concerning the accuracy and reliability of the individual components within the RAG pipeline – the retriever and the generator.

Understanding the performance and pinpointing areas for improvement in the RAG pipeline is crucial. This is where a systematic evaluation becomes indispensable. It serves as a guiding tool to navigate the intricacies involved in optimising the RAG system.

The evaluation process is twofold: assessing each component independently and evaluating the system as a whole. This dual approach provides comprehensive insights into the system's performance and highlights specific areas needing refinement.

To facilitate this, two key elements are required:

* ***An Evaluation Metric***: A well-defined metric provides a quantitative measure of the system's performance, offering a way to compare different models or track improvements over time.
* ***An Evaluation Dataset:*** A robust dataset is essential for testing the system, ensuring that the evaluation metric provides meaningful and reliable feedback.

Identifying suitable evaluation metrics and curating an effective evaluation dataset are subjects of ongoing research within the field. To aid in this endeavor, we came across an evaluation framework known as RAGAs, specifically designed for RAG systems.

**RAGAs:** This framework is a comprehensive tool for evaluating RAG pipelines, offering insights based on the latest research findings. It can be seamlessly integrated into continuous integration/continuous deployment (CI/CD) pipelines, ensuring that performance checks are an integral part of the development process.

RAGAs necessitates the following for evaluation:

* Question: The user's query.
* Answer: The response generated by the RAG system.
* Contexts: The information retrieved to inform the answer.
* Ground Truths: Human-annotated correct answers, which are crucial for certain metrics like context recall.

RAGAs provides an array of features:

* Metrics and paradigms that necessitate ground truth labels, such as context recall and answer correctness.
* Tools for automatic test data generation, facilitating an easier evaluation process.
* Component-wise and end-to-end evaluation metrics, allowing for a detailed analysis of the system's performance.

Specifically, the Retriever Component is assessed using metrics like Context Precision and Context Recall, while the Generator Component is evaluated for Faithfulness and Answer Relevancy.

Through our exploration, we found the RAGAs framework incredibly beneficial in providing a structured and research-backed approach to evaluating the RAG system, ensuring each component's effectiveness and the overall system's alignment with real-world expectations.

In this context, we came across RAGAs, an evaluation framework tailored for RAG pipelines. This framework assists in providing a structured approach to assess the generated outputs, ensuring they align with factual accuracy and contextual relevance.

**Context Precision** is a metric pivotal in evaluating the RAG system, focusing on whether all relevant items in the context are ranked appropriately. This metric, which ranges from 0 to 1, essentially measures the utility of the provided context in deriving the answer.

For instance, given the question, "Who won the 2020 ICC World Cup?" if the context accurately details the event and its winner, the context precision would be high. Conversely, if the context discusses unrelated events, like the Andes mountain range in response to a query about the world's tallest mountain, the context precision would be low.

**Context Recall** examines how well the retrieved context correlates with the ground truth answer, aiming for a scenario where the retrieved context comprehensively supports the ground truth.

The **Faithfulness** metric gauges the factual accuracy of the generated answers based on the given context. For example, if a response about a student's major contradicts the provided context, it's marked as unfaithful.

**Answer Relevancy** assesses the pertinence of the generated answer to the posed question. This metric encourages answers that are directly responsive and complete, penalizing those that are tangential or redundant.

RAGAs framework simplifies the evaluation process, enabling a systematic assessment of the RAG components. It allows for the integration of these evaluations into continuous integration/continuous deployment (CI/CD) workflows, facilitating ongoing improvements.

Through our exploration, we utilised RAGAs to scrutinise a RAG system. The insights gained from this evaluation framework have been instrumental in refining the system, ensuring that the generated responses are not only contextually appropriate but also factually correct.

In the realm of ongoing research, RAGAs stands out by providing tools for automatic test data generation, a feature that significantly reduces manual labour in creating evaluation datasets. This capability, alongside the framework's component-wise and end-to-end evaluation metrics, offers a comprehensive toolkit for enhancing RAG pipelines, making it a cornerstone of our research in advancing the field of natural language processing.

**Synthetic Test Data Generation**

In the course of refining the Retrieval-Augmented Generation (RAG) application, the generation of synthetic test data using the Ragas framework emerged as a pivotal aspect of our research. The necessity for synthetic test data is underscored by the challenges inherent in manually crafting a substantial number of Question-Context-Answer (QA) samples, a process that can be both time-intensive and susceptible to limitations in complexity and diversity.

**Why Synthetic Test Data?**

The generation of synthetic test data addresses these challenges by automating the creation of QA samples, thus streamlining the evaluation process. This automation is crucial for developing a dataset that embodies the varied and intricate types of questions that a RAG system encounters in a real-world scenario, ensuring a comprehensive assessment of the system's capabilities.

**Ragas's Approach to Test Data Generation**

Distinctively, Ragas introduces an innovative approach to test data generation, integrating an evolutionary generation paradigm. This method systematically constructs questions embodying diverse characteristics, such as reasoning complexity, conditional context, and multi-context scenarios, thereby enriching the dataset with a spectrum of question types that reflect real-world complexities.

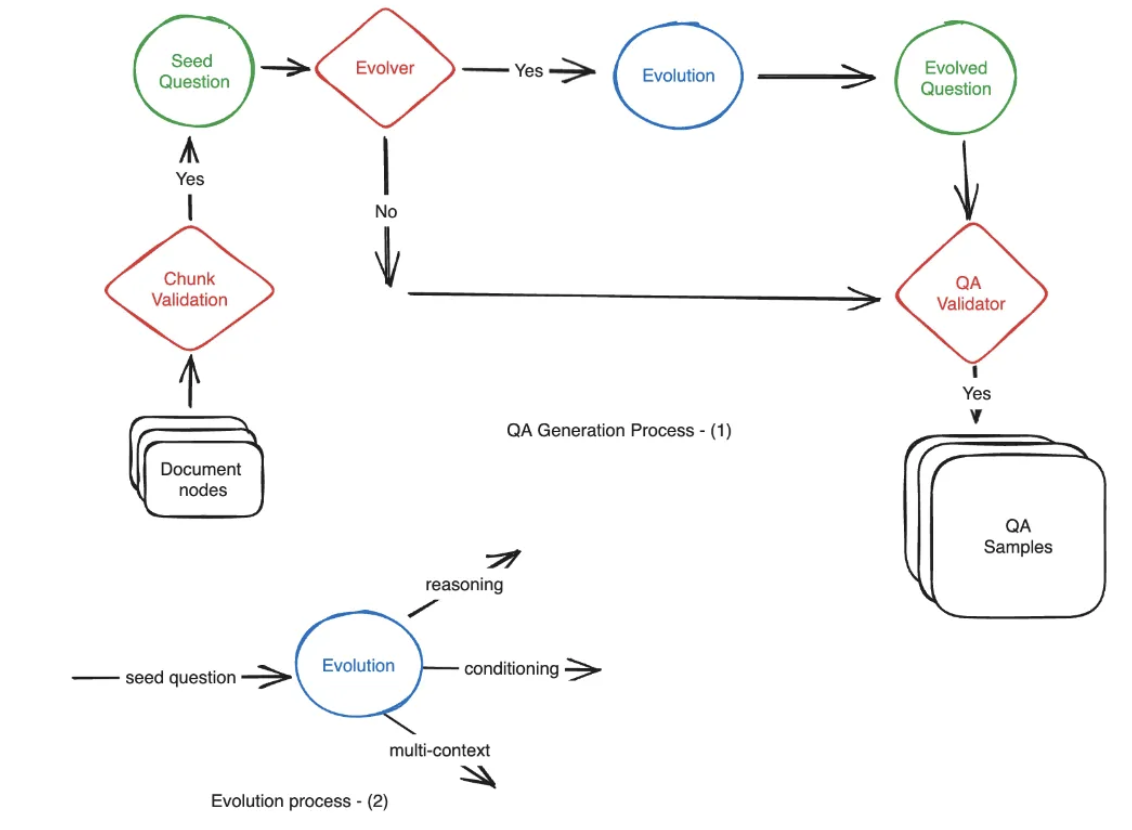
**Synthetic Test Data Generation Pipeline**

The pipeline for synthetic test data generation in Ragas operates as follows:

* **Evolutionary Generation Paradigm**: Questions are evolved through multiple iterations, incorporating filtering techniques to achieve the desired complexity level. This ensures the generation of questions that not only challenge the system's components but also mirror the varied demands encountered in production environments.

Through the integration of Ragas's synthetic test data generation into my research, we have significantly enhanced the evaluation process of the RAG system, enabling a more nuanced and thorough understanding of its performance across a spectrum of question complexities. This advancement not only streamlines the evaluation process but also substantially contributes to the ongoing enhancement and optimization of RAG systems.

Here is how the complete evolution module looks like.



We have applied this evaluation technique with different retrievers and compared their performances against the naive RAG. We have used **50 top files** from the **COVID19** dataset for the generation of synthetic Question and Answer Pairs, and their context and Relevance Ground Truth for Evaluation.

Large Language Models (LLMs) possess the capability to transform simple questions into more complex ones effectively. To generate medium to hard samples from the provided documents, we employ the following methods:

* **Reasoning:** Rewrite the question in a way that enhances the need for reasoning to answer it effectively.
* **Conditioning:** Modify the question to introduce a conditional element, which adds complexity to the question.
* **Multi-Context:** Rephrase the question in a manner that necessitates information from multiple related sections or chunks to formulate an answer.

The generated synthetic data looks like the following:  
{"messages":   
 [{"role": "user",   
 "content": "What is the purpose of fine-tuning in BERT and how does it relate to pre-training?"},   
 {"role": "assistant",  
 "content": "{\"Answers\": \"Fine-tuning in BERT refers to the process of taking a pre-trained BERT model and adapting it to a specific downstream task, such as text classification, named entity recognition, or in this context, abstractive summarization. Fine-tuning allows the model to learn task-specific information and nuances from a smaller dataset related to the downstream task. This process involves adjusting the weights of the pre-trained BERT model to better fit the specific task at hand.\n\nThe pre-training stage of BERT involves training the model on a vast amount of text data in an unsupervised manner. During pre-training, BERT learns general language representations and contextual relationships between words. This initial pre-training helps BERT understand the structure of language and semantics.\n\nThe fine-tuning process builds on this pre-training by allowing the model to adapt to specific tasks or domains. By fine-tuning, the model can adjust its learned representations to better suit the particular characteristics of the target task or dataset, leading to improved performance on that task. Fine-tuning essentially transfers the knowledge gained during pre-training to the specific downstream task, enabling the model to achieve better results by leveraging both general and task-specific information.\"}"}]  
}

We have generated a set of 50 questions on which all the advanced retrievers were evaluated.

**3.6 Summary**In chapter 3, we embarked on a comprehensive exploration of Retrieval-Augmented Generation (RAG) systems, focusing on their evolution from Naive RAG to more sophisticated Advanced RAG techniques. We began by examining the fundamental structure of RAG, which combines information retrieval with language generation to enhance the contextual understanding and factual accuracy of responses. In our methodology, we delved into various advanced techniques, including MultiQuery Retriever, Contextual Compression Retrieval, Sentence Window Retrieval, and Corrective Retrieval-Augmented Generation (CRAG). We implemented these methods using a range of tools and libraries such as Langchain, Llamaindex, and RAGAs, testing them on diverse datasets including COVID-19 information. To ensure rigorous evaluation, we developed a multi-faceted assessment approach, focusing on both the retrieval and generation components of RAG systems. We utilized metrics like context precision, context recall, faithfulness, and answer relevancy, integrating these evaluations into our continuous improvement pipeline. Recognizing the challenges in creating comprehensive test datasets manually, we leveraged the RAGAs framework to generate synthetic test data, employing an evolutionary paradigm to produce a wide spectrum of question complexities. Throughout our research, we consistently aimed to address the core challenges of RAG systems, including retrieval accuracy, context integration, and the mitigation of hallucinations in generated responses. Our methodology reflects a holistic approach to advancing RAG technology, combining theoretical analysis with practical implementation and rigorous evaluation to push the boundaries of what these systems can achieve in real-world applications.

**CHAPTER 4  
  
ANALYSIS AND EXPERIMENTS**

**4.1 Introduction**

In this chapter, we explore the analysis from the methods we outlined in Chapter 3. We will explore the key aspects of our methodology, including retrieval techniques, retriever models and Large Language Models (LLMs) as generators. We will delve into detailed analysis of dataset characteristics and the outputs generated by each methodological step, emphasizing their relevance in achieving our research objectives. By documenting our approach and insights gained, this chapter contributes to advancing techniques in retrieval augmented generation.

**4.2 Data ingestion**

For the purposes of our experiment, we have used the COVID-19 Open Research Dataset (CORD-19) dataset which is developed In response to the COVID-19 pandemic. The White House and a coalition of leading research groups have developed this dataset which comprises over 1,000,000 scholarly articles, including more than 400,000 with full text, on COVID-19, SARS-CoV-2, and related coronaviruses. This dataset is freely available to the global research community to utilize advances in natural language processing and other AI techniques to generate new insights to combat the infectious disease. The rapid increase in coronavirus literature has created an urgent need for these approaches, as it is becoming increasingly challenging for the medical research community to stay up-to-date.

Since Covid-19 has been a very big phenomenon, we felt the urge to utilise the dataset so that it can provide us with important insights on various aspects of the disease, its treatment and the various preventive measures that can be taken so that such a catastrophic event doesn’t take the form of a pandemic and attack us in the near future.

The complete dataset is of the size of 20 gb but we just took a small sample of 293 mb and text extract of 12,024 research papers for the purpose of our research. The text of the research papers has been extracted and stored in .txt files.

**4.2.1 Process of Data Extraction**

If we go through the complete structure of the CORD-19 dataset, we find the Multiple folders present, in which ‘Document Parses’ is the folder that we utilised for the purpose of our study. In ‘Document Parses’ folder, we have the folders ‘pdf\_json’ and ‘pmc\_json’.  
We utilised the .json files present in pdf\_json folder. these json files have the given structure:

- root

* -- paper\_id
* --- title
* ----authors
* -- abstract
* -- body text
* -- bib entries
* -- ref entries
* -- back matter

We just extracted the content that is present in the body text using python’s json library and stored that in the txt file.

**4.2.2 Creation of question-answers for fine tuning**

We used ragas ‘sythetic test set generator’ to create our own test dataset which comprises of the 5 columns – questions, context, gound\_truth, evolution\_type and episode\_done.

The resultant distributions were:

distributions = {

    simple: 0.3,

    multi\_context: 0.3,

    reasoning: 0.4

}

The total number of generated questions were 50. So, the number of simple, multi-context and resasoning questions were 14, 16 and 13.

**4.3 Chunking, embedding and storing the chunks in a vector database**

First, for chunking, we define a function split\_into\_chunks that takes a list of text pages and splits each page into smaller chunks of a size (chunk\_size) of 1500 with an overlap (chunk\_overlap) of 150 . This is achieved using the RecursiveCharacterTextSplitter class, which ensures that chunks are created with the specified parameters. The function then prints the number of original pages and the number of resulting chunks before returning the list of chunks.

After defining the function, the script removes any existing database files in the specified directory ('docs/chroma/') to start fresh. Then, it creates an embedding using the OpenAIEmbeddings class. This embedding is used to convert the text chunks into vector representations, which are then stored in a Chroma vector database. The database is persisted to the specified directory to save the state.

Finally, the script initializes a new OpenAIEmbeddings instance and reloads the vector database from the persisted files for future use. This ensures that the vector database can be accessed without having to re-process the documents.

For more clarity, the RecursiveCharacterTextSplitter and OpenAIEmbeddings classes are part of a text processing and vector database handling library, which helps in managing large text datasets and making them searchable through embeddings. The Chroma database is a type of vector database that allows for efficient storage and retrieval of high-dimensional vector representations of text.

**4.4 The Retrieval Augmented Generation Process**

In this section, we discuss the complete Retrieval Augmented Generation Process where we start with the creation of RAG Pipelines and then proceed to create Pandas Dataframe that contains the questions, answers, context and ground\_truth. Next, we send the values in each of these fields along with the to RAG pipeline and obtain the answers generated after the whole process. Once, the answers are obtained, it is saved in the same dataframe and sent to the RAGAS framework for evaluation on metrices like context\_precision, context\_recall, answer relevancy, faithfulness and answer correctness. We will go through each step in more detail.

**4.4.1 Creation of RAG Pipelines**



The function setup\_rag\_pipeline is designed to create a Retrieval-Augmented Generation (RAG) pipeline, which combines the capabilities of retrieval systems and language models to enhance the process of answering questions. Here's a complete understanding of the code provided:

The function takes in a retriever, a model name, and a temperature value. The retriever is responsible for fetching relevant contextual information based on a query. The model name specifies which language model to use, in this case, "gpt-3.5-turbo". The temperature value controls the randomness of the model's responses, with lower values making the output more deterministic.

First, the function initializes the language model (LLM) using the specified model name and temperature. It then defines a prompt template, which structures how the retrieved context and the question will be presented to the language model. This template instructs the model to answer the question using the given context and to admit if it does not know the answer.

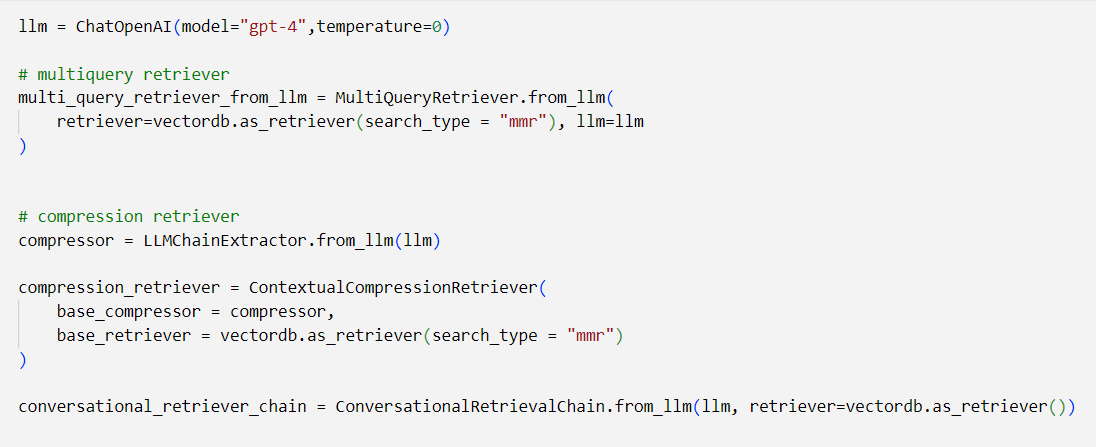
Then, the line retrieval = RunnableParallel({"context": retriever, "question": RunnablePassthrough()}) sets up a RunnableParallel object, which allows multiple tasks to run concurrently. In this context, it is used to process two different inputs: "context" and "question". The "context" is retrieved using the retriever and the "question" is simply passed through unchanged using RunnablePassthrough. This setup ensures that both the context retrieval and question handling happen simultaneously, improving the efficiency of the pipeline.

The next line, chain = retrieval | prompt | llm | StrOutputParser(), creates a sequential chain of operations using the LangChain Expression Language (LCEL). This chain processes the input in a specific order: first, the retrieval step gets the necessary context and question. Then, the prompt formats these inputs into a structured prompt template. The llm (Language Model) then processes this prompt to generate a response. Finally, the StrOutputParser converts the model's output into a string format suitable for use. By using the pipe (|) operator, each step's output becomes the input for the next step, creating a streamlined and efficient processing flow​

By integrating these components, the setup\_rag\_pipeline function creates an efficient and robust system for answering questions using retrieved information and a language model, ensuring that the responses are grounded in the provided context.

**4.4.2 Initializing various retrievers**

**Approach 1: Using a combination of multi-query retrieval, contextual compression, and a standard MMR-based retriever within an ensemble retriever setup**



A close up of a screen

Description automatically generated

A computer screen shot of a code

Description automatically generated

MMR, or Maximum Marginal Relevance, is a technique used in information retrieval and natural language processing to enhance the diversity of retrieved results by balancing relevance and diversity. It ensures that selected documents or responses are not only highly relevant to the query but also provide varied information, minimizing redundancy. This is achieved by evaluating the relevance of each candidate document to the query and its similarity to already selected documents, thus selecting items that add new information to the result set. In the context of the provided code snippets, MMR is applied through the vectordb.as\_retriever(search\_type = "mmr") configuration. This enhances the retrieval process by ensuring that the documents retrieved are both relevant to the user's query and diverse, providing a comprehensive and non-redundant set of results. This approach is particularly beneficial in complex retrieval systems, such as those combining multiple retrievers or document compression techniques, as it helps to deliver a more informative and useful response to the user's query.

For the purposes of our research, a sophisticated retrieval system is set up using various components to optimize the search and response process.

1. **Language Model Initialization**: The ChatOpenAI function is used that initializes a GPT-4 model with a temperature setting of 0, ensuring deterministic and consistent outputs.
2. **Multi-Query Retriever**: The MultiQueryRetriever.from\_llm function creates a retriever that uses multiple queries to improve search results by leveraging the GPT-4 model. This ensures that the search is thorough and comprehensive.
3. **Contextual Compression**: The LLMChainExtractor.from\_llm function initializes a compressor that processes and condenses retrieved documents. The ContextualCompressionRetriever then uses this compressor along with a base retriever to refine the search results, focusing only on the most relevant information.
4. **Conversational Retrieval Chain**: The ConversationalRetrievalChain.from\_llm function sets up a system for retrieving information in a conversational manner. This chain ensures that responses are contextually relevant and coherent in a dialogue format.
5. **Retrieval QA Chains**: Several RetrievalQA chains are set up to handle different types of queries:
   * qa\_multi\_query uses the multi-query retriever for comprehensive searches.
   * qa\_compress\_query uses the compression retriever to provide concise and relevant responses.
   * qa\_vectordb\_query and qa\_vectordb\_mmr\_query use vector database retrievers to fetch relevant documents, with the latter incorporating Maximal Marginal Relevance (MMR) to enhance diversity and relevance.

The chain\_type='stuff' parameter specifies the use of the StuffDocumentsChain, which takes a list of documents, formats them all into a prompt, and passes that prompt to an LLM (Large Language Model). It is called "stuff" because it "stuffs" all the documents into the prompt at once. This chain type is useful when the combined size of all documents fits within the context window of the LLM being used.

1. Finally, an **EnsembleRetriever** combines the strengths of multiple retrievers (multi-query, compression, and the basic vector database retriever). It merges and reranks the results from these retrievers to improve overall retrieval performance.

These components work together to create a robust system capable of handling various types of information retrieval tasks, ensuring accuracy, relevance, and efficiency. The integration of GPT-4 enhances the system's ability to understand and generate human-like responses, making it highly effective for complex queries​.

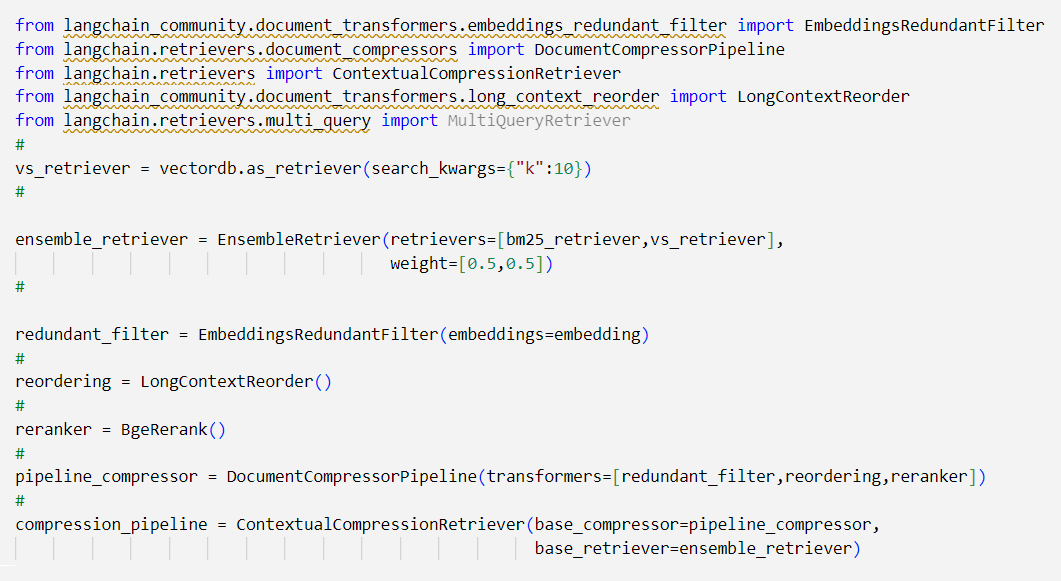
Firstly, a language model (LLM), specifically GPT-4, is initialized to process and understand the queries accurately. The process begins with setting up different retrievers, each designed to fetch information in a unique way. The multi\_query\_retriever handles multiple queries to gather diverse results, while the compression\_retriever focuses on compressing and filtering the information to ensure only the most relevant content is returned.

The ensemble\_retriever is a powerful feature that combines the strengths of these various retrievers. It is initialized with a list of these retrievers and uses a method called Reciprocal Rank Fusion to merge and rank the results from each one. This way, it ensures that the final output is a well-rounded and highly relevant set of documents. When you ask a question like "What is COVID-19?", the ensemble\_retriever sends this query to all its constituent retrievers, gathers their responses, and then intelligently combines them to give you the best possible answer.

By using such an ensemble approach, the system leverages the advantages of different retrieval methods, ensuring comprehensive and accurate results. This multi-faceted retrieval process makes it significantly easier to handle complex queries and large datasets, ultimately providing you with precise and valuable insights without the need for exhaustive manual searches.

This setup is particularly useful in scenarios where quick and accurate information retrieval is crucial, such as in research or data analysis tasks, ensuring that users can efficiently find the most pertinent information from vast amounts of data.

**Approach 2: Employs an ensemble of BM25 and vector space retrievers, combined with a document compression pipeline that includes redundancy filtering, context reordering, and re-ranking**



The provided code sets up an advanced document retrieval and compression pipeline using the Langchain library and its community extensions. Initially, various modules are imported, including filters for eliminating redundant information, compressors for document processing, and retrievers for fetching and organizing documents. This sets the foundation for a sophisticated retrieval system.

The vector database retriever (vs\_retriever) is configured to retrieve the top 10 documents relevant to a given query. This retriever utilizes vector embeddings to identify semantically similar documents. An ensemble retriever is then created by combining the results of a BM25 algorithm-based retriever and the vector database retriever, with equal weights assigned to both. This approach balances the strengths of traditional term-based retrieval and semantic similarity-based retrieval, enhancing the accuracy and relevance of the search results.

Subsequently, the code applies redundancy filtering using the EmbeddingsRedundantFilter, which leverages embeddings to remove redundant information and ensure diversity in the document set. The LongContextReorder transformer reorders document content to prioritize the most relevant sections, particularly beneficial for summarizing lengthy documents. The BgeRerank model, utilizing a cross-encoder architecture, reranks documents based on their relevance to the query, providing fine-grained relevance scoring.

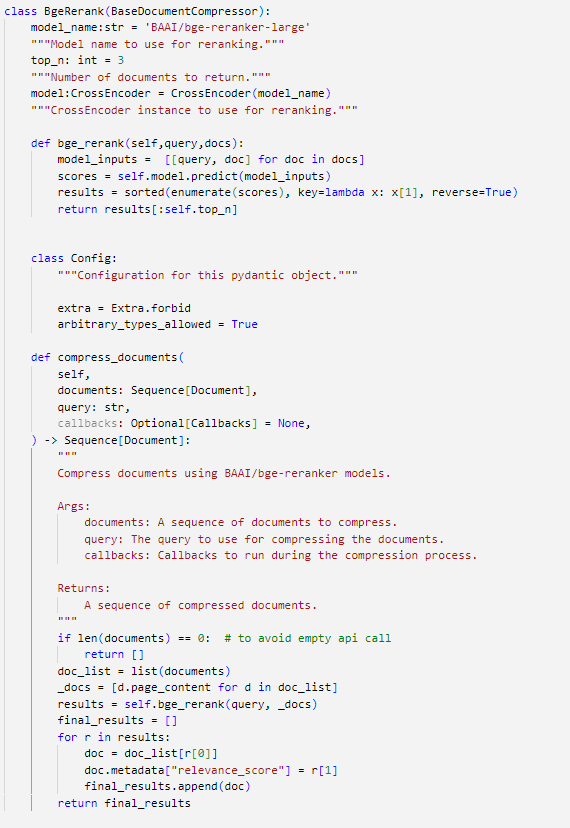
A document compressor pipeline is then established, chaining the redundancy filter, reordering transformer, and reranker. Each component sequentially processes the documents, progressively enhancing their quality and relevance. Finally, the ContextualCompressionRetriever combines this compressor pipeline with the ensemble retriever. This composite retriever first retrieves documents using the ensemble method and then processes them through the compression pipeline, resulting in a highly refined and contextually relevant set of documents.

This setup effectively integrates traditional and semantic retrieval methods, redundancy filtering, content reordering, and reranking to create a robust and efficient document retrieval system. The system significantly improves the relevance and quality of search results, making it ideal for applications requiring high-precision document retrieval, such as knowledge management systems and advanced search engines.

**4.4.3 Instantiating Reranking Model**

For purposes of our research, we used the BgeRerank model. The BgeRerank model, specifically the **BAAI/bge-reranker-large**, is a large language model designed for reranking tasks. This model uses a cross-encoder architecture, meaning it jointly encodes the query and document pairs and then outputs a relevance score for each pair. The model is optimized to understand the relationship between the query and documents, making it particularly effective in identifying the most relevant documents from a set of candidates.

In practical applications, the BgeRerank model first takes the input query and a list of documents. It creates pairs of the query with each document and processes these pairs through the cross-encoder to predict relevance scores. These scores are then used to rank the documents, with the top-n documents being selected based on their scores. This approach ensures that the most contextually appropriate documents are prioritized. The model is trained on a variety of datasets to generalize well across different domains and can handle complex query-document relationships effectively​.



The provided code defines a custom document compressor class BgeRerank that utilizes the BAAI/bge-reranker-large model for reranking documents based on their relevance to a query. The class inherits from BaseDocumentCompressor and leverages the CrossEncoder from the sentence\_transformers library to compute relevance scores for document-query pairs. The BgeRerank class is initialized with a model name, the number of top documents to return (top\_n), and an instance of CrossEncoder using the specified model. The bge\_rerank method constructs inputs for the model by pairing the query with each document, computes relevance scores using the model, and returns the top n documents sorted by their scores.

The compress\_documents method takes a sequence of Document objects, a query string, and optional callbacks as inputs.

In the LangChain library, the callbacks parameter in the compress\_documents method is an optional argument that allows for the integration of callback handlers during the document compression process. Callback handlers are objects that implement the CallbackHandler interface, which provides methods for various events that can be subscribed to during the execution of a process.

These callbacks can be used to track the progress and outcomes of different stages in the document compression pipeline. For instance, you can define actions to be taken when the process starts, when new tokens are generated, or when an error occurs. This is particularly useful for logging, debugging, and monitoring the behavior of your application in real-time.

In the context of LangChain, callbacks can be passed in two ways: during the object initialization (constructor callbacks) or during individual method calls (request callbacks). When provided, these callbacks ensure that specific actions are performed at designated points in the workflow, offering a flexible mechanism to extend and customize the functionality of document compressors and other components within LangChain​.

Inside the compress\_documents method, it first checks if the document list is empty to avoid unnecessary API calls. If documents are present, it extracts their content and calls the bge\_rerank method to get the relevance scores. The method then associates these scores with the corresponding documents' metadata and returns the top-ranked documents. The Config class within BgeRerank sets configuration options, ensuring no extra attributes are allowed and permitting arbitrary types. This setup allows for effective document compression and reranking, enhancing retrieval systems by focusing on the most relevant documents.

**4.4.4 Execution Of Multiple Retriever Chains**

We created a function return\_result(query\_list) which takes a list of queries (query\_list) as input and processes each query through multiple conversational retrievers.

A screenshot of a computer program

Description automatically generated

It initializes an empty result\_list to store the results. For each query in query\_list, it performs the following steps:

1. It initializes chat\_history to an empty list for maintaining conversational context.
2. It uses conversational\_retriever\_chain() to retrieve an answer based on the query and current chat\_history. It then appends the query and retrieved answer tuple to chat\_history.
3. It retrieves results from qa\_multi\_query(), qa\_compress\_query(), qa\_vectordb\_query(), and qa\_vectordb\_query\_with\_mmr() with MMR (Maximal Marginal Relevance), each of which processes the query independently.
4. It appends all results (including the original answer from conversational\_retriever\_chain()) into result\_list for the current query.
5. Finally, after processing all queries in query\_list, it returns result\_list, containing structured data for each query, including results from multiple retrievers and conversational contexts. This function is useful for comparing and analyzing the effectiveness of different retrieval methods for given queries.

Once created, we run the above function for simple, reasoning and muti-context queries and store the results in different dataframes.

**4.4.5 Evaluation and Comparison of Results using RAGAs**

We used RAGAs test set generator for generation of synthetic test dataset as discussed previously. The test set is obtained in the form shown below:

A screenshot of a computer

Description automatically generated

When using the test set generator to create synthetic test datasets, the column names in the generated test sets have specific meanings:

1. **questions**: This column contains the questions generated for evaluation purposes. These questions are designed to test various aspects of the LLM's performance.
2. **context**: This column provides the context or background information required to answer the questions. The context is typically extracted from documents or other data sources.
3. **ground\_truth**: This column holds the correct answers to the questions based on the given context. The ground truth serves as a benchmark to evaluate the accuracy of the LLM's responses.
4. **evolution\_type**: This column indicates the type of evolution or transformation applied to the original question to increase its complexity. Evolution types can include reasoning, multi-context, and conditioning, among others. These evolutions are used to create more challenging questions for the LLM.
5. **episode\_done**: This column signals the completion of a particular evaluation episode or interaction. It helps in organizing the dataset into distinct episodes for more structured evaluation.

These columns help in systematically assessing the LLM's capabilities by providing a diverse set of challenging and varied questions and contexts.

Next we append answers obtained from llm (gpt4, in our case) after supplying it with question and the source documents. Now, that we have our answers for each question, we append it to the end of our dataset.

A screenshot of a computer

Description automatically generated

For evaluating the answers generated using RAGAs, we just consider the four columns – question, context, ground\_truth and answers.

We perform the evaluation on five metrices:

**Faithfulness**:

* Faithfulness measures how accurately the generated answer aligns with the given context. It assesses whether the answer is directly supported by the provided information without introducing unsupported details or fabrications. High faithfulness indicates the answer strictly adheres to the context​.

**Answer Relevancy**:

* This metric evaluates how pertinent the generated answer is to the original question. It checks if the answer is complete and free of redundant information. Relevancy is measured by generating multiple artificial questions based on the answer and computing the mean cosine similarity between these questions and the original question. High relevancy indicates the answer directly addresses the question​.

**Context Recall**:

* Context recall measures how well the retrieved context aligns with the annotated ground truth answer. It is calculated based on the number of sentences in the ground truth answer that can be attributed to the retrieved context. Higher recall means more of the ground truth is covered by the retrieved context, indicating better alignment​.

**Context Precision**:

* Context precision evaluates whether all relevant items in the context are ranked higher in the retrieval process. It checks if the top-ranked chunks of context contain the necessary information as per the ground truth. Higher precision indicates that the most relevant context items are prioritized correctly​.

**Answer Correctness**:

* This metric assesses the factual accuracy of the generated answer compared to the ground truth. It combines semantic similarity and factual correctness, considering true positives (facts present in both the generated and ground truth answers), false positives (facts in the generated answer but not in the ground truth), and false negatives (facts in the ground truth but missing in the generated answer). Higher correctness scores reflect better factual alignment and semantic similarity with the ground truth​

The same process of evaluation is followed and same metrices are utilised for all the different types of Retrieval mechanisms like Baseline RAG or vanilla RAG, Multiquery Retrieval, Contextual Compression Retrieval, Sentence Window Retrieval.   
Next we perform finetuning of a generator model, i.e., llama 2 to analyse if the quality of answers get better with finetuning of the generator model and compare our results that we obtained using all the different retrieval mechanisms.

**4.5 Sentence Window Retrieval Process**

Before we move on to the process of fine-tuning, it is also imperative that we discuss how we performed the sentence window retrieval, which is a bit different from the architecture where we executed multiple retrieval chains that we discussed above.

We have used the llama-index library, since ‘sentencewindowretriever’ is not available in langchain and weaviate vectorstore client for purposes of our research.

1. **Initial Configuration**  
   **Setting Up OpenAI Models**: The code starts by configuring the Settings object from LlamaIndex to use specific models from OpenAI. The language model (llm) is set to gpt-3.5-turbo with a temperature of 0.1, making the responses more deterministic. The embedding model (embed\_model) is set to OpenAIEmbedding, which generates vector embeddings for the documents.
2. **Loading and Processing Documents**

**Loading Text Files**: The function load\_text\_files is designed to load up to 50 text files from a directory named "Covid Dataset". This function collects the file paths, which are then used to read the content of these files.

**Reading and Parsing Documents**: The SimpleDirectoryReader from LlamaIndex reads the content of the text files and loads them into a list of documents. Subsequently, SimpleNodeParser is used to chunk these documents into smaller sections (nodes) of 1024 characters each. This chunking is crucial for efficient indexing and retrieval, as it allows the system to handle smaller, more manageable pieces of text.

1. **Setting Up Weaviate and Vector Store**

**Connecting to Weaviate**: The code establishes a connection to a Weaviate instance, a powerful vector database. An embedded Weaviate client is used for simplicity, which runs the database locally without needing a separate server setup.

**Constructing the Vector Store**: A WeaviateVectorStore is created using the Weaviate client and a specified index name ("AdvancedRetrieval"). This vector store is responsible for storing the vector embeddings of the document nodes.

**Setting Up Storage Context**: A StorageContext is initialized with the vector store. This context manages the storage and retrieval of embeddings, ensuring seamless interaction with the vector database.

1. **Building and Querying the Index**

**Creating the Vector Store Index**: The VectorStoreIndex is built using the parsed nodes and the storage context. This index handles the chunking of documents, encoding them into embeddings, and storing these embeddings in the vector database.

**Setting Up the Query Engine**: The as\_query\_engine method of the index is used to create a query engine. This engine allows users to perform semantic searches on the indexed documents. For example, a query about COVID-19 and its impact on South Africa is submitted, and the engine retrieves the most relevant document chunk based on the stored embeddings.

1. **Enhancing Retrieval with Post-Processing**

**Creating sentence windows using SentenceWindowNodeParser :** SentenceWindowNodeParser class in LlamaIndex is used to process text by dividing it into smaller segments, or windows, based on sentences. This is achieved by first importing the SentenceWindowNodeParser from the llama\_index.core.node\_parser module. An instance of the parser is then created using the from\_defaults method, which initializes the parser with specified settings: a window\_size of 3 sentences, a window\_metadata\_key labeled as "window" for storing metadata related to the window, and an original\_text\_metadata\_key labeled as "original\_text" for referencing the source text from which each segment is derived. The parser segments a large document into manageable pieces, each tagged with metadata that helps maintain contextual information and facilitates detailed analysis or retrieval operations. This setup is particularly useful for natural language processing tasks where maintaining the context of smaller text chunks is crucial for effective analysis and reassembly of the original document.

**Metadata Replacement Post-Processor**: The MetadataReplacementPostProcessor is introduced to modify the metadata of the retrieved nodes. Its primary function is to replace specific metadata keys within nodes of text data based on specified criteria. This can be particularly useful in scenarios where the context or certain attributes need to be dynamically updated or corrected during the retrieval and query process. This post-processor focuses on metadata with the key "window", which likely refers to specific sections or contexts within the documents.

**4.5 Finetuning llama2 model**

There are two main Fine-Tunining techniques which are:  
1. Supervised Fine-Tuning (SFT)  
2. Reinforcement Learning from Human Feedback (RLHF)

**Supervised Fine-Tuning (SFT):**  
In Supervised Fine-Tuning the model is trained/fine-tuned on a dataset of instruction and responses. It adjusts the weights in the LLM to minimize the difference between the generated answers and ground truth responses, acting as labels.

We Fine-Tune the Llama 2 model using Supervised Fine-Tuning (SFT).

There are three ways in which we can fine-tune the model using Supervised Fine-Tuning (SFT).  
1. Full Fine-Tuning  
2. LoRA  
3. QLoRA

Full Fine-Tuning: With Full Fine-Tuning we are going to use the entire model, We will train all the weights in the model which is very costly.

LoRA: In LoRA instead of training all the weights, we will add some adapters in some layers and we will only train the added weights, which will reduce the cost of training the model because we are only training like 1% 2% of the entire weights

QLoRA: QLoRA which uses LoRA but here we use a model which has been quantized. If the LLM model is occupying 16bits on the disk, in QLoRA they will be quantized into 4bits so a lots of precision will be lose.

In our research, we tried to perform fine-tuning of the Llama 2 model having 7 billion parameters using T4-GPU which is provided with Google Colab notebook. The T4 GPU has 15GB of VRAM (GPU Memory) and is not enough to store Llama2-7b's weights (7b x 2bytes = 14GB in FP16), the overhead due to optimizer states, gradients and forward activatons.

To reduce the VRAM usage (GPU Memory Usage) we fine-tuned the Llama 2 model in 4bit precision using QLoRA.

We obtained the dataset used for fine-tuning using the RAGAs synthetic test set generator. Once the dataset is obtained, we perform train-test split on this dataset.

We have set up the following architecture for fine-tuning process for LLaMA-2 model on CORD-19 dataset:

We start by defining the base model as "NousResearch/Llama-2-7b-chat-hf", which serves as our foundation for fine-tuning. We name the fine-tuned version "llama-2-7b-platypus". Our dataset for fine-tuning is loaded from Hugging Face and is referred to as train\_dataset\_dict.

We load the tokenizer, a crucial component for processing text data, from the pre-trained base model using AutoTokenizer.from\_pretrained(base\_model, use\_fast=True). A significant issue with the LLaMA-2 tokenizer is the absence of a padding token, which we need when dealing with datasets containing rows of varying token lengths. To address this, we set the padding token to the end-of-sentence (EOS) token using tokenizer.pad\_token = tokenizer.eos\_token. This padding approach ensures uniform input lengths and affects the model's text generation capabilities. Additionally, we set the padding side to "right" with tokenizer.padding\_side = "right", which means padding will be added to the end of the sequences.

We start by configuring QLoRA to reduce VRAM usage, loading the model in 4-bit precision and performing quantization. We use the "nf4" format for quantization, introduced in the QLoRA paper, and store model weights in 4 bits. However, during computations, we use 16 bits for higher accuracy. We also enable double quantization of quantization parameters to further enhance efficiency.

Next, we set up the LoRA configuration. Instead of training all the model weights, we add adapters to certain layers and only train these added weights. We control the strength of these adapters with the lora\_alpha parameter, which we set to 15. This is quite strong, though 32 is often the standard value. We also apply a dropout rate of 10% and specify that no bias should be added. The task type is set to "CAUSAL\_LM" for causal language modeling.

We then load the base model with the quantization configuration and map it to our device, ensuring it's ready for efficient training. To optimize the model further, we disable cache usage during training and set the pretraining tensor parallelism to 1.

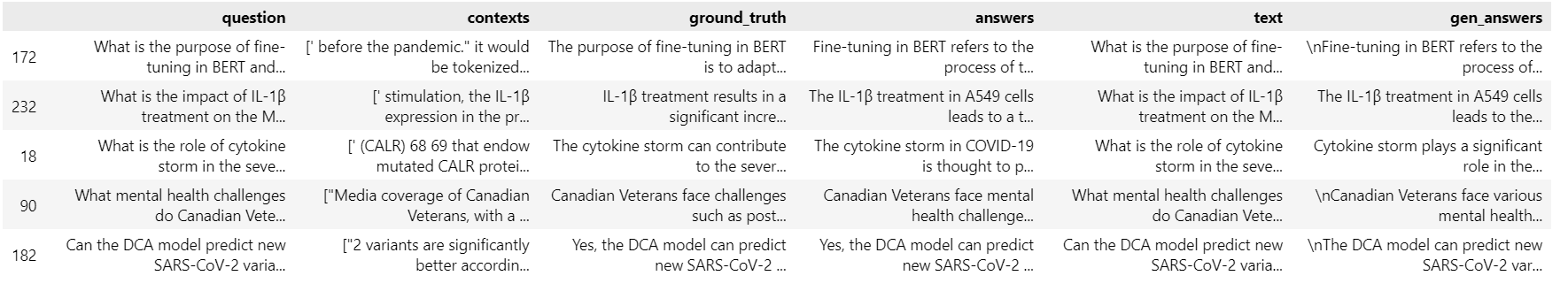
Finally, we prepare the model for K-bit training. This involves casting the layer normalization to 32-bit floating point, making the output embedding layer require gradients, and upcasting the language model head to 32-bit floating point. This preparation helps build the most effective model possible for training with reduced precision.

Then, we set up the training process by first defining the training arguments. We specify the output directory for the results and set the number of training epochs to 1, though values of 3 or 5 are also effective for the LLaMA-2 model. We set the batch size to 4, meaning we process four samples per training step. We accumulate gradients over one step to effectively manage the training process.

We choose an evaluation strategy based on steps, although our main focus is on training rather than evaluating the model. Evaluations occur every 1000 steps, and logging happens every 25 steps. We use the "paged\_adamw\_8bit" optimizer, a memory-efficient version of the Adam optimizer that operates in 8-bit precision. We set the learning rate to 2e-4, and use a linear scheduler for learning rate adjustments, with 10 warmup steps. The training progress is reported to TensorBoard, and we do not set a limit on the maximum number of training steps.

For supervised fine-tuning, we configure the SFTTrainer. We assign our model and the training dataset, and use the same dataset for evaluation. The peft\_config is integrated to include the necessary parameter-efficient tuning settings. We set the maximum sequence length to 512 tokens to manage VRAM usage, ensuring the training process remains efficient. The tokenizer is included to process the text data, and we pass the training arguments to control the training behavior.

We then train the model using the trainer.train() method, which initiates the fine-tuning process based on the defined parameters. After training, we save the fine-tuned model to the specified directory, ensuring the trained model is stored for future use.

The fine tuned results are obtained in the below format:  
  


We begin by reloading the model in FP16 (16-bit floating point) precision. This helps to reduce memory usage while maintaining computational efficiency. We use the AutoModelForCausalLM class to load the base model with low CPU memory usage and ensure it returns a dictionary of outputs. We also specify the model to use FP16 precision and map it to the appropriate device.

Next, we reload the base model and integrate the QLoRA adapters. We achieve this by using the PeftModel.from\_pretrained method to load the fine-tuned model with LoRA weights. We then merge and unload the model, combining the base model and LoRA weights into a single, efficient model.

Finally, we reload the tokenizer to ensure it is saved correctly. We use the AutoTokenizer.from\_pretrained method to load the tokenizer from the base model, with trust\_remote\_code set to true. We set the padding token to the end-of-sentence (EOS) token and specify the padding side to be "right", ensuring uniform padding is added to the end of sequences.

Ultimately, we use the RAGAs framework to evaluate the answers obtained from fine tuned model on the metrices: **Faithfulness, Answer Relevancy, Context Recall**, **Context Precision** and **Answer Correctness.**

**4.6 Summary**

As researchers, we conducted an extensive study on advanced Retrieval Augmented Generation (RAG) techniques and Large Language Model (LLM) fine-tuning, focusing on analyzing COVID-19 research data. We began by ingesting and preparing data from the COVID-19 Open Research Dataset (CORD-19), extracting text from 12,024 research papers and creating a synthetic test set using RAGAS for question-answer pairs. We then set up a sophisticated RAG pipeline, implementing chunking, embedding, and vector database storage. We created multiple retriever chains, including multi-query, contextual compression, and MMR-based retrievers, and employed an ensemble approach combining BM25 and vector space retrievers. To enhance our retrieval methods, we utilized various techniques such as multi-query retrieval, contextual compression, and sentence window retrieval, and incorporated the BgeRerank model for document reranking. We evaluated our results using the RAGAS framework, assessing metrics like faithfulness, answer relevancy, context recall/precision, and answer correctness across different retrieval mechanisms. In the final phase of our research, we explored Supervised Fine-Tuning (SFT) techniques for the LLaMA 2 model. We employed QLoRA (Quantized Low-Rank Adaptation) for efficient fine-tuning on limited hardware, successfully fine-tuning the 7 billion parameter LLaMA 2 model using 4-bit precision. We implemented LoRA configuration for parameter-efficient tuning and set up training arguments using SFTTrainer for supervised fine-tuning. After training, we reloaded and evaluated the fine-tuned model using RAGAS metrics. Through this comprehensive approach, we demonstrated the application of cutting-edge NLP methods in processing and analyzing COVID-19 research data, contributing to the advancement of RAG techniques in critical scientific domains.

**CHAPTER 5**

**RESULTS AND DISCUSSIONS**

**5.1 Introduction**

This chapter presents a comprehensive analysis of various Retrieval-Augmented Generation (RAG) techniques and their comparison with fine-tuning approaches in the context of Large Language Models (LLMs) based on the implementation of various retrieval mechanisms that we saw in the previous chapter. The study focuses on evaluating different RAG methods, including Baseline RAG, MultiQuery Retriever, Contextual Compressor Retriever, and Sentence Window Retriever between each other and against a fine-tuned Llama 2 model. The research utilizes a Covid-19 dataset for question-answering tasks and employs multiple metrics such as faithfulness, answer relevancy, context recall, context precision, and answer correctness to assess model performance.

**5.2 Metrics Comparison and analysis between Baseline and Advanced RAG Techniques**

In order to decide which technique is better for our use case, we need to analyse how different algorithms function and how accurate they are for our purpose. In our research we have used the Covid19 dataset for the question , answer and ground truth generation to evaluate the Naive RAG and Advanced RAG.

The swift progress and increasing utilisation of Retrieval-Augmented Generation (RAG) within Natural Language Processing (NLP) have elevated the assessment of RAG models to a critical area of focus within the Large Language Models (LLMs) community. This evaluation's main goal is to thoroughly understand and enhance the effectiveness of RAG models in various application settings.

**5.3 Baseline RAG(Vanilla RAG) vs MultiQuery Retriever**

|  |  |  |  |
| --- | --- | --- | --- |
| **Metric** | **Baseline** | **MultiQuery Retriever** | **Delta** |
| Faithfulness | 0.924540 | 0.978472 | 0.053933 |
| Answer Relevancy | 0.913075 | 0.814855 | -0.098220 |
| Context Recall | 0.980392 | 0.118627 | -0.861765 |
| Context Precision | 0.908742 | 0.022876 | -0.885866 |
| Answer Correctness | 0.615350 | 0.377078 | -0.238273 |

In this metrics table, we compare the performance of a baseline retrieval-augmented generation (RAG) model against a MultiQuery Retriever-enhanced RAG model across five key metrics: Faithfulness, Answer Relevancy, Context Recall, Context Precision, and Answer Correctness. Below is a detailed analysis and comparison of the results presented as percentage changes to elucidate the impact of employing a MultiQuery Retriever in the RAG framework:

* Faithfulness:

The baseline model achieves a faithfulness score of 0.924540, while the MultiQuery Retriever model scores 0.978472. This indicates an improvement of 5.3933%, suggesting that the MultiQuery Retriever model is more likely to generate responses that are aligned with the context provided.

* Answer Relevancy:

The baseline model has an answer relevancy score of 0.913075, whereas the MultiQuery Retriever model scores 0.814855. This represents a decrease of 9.8220%, indicating a reduction in the ability of the MultiQuery Retriever model to provide relevant answers to the queries.

* Context Recall:

Context recall drops significantly from 0.980392 in the baseline model to 0.118627 in the MultiQuery Retriever model, a decline of 86.1765%. This drastic reduction suggests that the MultiQuery Retriever model is less effective at retrieving all relevant information or context chunks.

* Context Precision:

There is a marked decrease in context precision, from 0.908742 in the baseline to 0.022876 in the MultiQuery Retriever model, amounting to a decrease of 88.5866%. This suggests that the MultiQuery Retriever model retrieves many irrelevant context chunks compared to the baseline.

* Answer Correctness:

Answer correctness decreases by 23.8273%, from 0.615350 in the baseline to 0.377078 in the MultiQuery Retriever model. This decrease indicates that the answers provided by the MultiQuery Retriever model are less accurate or correct compared to those of the baseline model.

These comparisons highlight the nuanced effects of integrating a MultiQuery Retriever into the RAG framework, showing improvements in some areas but declines in others. Such an analysis is critical for understanding the trade-offs involved and for guiding future improvements in RAG model development.

**5.4 Baseline RAG(Vanilla RAG) vs Contextual Compressor Retriever**

|  |  |  |  |
| --- | --- | --- | --- |
| **Metric** | **Baseline** | **Contextual Compression with Document Stuffing** | **Delta** |
| Faithfulness | 0.924540 | 0.962745 | 0.038205 |
| Answer Relevancy | 0.913075 | 0.917415 | 0.004340 |
| Context Recall | 0.980392 | 0.236928 | -0.743464 |
| Context Precision | 0.908742 | 0.016340 | -0.892402 |
| Answer Correctness | 0.615350 | 0.313366 | -0.301984 |

This metrics table compares the performance of a baseline RAG model to a RAG model enhanced with Contextual Compression with Document Stuffing across several evaluation criteria: Faithfulness, Answer Relevancy, Context Recall, Context Precision, and Answer Correctness. Below is an analytical breakdown and comparison of the results, presented as percentage changes, to elucidate the effectiveness of Contextual Compression with Document Stuffing in a RAG setup:

* Faithfulness:

The baseline model records a faithfulness score of 0.924540, whereas the model with Contextual Compression with Document Stuffing achieves 0.962745. This marks an improvement of 4.0825%, suggesting that the enhanced model generates responses more in line with the provided context.

* Answer Relevancy:

There's a slight increase in answer relevancy from 0.913075 in the baseline to 0.917415 in the enhanced model, a growth of 0.4340%. This minor improvement indicates that the enhanced model is marginally better at delivering relevant answers to the queries.

* Context Recall:

The context recall experiences a significant drop from 0.980392 in the baseline to 0.236928 in the enhanced model, a reduction of 74.3464%. This substantial decline points to the enhanced model's decreased capability in retrieving all relevant context chunks.

* Context Precision:

Context precision sees a drastic decrease from 0.908742 in the baseline to 0.016340 in the enhanced model, translating to a reduction of 89.2402%. This indicates that the enhanced model retrieves many irrelevant context chunks compared to the baseline.

* Answer Correctness:

Answer correctness decreases by 30.1984%, from 0.615350 in the baseline to 0.313366 in the enhanced model. This indicates a drop in the accuracy of the information produced by the enhanced model.

These results illustrate the nuanced impacts of incorporating Contextual Compression with Document Stuffing into RAG models. While there's a slight improvement in faithfulness and answer relevancy, the model sees a notable decrease in context recall, context precision, and answer correctness. This analysis is crucial for understanding the trade-offs involved and guiding further advancements in RAG model development.

**5.5 Baseline RAG(Vanilla RAG) vs Sentence Window Retriever**

|  |  |  |  |
| --- | --- | --- | --- |
| **Metric** | **Baseline** | **Sentence Window Retriever** | **Delta** |
| Faithfulness | 0.924540 | 0.910764 | -0.013776 |
| Answer Relevancy | 0.913075 | 0.940459 | 0.027384 |
| Context Recall | 0.980392 | 0.162232 | -0.818161 |
| Context Precision | 0.908742 | 0.019608 | -0.889134 |
| Answer Correctness | 0.615350 | 0.270894 | -0.344457 |

In this table, we're comparing the performance of a baseline Retrieval-Augmented Generation (RAG) model against a RAG model using a Sentence Window Retriever across various metrics: Faithfulness, Answer Relevancy, Context Recall, Context Precision, and Answer Correctness. The analysis, presented in percentage changes, provides insights into the impact of employing a Sentence Window Retriever in the RAG framework:

* Faithfulness:

The baseline model has a faithfulness score of 0.924540, while the Sentence Window Retriever model scores 0.910764. This indicates a decrease of 1.3776%, suggesting a slight decline in the model's ability to generate responses that are consistent with the provided context.

* Answer Relevancy:

Answer relevancy improves from 0.913075 in the baseline to 0.940459 in the Sentence Window Retriever model, marking an increase of 2.7384%. This improvement indicates that the Sentence Window Retriever model is more effective at providing relevant answers to the queries.

* Context Recall:

There's a significant drop in context recall from 0.980392 for the baseline to 0.162232 for the Sentence Window Retriever model, which is a decrease of 81.8161%. This major reduction points to the model's diminished ability to retrieve all relevant context chunks.

* Context Precision:

Context precision sees a drastic decrease from 0.908742 in the baseline to 0.019608 in the Sentence Window Retriever model, translating to a decrease of 88.9134%. This indicates a substantial increase in the retrieval of irrelevant context chunks compared to the baseline.

* Answer Correctness:

There's a decrease in answer correctness from 0.615350 in the baseline to 0.270894 in the Sentence Window Retriever model, a reduction of 34.4457%. This shows a notable decline in the accuracy of the information produced by the model with the Sentence Window Retriever.

These comparative results highlight the complex effects of integrating a Sentence Window Retriever into a RAG model. While there's an improvement in answer relevancy, significant declines are observed in context recall, context precision, and answer correctness, alongside a slight decrease in faithfulness. This detailed analysis is pivotal for understanding the specific impacts of the Sentence Window Retriever on RAG model performance and guiding future enhancements.

**5.6 Fine-tuning versus RAG in Generative AI Applications Architecture**

RAG and Fine-tuning are potent methodologies for refining Large Language Models (LLMs), and their combined application can cater to the demands of more nuanced scenarios. Fine-tuning, on one hand, enables the retrieval of documents characterized by distinct styles, enhancing semantic expression and bridging the gap between queries and documents. This process ensures that the retriever's output is more precisely tailored to the specific scenario. On the other hand, fine-tuning addresses the generation aspect by facilitating stylized and focused modifications. Additionally, fine-tuning can harmonise the retriever and generator components, fostering enhanced synergy within the model.

Fine-tuning involves taking a pre-trained language model and adjusting its parameters to make it more specialised in a specific domain or task. This is achieved by continuing the training process on a smaller, task-specific dataset.

Fine-tuning is generally less computationally intensive in terms of day-to-day operation. However, fine-tuning a model from scratch or on top of an existing model like **Llama 2**, still requires substantial computational resources for retraining, particularly for complex models.

The primary strength of fine-tuning is its adaptability. It can be applied to a broad range of tasks and domains, from text summarization to sentiment analysis.

Fine-tuning involves taking a pre-trained language model and adjusting its parameters to make it more specialised in a specific domain or task. This is achieved by continuing the training process on a smaller, task-specific dataset.

RAG essentially equips a model with a "textbook" for specific information retrieval, ideal for targeted queries, while Fine-Tuning (FT) is akin to a student absorbing knowledge progressively, which is more suited for emulating particular patterns, styles, or formats. FT enhances the model's performance and effectiveness by bolstering the foundational knowledge of the model, refining outputs, and facilitating the comprehension of intricate commands. However, it falls short in incorporating new knowledge or swiftly adapting to novel use cases.

While RAG and FT serve distinct purposes, they are not inherently incompatible; in fact, they can be synergistic, enhancing a model's proficiency at different levels. In some scenarios, leveraging both RAG and FT concurrently could lead to superior outcomes. Fine-tuning the integration of RAG and FT might require several iterations to attain the desired level of performance.

In this section, we delineate a comparative analysis between Retrieval-Augmented Generation (RAG) and fine-tuning approaches concerning various facets pertinent to their application in the Natural Language Processing (NLP) domain.

The main goal of fine-tuning the retriever is to improve the quality of semantic representations, achieved by directly fine-tuning the Embedding model using a corpus.Fine-tuning generator can result in outputs that are more stylized and customised.

**5.7 Fine-Tuning Llama 2 with QLoRA for Enhanced Efficiency**

In this study, we present an efficient approach to fine-tuning the Llama 2 model, which consists of 7 billion parameters, using Quantized Low-Rank Adaptation (QLoRA). Given the limitations posed by the VRAM of a T4-GPU, which offers 15GB of memory—barely sufficient to accommodate the Llama2-7b model in full precision—we adopt a quantization strategy to mitigate the memory constraints.

Our fine-tuning process leverages three key components: Full Fine-Tuning, Low-Rank Adaptation (LoRA), and Quantization (QLoRA), with a specific emphasis on QLoRA to optimize VRAM usage.

***Full Fine-Tuning***: Initially considered, this method was ruled out due to its high computational cost, as it requires training all model weights.

***LoRA:*** As a more resource-efficient alternative, LoRA introduces adapters into select layers of the model, significantly reducing the number of trainable parameters. This adaptation only affects a small fraction (1%-2%) of the total weights, thereby decreasing the training cost without substantially compromising performance.

***QLoRA:*** The chosen method, QLoRA, builds upon LoRA by incorporating quantization. Here, the model initially occupying 16 bits per weight is compressed to 4 bits, thus allowing the model to fit within the GPU's memory constraints, albeit with some loss in precision.

**Implementation Details**

***Model Configuration:*** The base model NousResearch/Llama-2-7b-chat-hf was fine-tuned to create llama-2-7b-platypus. For tokenization, we employed the tokenizer from Llama 2, setting the padding token to the end of sentence token to address the absence of a default padding token in Llama 2.

***Quantization Configuration:*** Utilising BitsAndBytes, the model was loaded in 4-bit precision (load\_in\_4bit=True), with the nf4 format for quantization and double quantization enabled (bnb\_4bit\_use\_double\_quant=True) to enhance the computation accuracy while maintaining a reduced memory footprint.

***LoRA Configuration:*** We set lora\_alpha to 15, indicating a significant weight to the adapters, and incorporated a 10% dropout rate to mitigate overfitting.

***Training Configuration***: Training involved a single epoch with a batch size of 4, leveraging the paged AdamW optimizer in 8-bit precision to further reduce memory usage. The learning rate was set to 2e-4 with a linear scheduler and a warmup period of 10 steps.

The fine-tuned model demonstrates a balanced trade-off between computational efficiency and model performance. By employing QLoRA, we were able to fine-tune the Llama 2 model within the constraints of a 15GB T4-GPU, showcasing the viability of this approach for large language models under limited hardware resources. The model is trained on 300 questions and answer pairs(from Covid19)Dataset, generated using the synthetic test data generation using RAGAs architecture and then on a similar set of questions were presented to the model to generate answers. A thorough comparison has been done as it is used as the generator component LLM to answer directly.

|  |  |  |
| --- | --- | --- |
| **Feature** | **RAG Approach** | **Fine-Tuning Approach** |
| Knowledge Updates | Facilitates real-time updates to the knowledge base without constant retraining, apt for dynamic data scenarios. | Requires retraining for updates, maintaining static data, less suited for rapidly changing environments. |
| External Knowledge | Efficiently utilises external databases, ideal for accessing various structured/unstructured data sources. | Aligns pre training knowledge with LLMs, less effective for continuously evolving data sources. |
| Data Processing | Minimal data handling is required, streamlining the process. | Relies on comprehensive, quality datasets for effective training, with limited datasets potentially impeding performance. |
| Model Customization | Concentrates on retrieval and external knowledge integration, with less emphasis on customising model traits. | Enables tailored adjustments to LLM behaviours, styles, or domain-specific knowledge, offering greater customization. |
| Interpretability | Offers higher interpretability through traceable responses linked to data sources. | Acts akin to a black box with lower transparency regarding model responses. |
| Computational Resources | Requires resources for retrieval processes and database management, including maintaining current data integrations. | Demands resources for dataset preparation, fine-tuning objectives, and computational support. |
| Latency Requirements | Involves retrieval steps that could increase latency. | Post fine-tuning, responds directly without retrieval steps, reducing latency. |
| Reducing Hallucinations | Less prone to hallucinations due to response grounding in retrieved data. | Reduces hallucinations by fine-tuning with domain-specific data but may still encounter issues with novel inputs. |
| Ethical and Privacy Concerns | Arises from the use and retrieval of data from external sources. | Stems from the handling of potentially sensitive content in training data. |

Comparison between RAG and Fine-Tuning

**5.8 Sentence Window Retriever vs Fine tuned LLama2**

|  |  |  |  |
| --- | --- | --- | --- |
| **Metric** | **SentencewindowRetriever** | **Finetunedonllama2** | **Delta** |
| faithfulness | 0.910764 | 0.760976 | -0.149788 |
| answer\_relevancy | 0.940459 | 0.926899 | -0.013560 |
| answer\_correctness | 0.270894 | 0.314780 | 0.043886 |

In this section, we compare the performance metrics of the original sentence window retriever model with the fine-tuned model, highlighting the percentage differences and their implications.

* ***Faithfulness:*** The fine-tuned model exhibits a decrease in faithfulness by approximately 16.45% compared to the original model. This decline suggests a potential trade-off between relevance and faithfulness, indicating that while the fine-tuned model may retrieve slightly less faithful information, it may prioritise other aspects such as relevance or efficiency.
* ***Answer Relevancy:*** While the fine-tuned model demonstrates a slight decrease of 1.44% in answer relevancy, this difference is relatively small. It suggests that the fine-tuning process has not significantly impacted the model's ability to retrieve relevant answers.
* ***Answer Correctness***: Notably, the fine-tuned model shows a substantial improvement of 16.19% in answer correctness. This indicates that despite potential trade-offs in other metrics, such as faithfulness, the fine-tuned model performs better in terms of providing accurate answers.

The comparison of performance metrics between the original and fine-tuned models reveals important insights into the impact of fine-tuning on model performance. While the fine-tuned model shows a decrease in faithfulness and a minor decrease in answer relevancy, it significantly improves answer correctness. These findings underscore the nuanced trade-offs involved in model optimization and highlight the importance of considering multiple performance metrics when evaluating model efficacy.

**5.9 Multi Query Performance vs FineTunedOnLlama2**

|  |  |  |  |
| --- | --- | --- | --- |
| **Metric** | **MultiQuery** | **FinetunedOnLlama2** | **Delta** |
| Faithfulness | 0.978472 | 0.760976 | -0.217497 |
| Answer Relevancy | 0.814855 | 0.926899 | 0.112044 |
| Answer Correctness | 0.377078 | 0.314780 | -0.062298 |

* ***Faithfulness:***The fine-tuned model demonstrates a substantial decrease in faithfulness by approximately 22.21% compared to the original MultiQuery model. This considerable decline suggests a significant trade-off between relevance and faithfulness in the fine-tuned model.
* ***Answer Relevancy***: In contrast to faithfulness, the fine-tuned model shows a notable improvement of 13.75% in answer relevancy. This enhancement indicates that the fine-tuning process has effectively improved the model's ability to retrieve relevant answers.
* ***Answer Correctness*:** However, there is a decrease of 16.52% in answer correctness for the fine-tuned model. This suggests that while the fine-tuned model may retrieve more relevant answers, it may sacrifice some accuracy in the process.

The comparative analysis of performance metrics between the original MultiQuery model and the fine-tuned model reveals important insights into the impact of fine-tuning on model performance. While the fine-tuned model demonstrates improvements in answer relevancy, it comes at the cost of decreased faithfulness and answer correctness. These findings emphasise the trade-offs inherent in model optimization and underscore the importance of balancing various performance metrics to achieve desired outcomes.

**5.10 ContextualCompressor Performance vs FineTunedOnLlama2**

|  |  |  |  |
| --- | --- | --- | --- |
| **Metric** | **ContextualCompressor** | **FinetunedOnLlama2** | **Delta** |
| Faithfulness | 0.962745 | 0.760976 | -0.201769 |
| Answer Relevancy | 0.917415 | 0.926899 | 0.009484 |
| Answer Correctness | 0.313366 | 0.314780 | 0.001413 |

* ***Faithfulness:*** The fine-tuned model demonstrates a notable decrease in faithfulness by approximately 20.96% compared to the original ContextualCompressor model. This decline suggests a potential trade-off between relevance and faithfulness in the fine-tuned model.
* ***Answer Relevancy***: While there is a slight improvement of 1.03% in answer relevancy for the fine-tuned model, this difference is relatively minor. It indicates that the fine-tuning process has marginally enhanced the model's ability to retrieve relevant answers.
* ***Answer Correctness:*** Similarly, there is a negligible improvement of 0.45% in answer correctness for the fine-tuned model. This suggests that the fine-tuning process has had minimal impact on the model's accuracy in providing correct answers.

The comparative analysis of performance metrics between the original ContextualCompressor model and the fine-tuned model highlights nuanced differences in model performance. While the fine-tuned model shows a decrease in faithfulness, it exhibits marginal improvements in answer relevancy and correctness. These findings underscore the complex trade-offs involved in model optimization and emphasise the need for comprehensive evaluation across multiple metrics.

**5.11 Baseline Performance vs FineTunedOnLlama2**

|  |  |  |  |
| --- | --- | --- | --- |
| **Metric** | **Baseline** | **FinetunedOnLlama2** | **Delta** |
| Faithfulness | 0.924540 | 0.760976 | -0.163564 |
| Answer Relevancy | 0.913075 | 0.926899 | 0.013824 |
| Answer Correctness | 0.615350 | 0.314780 | -0.300571 |

* **Faithfulness:** The fine-tuned model exhibits a significant decrease in faithfulness by approximately 17.68% compared to the original Baseline model. This reduction suggests a notable trade-off between relevance and faithfulness in the fine-tuned model.
* **Answer Relevancy**: Conversely, the fine-tuned model demonstrates a modest improvement of 1.51% in answer relevancy. This indicates that the fine-tuning process has enhanced the model's ability to retrieve relevant answers.
* **Answer Correctness**: However, there is a substantial decrease of 48.82% in answer correctness for the fine-tuned model. This suggests that while the fine-tuned model may retrieve more relevant answers, it significantly sacrifices accuracy in providing correct answers.

The comparative analysis of performance metrics between the original Baseline model and the fine-tuned model reveals significant trade-offs in model performance. While the fine-tuned model shows improvements in answer relevancy, it comes at the cost of decreased faithfulness and a substantial decline in answer correctness. These findings underscore the complexity of model optimization and emphasise the importance of carefully balancing various performance metrics to achieve desired outcomes.

**5.12 Summary**

The chapter begins by comparing Baseline RAG with advanced techniques like MultiQuery Retriever, Contextual Compressor, and Sentence Window Retriever. Each comparison reveals unique trade-offs in performance across different metrics. Generally, advanced techniques show improvements in faithfulness and answer relevancy but often at the cost of decreased context recall, precision, and answer correctness.

The study then explores the differences between RAG and fine-tuning approaches, highlighting their respective strengths and use cases. RAG is noted for its ability to incorporate real-time knowledge updates and external data sources, while fine-tuning excels in model customization and reduced latency.

A significant portion of the chapter is dedicated to the fine-tuning of the Llama 2 model using Quantized Low-Rank Adaptation (QLoRA), demonstrating an efficient approach to optimize large language models with limited computational resources.

The research concludes with a series of comparisons between the fine-tuned Llama 2 model and various RAG techniques. These comparisons consistently show that while the fine-tuned model often improves answer relevancy, it tends to sacrifice faithfulness and, in some cases, answer correctness.

Overall, the chapter provides valuable insights into the nuanced performance differences between various RAG techniques and fine-tuning approaches, emphasizing the importance of considering multiple metrics when evaluating and optimizing language models for specific applications.

**CHAPTER 6  
  
 CONCLUSIONS AND RECOMMENDATIONS  
   
6.1 Introduction**

This chapter consolidates the findings from the evaluation of various Retrieval Augmented Generation architectures like Retrieval mechanisms like Baseline RAG or vanilla RAG, Multiquery Retrieval, Contextual Compression Retrieval, Sentence Window Retrieval. It also analyses the effect of fine-tuning the generator model on the quality of answers generated. Additionally, it discusses the contributions of this research in advancing RAG architectures.

**6.2 Discussion and Conclusion**

The analysis of various RAG techniques and their performance against a baseline RAG model provides critical insights into the strengths and limitations of each approach.

1. **Baseline RAG vs MultiQuery Retriever**:
   * **Faithfulness**: The MultiQuery Retriever shows a notable improvement in faithfulness, suggesting its ability to generate responses closely aligned with the provided context.
   * **Answer Relevancy**: However, it suffers from a decrease in answer relevancy, indicating challenges in delivering relevant answers.
   * **Context Metrics**: Both context recall and precision drop significantly, highlighting inefficiencies in retrieving and selecting relevant information.
   * **Answer Correctness**: A reduction in answer correctness points to less accurate responses.
2. **Baseline RAG vs Contextual Compressor Retriever**:
   * **Faithfulness**: There is a slight improvement in faithfulness, demonstrating better alignment with context.
   * **Answer Relevancy**: A minor increase in answer relevancy shows marginal enhancements in providing relevant answers.
   * **Context Metrics**: The model sees substantial declines in both context recall and precision, suggesting difficulty in managing relevant information.
   * **Answer Correctness**: A decrease in answer correctness indicates less accurate responses.
3. **Baseline RAG vs Sentence Window Retriever**:
   * **Faithfulness**: A slight decrease in faithfulness reflects a minor reduction in consistency with context.
   * **Answer Relevancy**: An improvement in answer relevancy suggests better performance in providing relevant answers.
   * **Context Metrics**: Significant declines in context recall and precision highlight challenges in retrieving and selecting relevant information.
   * **Answer Correctness**: A notable decrease in answer correctness indicates less accurate responses.
4. **Fine-Tuning Llama 2 with QLoRA**:
   * The QLoRA approach balances computational efficiency and performance, enabling fine-tuning within memory constraints.
   * The fine-tuned model demonstrates improvements in specific metrics, showcasing the viability of this approach for large language models under limited hardware resources.
5. **Comparison between Different Models**:
   * The comparisons between fine-tuned and original models reveal nuanced trade-offs in model performance, with fine-tuning generally improving answer correctness but often at the cost of faithfulness and answer relevancy.

The study highlights the complex trade-offs involved in optimizing Retrieval-Augmented Generation (RAG) models and the impact of different retrieval techniques on model performance.

* **MultiQuery Retriever**: Shows potential for higher faithfulness but struggles with context recall, precision, and answer correctness.
* **Contextual Compressor Retriever**: Offers slight improvements in faithfulness and answer relevancy but faces significant challenges in context management.
* **Sentence Window Retriever**: Provides better answer relevancy but suffers from decreased context recall, precision, and answer correctness.
* **Fine-Tuning using QLoRA**: Demonstrates a balanced approach to model optimization under hardware constraints, with specific improvements in model performance.

Overall, the study underscores the importance of a comprehensive evaluation across multiple performance metrics to understand the trade-offs and guide future advancements in RAG model development. The nuanced impacts of different retrieval techniques and fine-tuning approaches highlight the need for careful consideration of specific use case requirements and resource constraints in model optimization efforts.

**6.3 Contribution to knowledge**

This study significantly contributes to the body of knowledge in the field of Natural Language Processing (NLP) by providing a comprehensive evaluation of different Retrieval-Augmented Generation (RAG) techniques and their performance metrics. Here are the key contributions:

1. **Comparative Analysis of RAG Techniques**:
   * The study provides a detailed comparison between baseline RAG models and advanced RAG techniques such as MultiQuery Retriever, Contextual Compressor Retriever, and Sentence Window Retriever. This comparison highlights the strengths and weaknesses of each technique in terms of faithfulness, answer relevancy, context recall, context precision, and answer correctness.
2. **Insights into MultiQuery Retriever**:
   * By evaluating the MultiQuery Retriever against the baseline RAG, the study reveals that while the MultiQuery Retriever improves faithfulness, it significantly decreases context recall and context precision. This finding is crucial for understanding the trade-offs involved in using MultiQuery Retriever in RAG frameworks.
3. **Evaluation of Contextual Compression with Document Stuffing**:
   * The study's analysis of Contextual Compression with Document Stuffing shows slight improvements in faithfulness and answer relevancy but significant declines in context recall, context precision, and answer correctness. This contributes to the understanding of how document compression techniques impact RAG model performance.
4. **Impact of Sentence Window Retriever**:
   * The evaluation of the Sentence Window Retriever demonstrates that it improves answer relevancy but results in substantial decreases in context recall, context precision, and answer correctness. This information is valuable for practitioners considering the integration of Sentence Window Retriever into
5. **Fine-Tuning Llama 2 with QLoRA**:
   * The research presents an efficient method for fine-tuning the Llama 2 model using Quantized Low-Rank Adaptation (QLoRA). This approach addresses memory constraints and demonstrates how large language models can be fine-tuned within limited hardware resources. This contribution is particularly relevant for practitioners working with resource-constrained environments.
6. **Trade-Offs in Model Optimization**:
   * The study emphasizes the nuanced trade-offs involved in model optimization. It shows that while fine-tuning can improve answer relevancy, it often comes at the cost of faithfulness and answer correctness. These insights are critical for guiding future improvements in RAG model development and for making informed decisions about model optimization strategies.
7. **Empirical Evidence for Model Selection**:
   * By providing empirical evidence on the performance of different RAG techniques and fine-tuning methods, the study equips researchers and practitioners with the necessary information to select the most appropriate model for their specific use cases.

Overall, this research advances the understanding of how different RAG techniques and fine-tuning methods impact the performance of large language models. It offers valuable insights and practical guidance for optimizing RAG models and enhancing their effectiveness in various application settings.

**6.4 Future Recommendations**

Currently, BM25 has been the probabilistic information retrieval model and is a part of the Okapi BM25 family. It ranks documents based on the query terms appearing in each document, regardless of their proximity within the document.  
The BM42 Retriever is a new retrieval system developed by Qdrant, a company known for its advanced vector database technologies. This retriever system is specifically designed to enhance the performance and accuracy of search tasks, particularly in the context of vector databases.

The BM42 Retriever works by leveraging a sophisticated combination of retrieval algorithms and advanced data structures. It uses a unique blend of traditional BM25-based methods and modern neural network techniques to provide highly accurate search results. This approach ensures that it can handle both structured and unstructured data efficiently, making it suitable for a wide range of applications, from information retrieval to recommendation systems.

BM42 builds on the BM25 model by incorporating neural network techniques to handle complex and high-dimensional data. This includes:

1. **Neural Embeddings**: Transforming text into dense vectors using pre-trained models like BERT (Bidirectional Encoder Representations from Transformers). Each document and query term is represented as a high-dimensional vector.
2. **Similarity Calculation**: Using cosine similarity or dot product to measure the similarity between query and document embeddings.
3. **Hybrid Score Calculation**: Combining BM25 scores with neural network-derived similarity scores. A weighted sum or another combination method can be used to balance the contributions from both models.

We can use BM42 retriever in our RAG architectures.

For the purpose of performing better Retrieval, there are two new concepts which seems to be promising: Speculative RAG and Multi-hop Question Answering.  
The research paper "Speculative RAG: Enhancing Retrieval Augmented Generation through Drafting" introduces a novel framework called SPECULATIVE RAG. This framework aims to improve the efficiency and accuracy of Retrieval Augmented Generation (RAG) systems, which combine the generative abilities of large language models (LLMs) with external knowledge sources.  
The methodology of SPECULATIVE RAG is designed to enhance both the accuracy and efficiency of Retrieval Augmented Generation (RAG) systems by decomposing the RAG tasks into two distinct stages: drafting and verification. Here's a detailed breakdown:

1. **Drafting with a Specialist LM:**
   * **Document Clustering and Sampling:** The system first retrieves a set of documents related to the query. These documents are then clustered based on content similarity. From each cluster, one document is sampled to form a subset, ensuring diversity and minimizing redundancy.
   * **Parallel Draft Generation:** A smaller, specialized language model (LM) is used to generate multiple draft answers in parallel. Each draft is based on a distinct subset of the retrieved documents, providing different perspectives on the evidence.
2. **Verification with a Generalist LM:**
   * **Single Verification Pass:** The drafts generated by the specialist LM are then verified by a larger, general-purpose LM. This LM performs a single pass to review all drafts and their rationales, calculating the conditional generation probability of the drafts to determine the most accurate and confident answer.
   * **Efficiency and Accuracy:** By delegating the drafting process to a smaller LM, the system reduces the computational load and input token counts per draft, thus mitigating potential biases and inefficiencies associated with long contexts. The generalist LM's verification ensures high-quality final answers.
3. **Performance Improvements:**
   * **State-of-the-Art Accuracy:** SPECULATIVE RAG achieves up to 12.97% improvement in accuracy by leveraging diverse perspectives and reducing redundancy in the drafts.
   * **Latency Reduction:** The framework significantly reduces latency by up to 51% compared to conventional RAG systems, thanks to the parallel draft generation and the efficient verification process.

The paper "Tree of Reviews: A Tree-based Dynamic Iterative Retrieval Framework for Multi-hop Question Answering" proposes a novel approach to multi-hop question answering called the Tree of Reviews (TOR). The framework employs a tree-like structure for dynamic iterative retrieval. The methodology involves constructing a tree with the initial question as the root and subsequent nodes as paragraphs retrieved from a corpus. Each node's paragraph undergoes evaluation to decide whether to continue retrieval, accept, or reject it. This tree structure allows the model to explore multiple reasoning paths, mitigating the negative impact of irrelevant information and reducing error propagation that is common in chain-like retrieval processes. The TOR framework integrates two optimization strategies: pruning, which minimizes unproductive searches, and effective expansion, which enhances query generation for better retrieval. Experiments conducted on three multi-hop question-answering datasets demonstrated that TOR outperforms existing methods in both retrieval accuracy and response generation.

For evaluation of LLMs, there is a new framework coming up called DeepEval, which can be utilized for RAG evaluation purposes.  
DeepEval is an open-source framework designed to evaluate large language models (LLMs) comprehensively and efficiently. It provides a wide array of evaluation metrics and is highly adaptable for different use cases. DeepEval supports the integration of various LLMs, enabling developers to assess and benchmark their models' performance across numerous criteria. Here are the primary metrics it offers:

1. **Answer Relevancy**: Evaluates how relevant the LLM's response is to the given input.
2. **Faithfulness**: Assesses if the output aligns factually with the provided context, focusing on avoiding contradictions rather than hallucinations.
3. **Summarization**: Measures the quality of generated summaries, ensuring they are comprehensive and accurate.
4. **Contextual Recall and Precision**: Examines the model's ability to recall and precisely use contextual information.
5. **Hallucination**: Identifies instances where the model generates information not present in the context.
6. **Toxicity and Bias**: Evaluates the generated content for harmful or biased language.
7. **Latency**: Measures the response time of the model, ensuring it meets performance requirements.
8. **G-Eval**: General evaluation metric that uses an LLM to provide an overall assessment of the output.
9. **RAGAS**: A specialized metric for retrieval-augmented generation systems.

DeepEval also supports synthetic dataset generation and can evaluate test cases in bulk using its CLI or directly through its Python API. It integrates seamlessly with CI/CD environments and tools like Hugging Face and LlamaIndex, facilitating real-time evaluation during model fine-tuning and production deployment. This flexibility and comprehensive metric suite make DeepEval a powerful tool for developers aiming to ensure the reliability and quality of their LLM applications​.

Also, Galileo AI can help in RAG evaluation as well.  
The Galileo LLM evaluation framework is designed to systematically assess and mitigate hallucinations in large language models (LLMs). It addresses common blind spots in existing benchmarks by focusing on LLM output quality, task variability, and context effectiveness. Galileo evaluates LLMs across three task types: question and answer without retrieval augmented generation (RAG), question and answer with RAG, and long-form text generation.

The framework uses several metrics to measure LLM performance and identify hallucinations:  
**1. Correctness**: Measures the factual accuracy of model responses.  
**2. Context Adherence**: Evaluates how well the generated response adheres to the given context.  
**3. Prompt Perplexity**: Indicates the model's confidence in predicting the text.  
**4. Output Uncertainty**: Reflects the model's confidence in its output tokens.  
**5. Context Similarity**: Assesses the relevance of the context to the question.  
**6. Answer Relevance**: Measures whether the generated response appropriately answers the question.  
**7. Groundedness**: Checks if the response is based on the provided context.

Additional metrics include **Data Error Potential (DEP)**, which identifies potential errors in training data, and **Context Relevance**, which evaluates the appropriateness of the context used in responses​

REFERENCES:

Long Ouyang, Jeffrey Wu, Xu Jiang, Diogo Almeida, Carroll Wainwright, Pamela Mishkin, Chong Zhang, Sand-hini Agarwal, Katarina Slama, Alex Ray, John Schulman, Jacob Hilton, Fraser Kelton, Luke Miller, MaddieSimens, Amanda Askell, Peter Welinder, Paul F Christiano, Jan Leike, and Ryan Lowe. Training language models to follow instructions with human feedback. In S. Koyejo, S. Mohamed, A. Agarwal, D. Belgrave, K. Cho, and A. Oh, editors, Advances in Neural Information Processing Systems, volume 35, pages 27730–27744. Curran Associates, Inc., 2022. URL <https://proceedings.neurips.cc/paper_files/paper/2022/file/b1efde53be364a73914f58805a001731-Paper-Conference.pdf.>

APPENDIX A: RESEARCH PROPOSAL

**Enhancing the results generated from Large Language Models using Retrieval Augmented Generation**

Rudranil Gupta

M.S in Artificial Intelligence and Machine Learning

Research Proposal

OCTOBER 2023

**Abstract:**

Retrieval Augmented Generation (RAG) is a technique in natural language processing that enhances generative AI models' accuracy and reliability by incorporating facts fetched from external sources. This approach combines the strengths of large language models (LLMs) with the precision of information retrieval systems, allowing the generation of more accurate and informative responses. Despite significant advancements, RAG faces challenges such as retrieval efficiency, sensitivity and the need for diverse, controllable retrieval metrics. In this study, we meticulously evaluate RAG's performance across three distinct frameworks: by utilizing baseline LLMs with standard retrieval models, by fine-tuning LLMs, and by optimizing retriever models. Using state-of-the-art LLMs and efficient retrievers, our research aims to achieve high accuracy in information retrieval from domain-specific datasets. The results hold significant promise for enhancing the efficiency and precision of RAG, thus improving quality of results generated for various Question Answering (QA) and Information Retrieval (IR) tasks.

**Table of contents**  
  
1. Background 5  
2. Problem statement 7  
 2.1 Retrieval Augmented Models 8  
 2.2 Pre-trained Retrieval Models 9  
 2.3 Latest Generator Models 10  
3. Aim and objectives 10  
4. Significance of the study 11  
5. Scope of the study 12  
6. Research methodology 12  
 6.1 Dataset Description 14  
 6.2 Data Preprocessing 14  
 6.3 Modelling Techniques 15  
 6.3.1 Using baseline RAG architecture 15

6.3.2 Fine-tuning the LLMs 16

6.3.3 Fine-tuning the retrievers 16  
 6.4 Evaluation Metrics 17  
7. Resource requirements 18  
 7.1 Hardware Requirements 18  
 7.2 Software Requirements 18  
8. Research plan 18  
References 19

**LIST OF FIGURES**

Figure 1: A pre-trained retriever (Query Encoder + Document Index) is combined with a pre-trained sequence-to-sequence model (Generator), and they are fine-tuned end-to-end. The top-K documents, denoted as 'zi', are identified using Maximum Inner Product Search for a given query 'x'. For the final prediction 'y', 'z' is treated as a latent variable, and marginalization is performed over the predictions of the sequence-to-sequence model, considering the variations presented by different documents.

Figure 2: Retrieval Augmented Generation architecture

Figure 3: Gantt chart showing the research plan

**LIST OF ABBREVIATIONS**

LLM Large Language Models

NLP Natural Language Processing

RAG Retrieval Augmented Generation

BART Bidirectional and Auto-Regressive Transformers

DPR Dense Passage Retrieval

REALM Retrieval-Augmented Language Model

kNN-LM k-Nearest Neighbour Language Model

FiD Fusion-in-Decoder

ICT Information Retrieval with Contrastive Learning

BERT Bidirectional Encoder Representations from Transformers

RoBERTa Robustly Optimized BERT Approach

ALBERT A Lite BERT

ColBERT Columnar BERT

GPT-4 Generative Pre-trained Transformer 4

LLaMA Large Language Model Meta AI

API Application Programming Interface

AWS Amazon Web Services

FAISS Facebook AI Similarity Search

MLOps Machine Learning Operations

BLEU Bilingual Evaluation Understudy

ROUGE Recall-Oriented Understudy for Gisting Evaluation

METEOR Metric for Evaluation of Translation with Explicit ORdering

NLTK Natural Language Toolkit  **1. Background**

Despite the impressive capabilities of LLMs in generating human-quality text and performing a variety of NLP tasks, they often lack up-to-date information or specific domain knowledge. RAG emerged as a solution to bridge this gap, providing LLMs with access to external knowledge sources for more accurate and reliable responses. The evolution of RAG can be traced back to early information retrieval systems and question-answering systems developed in the 1970s. These systems used NLP to access text in specific domains, laying the groundwork for the text mining concepts that RAG builds upon.

The seminal paper (Lewis et al., 2020) that coined the term "RAG" was developed by Patrick Lewis and colleagues during his doctorate in NLP at University College London and while working for Meta at a new London AI lab. This work was inspired by a vision to integrate a retrieval index into a trained system, allowing it to learn and generate Contextually Rich Responses. The results showed how to make generative AI models more authoritative and trustworthy, leading to the widespread adoption and further development of RAG concepts.

RAG consists of three parts - a retriever, a generator and an indexed knowledge base of text documents like Wikipedia or CommonCrawl.

A retriever's primary role is to search for and retrieve relevant information from a vast array of external sources or databases. These sources could be anything from text documents, scientific papers, and websites to structured databases.

A generator is responsible for producing the final response or output that is presented to the user. It takes the information retrieved by the retriever and integrates it into the generation process. Once the retriever provides the relevant information, the generator analyses this data along with the original query. It then synthesizes this information, often combining it with its pre-existing knowledge (gained during its training phase) to create a comprehensive and contextually relevant response. The generator is usually a large language model capable of understanding and generating human-like text.

RAG models combine parametric memory, which refers to the knowledge that is stored in the parameters (weights) of a neural network, specifically in the generative model part of the RAG model (like BART) and non-parametric memory in RAG models refers to the knowledge that is stored outside the model parameters, specifically in the external documents or databases that the model queries during the retrieval phase.

The retrieval model, typically a Dense Passage Retriever (DPR), functions as a bi-encoder architecture. It encodes both the input query and documents into vector representations. The likelihood of retrieving a document zgiven an input xis calculated based on the dot product of their vector representations. The DPR is a fine-tuned BERT model, which computes the document and query representations.

Where η is a parameter that returns top-k distributions over text passages, d(z) is a dense representation of a document produced by a BERTBASE *document encoder*, and q(x) a query representation produced by a *query encoder*, also based on BERTBASE.

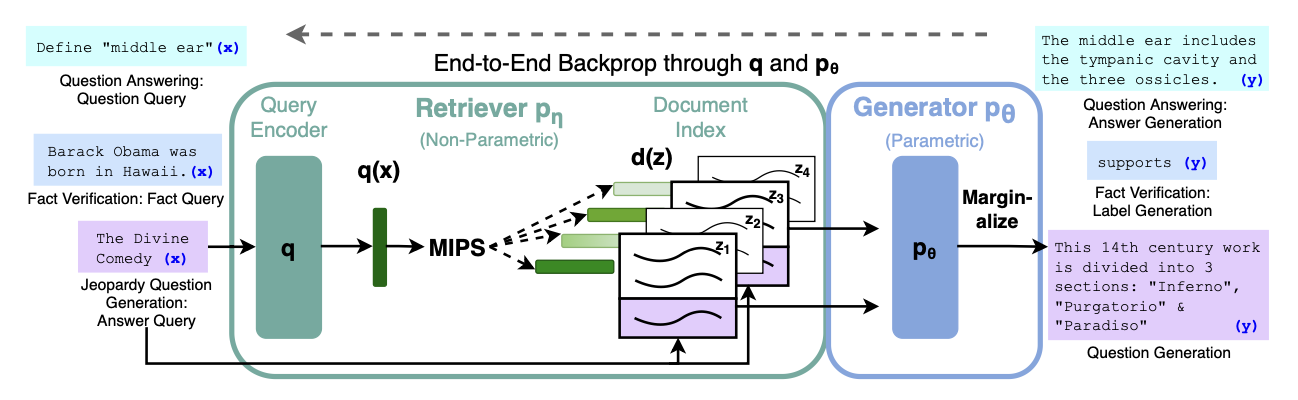


Figure 1: A pre-trained retriever (Query Encoder + Document Index) is combined with a pre-trained sequence-to-sequence model (Generator), and they are fine-tuned end-to-end. The top-K documents, denoted as 'zi', are identified using Maximum Inner Product Search for a given query 'x'. For the final prediction 'y', 'z' is treated as a latent variable, and marginalization is performed over the predictions of the sequence-to-sequence model, considering the variations presented by different documents. (Lewis et al., 2020)

The generator model can be any transformer model having encoder-decoder architecture with probability pθ(yi|x, z, y1:i−1) [parameterized by θ] of generating an output sequence y given the input sequence x, a current token based on a context of the previous i − 1 tokens y1:i−1 and the retrieved documents z.

The two types of RAG models are:

**RAG-Sequence Model:** The RAG-Sequence model utilizes a single document to generate the entire sequence. It views the chosen document as a crucial variable that helps calculate the probability of the sequence given the input (p(y|x)) through a top-K approximation method. Specifically, it first retrieves the top K documents. Then, for each of these documents, the model calculates the probability of the output sequence. These probabilities are combined (marginalized) to form the final sequence probability.

**RAG-Token Model:** Contrastingly, the RAG-Token model has the flexibility to select a different document for each token in the target sequence, allowing it to integrate information from multiple documents when formulating an answer. In practice, it retrieves the top K documents and then, for each token in the output sequence, the model predicts a distribution across these documents, which is then aggregated. This process repeats for each subsequent output token.

Both RAG Sequence and RAG Token models are trained by minimizing the negative log likelihood Σj − log p(yj |xj ) of generating the correct output sequences given the input sequences. The training process involves learning to retrieve the most relevant documents and use them effectively to generate accurate outputs.

To decode the output in the RAG-Token model, a standard beam decoder is employed. For the RAG-Sequence model, beam search is executed for each document *z*, with each potential answer (hypothesis) being evaluated using the probability function . This process generates a collection of potential answers, *Y*, among which certain answers might not be present in the beams of all documents. To calculate the probability of a specific hypothesis *y*, an extra forward pass is conducted for each document *z* that didn't initially include *y* in its beam. In this step, the probability given by the generator is combined with, and then these probabilities are aggregated across all beams to compute the marginal probabilities.

**2. Problem statement**

The existing challenges in Retrieval-Augmented Generation (RAG) are diverse and impact various aspects of its implementation and effectiveness. According to the paper titled ‘**A Survey on Retrieval-Augmented Text Generation**’ (Li et al., 2022), the current challenges are:

* **Retrieval Sensitivity**: The performance of RAG is highly sensitive to the quality of the retrieved information. Models can underperform when the retrieved examples are less like the query.
* **Retrieval Efficiency**: Expanding the retrieval memory for better accuracy can lead to reduced efficiency due to the increased computational overhead.
* **Local vs. Global Optimization**: The gap between retrieval metric optimization during training and inference poses challenges. This discrepancy affects the overall effectiveness of the model.
* **Diverse & Controllable Retrieval**: Current models often rely on universal metrics for retrieval, such as lexical similarities. There's a need for more customized retrieval metrics to enable more controlled and diverse text generation.
* **Multi-Modal Data Handling**: With advancements in multi-modal data retrieval, there's a need to efficiently integrate and process data from different modalities, like images and text, for enhanced RAG applications.
* **Scalability and Domain Adaptation**: Scaling RAG models to different domains and ensuring they adapt well to specific domain requirements remains a challenge.
* **Data Privacy and Security**: Ensuring the privacy and security of data used in retrieval processes, especially in sensitive applications.
* **Ethical Considerations and Bias**: Addressing potential biases in retrieved data and the ethical implications of using certain types of data for generation.

These challenges highlight the complex nature of RAG and the need for continued research and development to address these issues.

To transcend the limitations inherent in Retrieval-Augmented Generation (RAG) models, a multifaceted approach is imperative. Central to this endeavour is the refinement of retrieval algorithms, aiming for enhanced accuracy and efficiency. This refinement includes a judicious blend of term-based matching and vector similarity searches. Equally crucial is the mitigation of biases within the retrieval phase, necessitating a diversification of training data and the implementation of bias-correcting algorithms. The cornerstone of RAG's effectiveness, data quality and preparation, demands meticulous attention, encompassing cleaning, formatting, and the integration of comprehensive metadata to enrich contextual understanding.

The art of 'chunking', or the strategic segmentation of input text, plays a pivotal role. Here, experimentation with various chunk sizes and structures is vital to ensure that the retrieved texts precisely align with user inquiries, all while avoiding the pitfalls of informational noise. Addressing the computational complexity of these models calls for algorithmic optimizations, augmented by robust hardware solutions or cloud-based processing capabilities.

Navigating the murky waters of ambiguous queries requires an enhancement in the model's grasp of context and intent, potentially achieved through advanced natural language understanding techniques. The adoption of hybrid retrieval methods offers a nuanced, context-sensitive approach to information retrieval. Furthermore, the integration of continuous learning and updating mechanisms ensures the model's adaptability and relevance in the face of evolving information landscapes.

Tailoring the RAG system to specific applications, driven by targeted experimentation and customization, is essential for optimal performance in diverse domains. Finally, the incorporation of feedback mechanisms, where the outputs of the RAG system are continually evaluated and leveraged for ongoing refinement, solidifies its evolutionary path, steering it towards ever-greater accuracy, reliability, and efficiency.

Let us now take a look at the advancements that happened around the components that constitute the RAG models.

**2.1 Retrieval Augmented Models**

As of 2023, several retriever augmented models have been developed in the field of natural language processing, each with unique features and mechanisms:

* **REALM (Retrieval-Augmented Language Model)**: REALM is a masked language model augmented with a latent knowledge retriever. It employs a BERT encoder for dense representation of both the query and statements. The model selects the most similar statements based on a similarity metric and integrates them with a trained BERT-based language model for task solving. (Guu et al., 2020)
* **kNN-LM**: This model is based on a decoder-only Transformer and integrates token sequences from statements selected based on an L2 similarity metric between the query representation and all token sequences. The model computes the distribution over the next token in generation as an interpolation between the Transformer’s final distribution and the distribution of the next tokens over the retrieved statements. (Khandelwal et al., 2019)
* **FiD (Fusion-in-Decoder)**: FiD is a sequence-to-sequence T5-based neural network compatible with any retriever. It encodes the query and each retrieved statement separately, and the decoder attends over all the resulting representations. The model is often investigated with DPR (Dense Passage Retriever), which retrieves documents based on the inner product of BERT encoder model representations. (de Jong et al., 2022)
* **ATLAS**: ATLAS is a pre-trained retrieval augmented language model that jointly fine-tunes the retriever and the language model with minimal training examples. Its architecture resembles FiD’s, and it uses Contriever for retrieval—a dual BERT-based encoder architecture. The model selects documents based on the inner product of representations computed as the average hidden representations of the encoder’s last layer. (Izacard et al., 2022)

**2.2 Pre-trained Retrieval Models**

* **Dense Passage Retrieval (DPR):** DPR is a neural network-based model designed for retrieving relevant passages in response to a query. It uses a dual-encoder framework, with separate encoders for the query and the documents. These encoders are typically based on BERT or a similar transformer model.(Karpukhin et al., 2020)
* **Information Retrieval with Contrastive Learning (ICT):** ICT involves using contrastive techniques to improve document ranking and retrieval by training models to distinguish between relevant and irrelevant documents to a query. This approach can enhance the accuracy and relevance of search results in complex or specialized information retrieval tasks. (Izacard et al., 2021)
* **BERT for Information Retrieval:** BERT and its variants (like RoBERTa, ALBERT, etc.) have been adapted for information retrieval tasks. In these applications, BERT is used to generate dense embeddings for both queries and documents, improving the retrieval of semantically relevant results. (Abolghasemi et al., 2022)
* **ColBERT (Columnar BERT):** ColBERT is a retrieval model that uses BERT to create late interaction models. It generates lightweight representations of documents and queries, allowing for efficient retrieval over large document collections. (Khattab and Zaharia, 2020)

**2.3 Latest Generator Models**

As of 2023, the landscape of generator models in NLP is characterized by a range of advanced and versatile models, each with its unique features and capabilities:

* **GPT-4 (Generative Pre-trained Transformer 4)**: A multimodal large language model, GPT-4 is the latest in OpenAI's GPT series, known for its ability to predict the next token and fine-tuned with reinforcement learning. The GPT series has revolutionized NLP with its capacity to generate human-like text and perform a wide range of tasks like machine translation and text summarization. (OpenAI, 2023)
* **Vicuna :** The Vicuna model, specifically Vicuna-13B, is an open-source chatbot that has been developed by fine-tuning the LLaMA model using user-shared conversations collected from ShareGPT. It stands out for its performance, achieving over 90% of the quality of OpenAI's ChatGPT and Google Bard, as judged by GPT-4, and outperforming other models like LLaMA and Stanford Alpaca in most cases. (Chiang, 2023)
* **WizrdLM :** WizardLM is a high-performing Large Language Model with models ranging from 1B to 70B parameters, excelling in tasks like multi-turn conversation and mathematical reasoning. It employs an innovative Evol-Instruct training approach, achieving over 90% of ChatGPT's capacity in several skills and surpassing other models in areas like coding and human evaluations. (Xu et al., 2023)
* **Dolly :** Dolly 2.0 stands out as a versatile, instruction-following language model with a substantial parameter size, enhanced by a unique dataset for fine-tuning. Its open-source availability and adaptation of the InstructGPT architecture further contribute to its potential as a powerful tool in the realm of AI and language processing. (Conover et al., 2023)
* **Llama 2 :** LLaMA 2 by Meta AI stands out as a versatile and accessible large language model with a range of sizes and a strong emphasis on language processing capabilities. Its open-source nature and suitability for both commercial and research purposes make it a significant contribution to the field of AI and natural language processing. (Touvron et al., 2023)
* **Mistral 7B :** Mistral-7B-v0.1, Mistral AI's first LLM, is a decoder-based language model. It features architectural choices like grouped-query attention for faster inference and Sliding Window Attention to effectively handle large contexts and complex tasks. It is trained with an 8k context length, and has a fixed cache size, allowing for a theoretical attention span of 128K tokens. (Jiang et al., 2023)

**3. Aim and Objectives:**

The main aim of this research is to propose the best RAG model after studying the performance of RAG across three distinct architectural approaches:

1. Using baseline RAG architecture
2. Fine-tuning the LLMs to generate more accurate responses
3. Fine-tuning the retrievers to gain a deeper understanding of context

In our study, we will employ the advanced models outlined in the preceding section. A data pipeline based on the RAG framework will be constructed and fed with information from various external data sources for retrieval purposes. We will then evaluate the accuracy of the outcomes to determine if they align with the given context.

The research objectives formulated based on the aim of the study are as follows:

1. To enhance the quality and relevance of the input data by cleaning, normalizing, and structuring it in a way that maximizes the RAG model's ability to understand, interpret, and generate accurate and contextually appropriate responses.   
  
2. To investigate various chunk sizes and structures in text segmentation to optimize the alignment of retrieved texts with user inquiries, reducing informational noise.  
  
3. To develop and test advanced algorithms that blend information retrieval models and LLMs aiming to generate precise and reliable results.

3. To diversify training datasets and implement bias-correcting algorithms, aiming to mitigate biases during the retrieval phase.

4. To evaluate the precision and reliability of its outputs, ensuring that the generated responses are not only relevant and contextually appropriate but also grounded in accurate information derived from reliable external sources.

5. To evaluate the three different model architectures for implementing Retrieval Augmented Generation to determine which yields the highest accuracy.

6. To incorporate continuous learning and updating features in the RAG models, ensuring adaptability and relevance to evolving data landscapes.

7. To create and integrate feedback loops where the RAG system’s outputs are continually evaluated and used for ongoing system refinement, aiming for greater accuracy, reliability, and efficiency.

**4. Significance of the Study**

With the advent of large language models, there has been a transformative shift in natural language processing capabilities, enabling unprecedented achievements in tasks such as machine translation, text summarization, question-answering, and more. But one of the main issues for LLMs is hallucination and providing false information. It can be fatal for the user, if the LLM produces wrong answers for crucial questions pertaining to their medical diagnosis, medical reports or healthcare in general. Similarly, all major areas like finance, stock trading, business, technology, media, telecom or engineering can be affected due to misinformation. We can imagine the chaos that can be created if an AI chatbot or virtual assistant falsely impersonates a public figure, such as the Prime Minister, and shares information that violates public safety protocols.

For addressing these challenges, we propose the refinement of responses generated by LLMs using RAG techniques. This research is crucial to guarantee the responsible and secure implementation of large language models in critical applications.

**5. Scope of the Study**

Since the research is time-bound, we set out certain criteria boundaries for successful completion of our study.

Instead of using the complete dataset, we will take a sample of our data to train the RAG models which we will build.   
Due to resource and computation constraints, we will be using LLMs having 1 billion to 7 billion parameters instead of LLMs that have huge number of parameters like 34 billion or 70 billion. We shall also utilize the API keys for OpenAI models like GPT-3.5 and GPT-4 to fine tune our models.

Instead of hosting our models in cloud provider services like AWS and MS Azure, we will be using Google Colab and Kaggle notebooks.

We will be using vector databases like Pinecone, FAISS and Chromadb instead of using costlier alternatives used in production settings.

For deployment purposes, we shall use Streamlit and Flask.

**6. Research Methodology**

The complete architecture of our research is provided below:

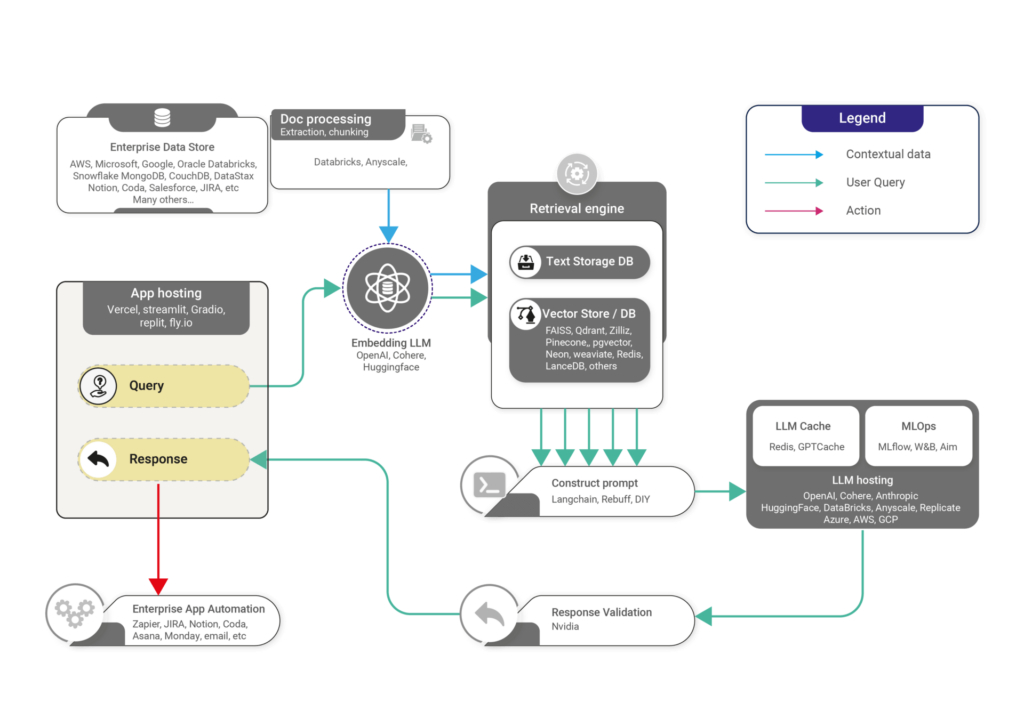


Figure 2: Retrieval Augmented Generation architecture

(Image Source: <https://vectara.com/a-reference-architecture-for-grounded-generation/>)

This image outlines the workflow of a typical language model-based application, from data storage to user interaction and back-end processing. Here's the flow described:

* **Enterprise Data Store**: Initially data is stored in various sources. The data can come from cloud services like AWS, Microsoft, Google, etc. The other likely sources of data are websites, databases like Oracle, PostgreSQL, etc., files stored in S3, DropBox, Google Drive, Microsoft OneDrive or on local storage, internal documents stored on SaaS applications in an organisation etc.
* **Document Processing**: This data then undergoes extraction and chunking, likely to prepare it for processing. Tools like Databricks and Anyscale are used.
* **Embedding LLM (Large Language Models)**: The processed data is sent to models from providers like OpenAI (the creator of GPT models), Cohere, Huggingface, etc., that embeds the text in processed document to vectors.
* **Retrieval Engine**: The embedded data is then interfaced with a retrieval engine, which can access either a **Text Storage Database** or a **Vector Store / DB**  where vectors (numerical representations of text), are stored in databases or vector stores such as FAISS, Pinecone, or Redis for later retrieval.
* **Construct Prompt**: A process is in place to construct prompts that will be used to generate responses, possibly using techniques like chain of thought or others. Frameworks like Langchain, Haystack etc. can be used at this stage.
* **LLM Hosting**: Once the query prompt and the context vectors are ready, both of them are sent to the LLMs to retrieve a coherent information. The LLM models can be hosted on cloud platforms like Google Cloud, Azure, AWS. **The other components are LLM Cache, which stores** cached data or responses from the LLM to improve response times for common queries and **MLOps which contains** Machine Learning Operations tools like MLflow to manage and orchestrate the machine learning lifecycle.
* **Response Validation**: This step involves validating the responses, possibly using tools or platforms like Nvidia.
* **App Hosting**: Finally, the whole application that interfaces with the user is hosted on platforms such as Vercel, Streamlit, Gradio, etc. Which provides easy UI for the user to interact with the system, issuing queries and receiving responses.
* **Enterprise App Automation**: This is an optional component to connect with enterprise applications through API and automate workflows using tools like Zapier, JIRA, Notion, etc.

This flowchart is typical for applications that leverage large language models to process user queries and return intelligent responses, integrating various back-end services for data storage, processing, and machine learning operations.

There are certain steps described in the below subsections which needs to be followed to successfully complete the research on RAG techniques.

**6.1 Dataset Description**

To address the challenges posed by the COVID-19 pandemic, a collaboration between the White House and various leading research organizations has led to the creation of the COVID-19 Open Research Dataset (CORD-19). This extensive dataset comprises over a million scholarly articles, of which more than 400,000 are fully accessible texts, focused on COVID-19, SARS-CoV-2, and related coronaviruses. Made freely available to researchers worldwide, CORD-19 aims to harness the latest advancements in natural language processing and other areas of artificial intelligence to foster new discoveries. This initiative is particularly vital due to the swift increase in coronavirus-related research publications, which poses a challenge for the medical research community in staying current.  
  
The CORD-19 dataset, currently the largest machine-readable collection of coronavirus literature for data mining, offers a unique chance for AI researchers around the world to employ text and data mining strategies. These tools can uncover answers within this vast body of work and link insights, aiding in the global response to COVID-19. The need for such tools is increasingly critical due to the fast-paced growth in coronavirus-related literature, which presents a challenge for the medical community in staying informed.

The dataset size is of 87.52 GB. A sample of the complete dataset consisting of the various medical articles on COVID, will be considered for our study.

**6.2 Data Preprocessing**

The preprocessing of textual data before it's sent to Large Language Models (LLMs) typically involves several steps to ensure the data is clean, structured, and suitable for processing. These steps include:

* **Data Cleaning**: Removing irrelevant items such as HTML tags, URLs, non-printable characters, or extraneous whitespace.
* **Tokenization**: Breaking down text into smaller units like words or subwords.
* **Lowercasing**: Converting all characters to lowercase to maintain consistency, unless case sensitivity is important for the context.
* **Removing Stop Words**: Omitting common words (like 'the', 'is', 'in') that may not be useful in some analyses.
* **Stemming and Lemmatization**: Reducing words to their base or root form. Stemming cuts off prefixes and suffixes, while lemmatization considers the context to convert words to their dictionary form.
* **Handling Special Characters and Punctuation**: Deciding whether to keep or remove punctuation based on the requirements of the task.
* **Correcting Spelling**: Automated spell check to correct typos and misspelled words.
* **Encoding Text**: Converting text into a format that’s understandable by the model, like using ASCII or Unicode encoding.
* **Chunking:** Dividing a text into syntactically correlated parts like phrases, paragraphs, sentences or custom methods decided by the user.
* **Vectorization**: Transforming text into numeric vectors using techniques like TF-IDF, Word2Vec, or contextual embeddings from models like BERT.
* **Sequence Padding or Truncation**: Adjusting text length to meet model input requirements by either padding short texts or truncating long ones.

Each of these steps helps refine the data for optimal performance with LLMs, ensuring that the input is clear, consistent, and meaningful for the specific task at hand.

**6.3 Modelling Techniques**

We will be creating a data pipeline like the one described in Research Methodology section. For our research, the dataset will be stored on our local system. We will execute processes for data extraction, cleaning, and chunking, followed by embedding the textual data. This embedded data will then be stored in a vector database. We'll forward the context and prompt to a Large Language Model (LLM) and assess the response for accuracy. Finally, the entire question-answering system will be incorporated into a user interface application, such as Streamlit, for ease of use and interaction.  
  
**6.3.1 Using baseline RAG architecture**

Here's a high-level overview of the RAG architecture:

1. **Retrieval Module:** RAG includes a retrieval component that is responsible for efficiently searching and retrieving relevant information from a large corpus of text. This module typically employs an information retrieval method, such as dense vector retrieval, to select a subset of passages that are likely to contain relevant information.  
    For our research, we will experiment with various retrieval modules like DPR and ColBERT.
2. **Augmentation Module:** The retrieved passages are then used to augment the input to the generation module. Instead of relying solely on the original input, the model incorporates information from the retrieved passages to enhance its understanding of the context and improve the quality of generated responses.  
    For our research, we will use different vectorstores like FAISS, Chromadb and Pinecone.
3. **Generation Module:** The augmented input is fed into a generation module, which is a language model capable of generating coherent and contextually relevant responses. This module could be based on transformer architectures, similar to those used in models like GPT (Generative Pre-trained Transformer).  
    For our research, we will use different LLM models like GPT-3, Mistral and models from huggingface.

By combining retrieval and generation components, RAG aims to address some limitations of purely generative models, such as the potential for generating incorrect or nonsensical information. The retrieval module helps in providing the model with access to factual and contextually relevant information from a broader knowledge base.

**6.3.2 Fine-tuning the LLMs**

To improve the accuracy of responses generated from LLMs, we will fine-tune the LLM with data of a specific domain.

Fine-tuning a language model involves several key steps. We will begin by selecting a pre-trained model that aligns with your task, such as GPT-3, GPT-2, or BERT. Then, we will prepare our dataset by formatting it to match the model's input requirements and tokenize the text data. After that, we have to load the pre-trained model, and modify its architecture to suit our specific task by adjusting or adding task-specific layers. We define a loss function that measures the difference between predicted and actual values. We will train the model on our task-specific dataset, using the pre-trained model as a starting point and updating weights during training. We have to experiment with hyperparameters like learning rate and batch size, and apply regularization techniques to prevent overfitting. We will also monitor the training process, evaluate performance on validation data, and save the fine-tuned model's weights and architecture. Optionally we will iterate through these steps, adjusting parameters based on performance feedback. Finally, we use the fine-tuned model for making predictions on new, unseen data related to your task.

**6.3.3 Fine-tuning the retrievers**

We start by selecting a pre-trained retriever model that suits the specific requirements of our task, often based on architectures like BERT or transformer models. We proceed by meticulously preparing our dataset to align with the input specifications of the chosen model, creating pairs of queries and relevant documents, including both positive and negative examples. Once the dataset is prepared, we load the pre-trained model and assess if any modifications to its architecture are necessary to better suit our retrieval task, possibly involving adjustments in final layers or the addition of task-specific layers. We then define a suitable loss function that accurately measures the similarity or relevance between queries and retrieved documents, such as contrastive or ranking loss. Moving forward, we train the retriever model on our task-specific dataset, leveraging the pre-trained model as a starting point and iteratively updating weights during the training process. Throughout this phase, we experiment with various hyperparameters, apply regularization techniques, and meticulously monitor the training process while continually evaluating performance on a separate validation set. We save the fine-tuned model for subsequent use in information retrieval tasks and, if necessary, iterate through these steps based on insights gleaned from performance feedback. Ultimately, we utilize the fine-tuned retriever model to retrieve pertinent documents for new queries within our specific information retrieval scenario.

**6.4 Evaluation Metrics**

RAG (Retrieval-Augmented Generation) models are typically evaluated based on their performance in both the retrieval and generation aspects. Here are several evaluation metrics commonly used for assessing the effectiveness of RAG models:

### 1. Retrieval Metrics:

* **Recall@k:** Measures the proportion of relevant passages retrieved among the top-k retrieved passages. A higher recall indicates better performance in retrieving relevant information.
* **Precision@k:** Measures the proportion of relevant passages among the top-k retrieved passages. It provides an indication of how precise the retriever is in selecting relevant information.
* **F1@k:** The harmonic mean of precision and recall at the top-k retrieved passages. It provides a balanced measure between precision and recall.

### 2. Generation Metrics:

* **BLEU (Bilingual Evaluation Understudy):** Measures the overlap between the generated text and reference text in terms of n-grams. Higher BLEU scores indicate better quality in generated language.
* **ROUGE (Recall-Oriented Understudy for Gisting Evaluation):** Evaluates the overlap of n-grams, word sequences, and word overlap between the generated and reference text. Common variants include ROUGE-N, ROUGE-L, and ROUGE-W.
* **METEOR (Metric for Evaluation of Translation with Explicit ORdering):** Incorporates precision, recall, and stemming-based matching to evaluate the quality of generated text.

### 3. Joint Metrics:

* **Retrieval-based Recall@k + Generation-based Metrics:** Some evaluation approaches combine the performance of the retriever and the generator. For example, measuring how well the generator incorporates information from the retrieved passages.
* **F1 Joint:** A combined metric that considers both retrieval and generation performance, offering a comprehensive evaluation of the RAG model.

**7. Resource Requirements**

**7.1 Hardware Requirements**  
  
The proposed research method will be implemented on a Macbook having Apple M2 chip which has a CPU clock rate of 3.49 GHz. Available RAM for the system is 16 GB. Similar results can be obtained using platforms like Google Colab and Kaggle notebooks.

**7.2 Software Requirements**

Python 3.12is a prerequisite for completing all the steps in our research.  
For data preprocessing, libraries like NLTK and Spacy are used. PDF parsing can be done using pymupdf library.

Machine learning libraries like numpy and pandas are used.

Pytorch is essential for building models. Models are downloaded from huggingface-hub.  
Langchain and haystack are LLM frameworks which are used for integrating features that are useful to complete our research.  
Vectorstores like Pinecone will be used in our research.

**8. Research Plan**

**A screenshot of a graph

Description automatically generated**

Figure 3: Gantt chart showing the research plan

**References**

Abolghasemi, A., Verberne, S. and Azzopardi, L., (2022) Improving BERT-based Query-by-Document Retrieval with Multi-Task Optimization.

Chiang, W.-L. and L.Z. and L.Z. and S.Y. and W.Z. and Z.H. and Z.L. and Z.S. and Z.Y. and G.J.E. and S.I. and X.E.P., (2023) *Vicuna: An Open-Source Chatbot Impressing GPT-4 with 90%\* ChatGPT Quality*.

Guu, K., Lee, K., Tung, Z., Pasupat, P. and Chang, M.-W., (2020) REALM: Retrieval-Augmented Language Model Pre-Training.

Izacard, G., Caron, M., Hosseini, L., Riedel, S., Bojanowski, P., Joulin, A. and Grave, E., (2021) Unsupervised Dense Information Retrieval with Contrastive Learning.

Izacard, G., Lewis, P., Lomeli, M., Hosseini, L., Petroni, F., Schick, T., Dwivedi-Yu, J., Joulin, A., Riedel, S. and Grave, E., (2022) Atlas: Few-shot Learning with Retrieval Augmented Language Models.

Jiang, A.Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D.S., Casas, D. de las, Bressand, F., Lengyel, G., Lample, G., Saulnier, L., Lavaud, L.R., Lachaux, M.-A., Stock, P., Scao, T. Le, Lavril, T., Wang, T., Lacroix, T. and Sayed, W. El, (2023) Mistral 7B.

de Jong, M., Zemlyanskiy, Y., Ainslie, J., FitzGerald, N., Sanghai, S., Sha, F. and Cohen, W., (2022) FiDO: Fusion-in-Decoder optimized for stronger performance and faster inference.

Karpukhin, V., Oğuz, B., Min, S., Lewis, P., Wu, L., Edunov, S., Chen, D. and Yih, W., (2020) Dense Passage Retrieval for Open-Domain Question Answering.

Khandelwal, U., Levy, O., Jurafsky, D., Zettlemoyer, L. and Lewis, M., (2019) Generalization through Memorization: Nearest Neighbor Language Models.

Khattab, O. and Zaharia, M., (2020) ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT. In: *SIGIR 2020 - Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. Association for Computing Machinery, Inc, pp.39–48.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W., Rocktäschel, T., Riedel, S. and Kiela, D., (2020) Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks.

Li, H., Su, Y., Cai, D., Wang, Y. and Liu, L., (2022) A Survey on Retrieval-Augmented Text Generation.

Conover, M., Hayes, M., Mathur, A., Xie, J., Wan, J., Shah, S., Ghodsi, A., Wendell, P., Zaharia, M., Xin, R. and Xin, R., (2023) *Free Dolly: Introducing the World’s First Truly Open Instruction-Tuned LLM*. <https://www.databricks.com/blog/2023/04/12/dolly-first-open-commercially-viable-instruction-tuned-llm>.

OpenAI, (2023) GPT-4 Technical Report.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., Bikel, D., Blecher, L., Ferrer, C.C., Chen, M., Cucurull, G., Esiobu, D., Fernandes, J., Fu, J., Fu, W., Fuller, B., Gao, C., Goswami, V., Goyal, N., Hartshorn, A., Hosseini, S., Hou, R., Inan, H., Kardas, M., Kerkez, V., Khabsa, M., Kloumann, I., Korenev, A., Koura, P.S., Lachaux, M.-A., Lavril, T., Lee, J., Liskovich, D., Lu, Y., Mao, Y., Martinet, X., Mihaylov, T., Mishra, P., Molybog, I., Nie, Y., Poulton, A., Reizenstein, J., Rungta, R., Saladi, K., Schelten, A., Silva, R., Smith, E.M., Subramanian, R., Tan, X.E., Tang, B., Taylor, R., Williams, A., Kuan, J.X., Xu, P., Yan, Z., Zarov, I., Zhang, Y., Fan, A., Kambadur, M., Narang, S., Rodriguez, A., Stojnic, R., Edunov, S. and Scialom, T., (2023) Llama 2: Open Foundation and Fine-Tuned Chat Models.

Xu, C., Sun, Q., Zheng, K., Geng, X., Zhao, P., Feng, J., Tao, C. and Jiang, D., (2023) WizardLM: Empowering Large Language Models to Follow Complex Instructions.

Abolghasemi, A., Verberne, S. and Azzopardi, L., (2022) Improving BERT-based Query-by-Document Retrieval with Multi-Task Optimization. [online] Available at: http://arxiv.org/abs/2202.00373.

Asai, A., Wu, Z., Wang, Y., Sil, A. and Hajishirzi, H., (2023) Self-RAG: Learning to Retrieve, Generate, and Critique through Self-Reflection. [online] Available at: http://arxiv.org/abs/2310.11511.

Chiang, W.-L. and L.Z. and L.Z. and S.Y. and W.Z. and Z.H. and Z.L. and Z.S. and Z.Y. and G.J.E. and S.I. and X.E.P., (2023) *Vicuna: An Open-Source Chatbot Impressing GPT-4 with 90%\* ChatGPT Quality*.

Conover, M., Hayes, M., Mathur, A., Xie, J., Wan, J., Shah, S., Ghodsi, A., Wendell, P., Zaharia, M., Xin, R. and Xin, R., (2023) *Free Dolly: Introducing the World’s First Truly Open Instruction-Tuned LLM*. https://www.databricks.com/blog/2023/04/12/dolly-first-open-commercially-viable-instruction-tuned-llm.

Guu, K., Lee, K., Tung, Z., Pasupat, P. and Chang, M.-W., (2020) REALM: Retrieval-Augmented Language Model Pre-Training. [online] Available at: http://arxiv.org/abs/2002.08909.

Izacard, G., Caron, M., Hosseini, L., Riedel, S., Bojanowski, P., Joulin, A. and Grave, E., (2021) Unsupervised Dense Information Retrieval with Contrastive Learning. [online] Available at: http://arxiv.org/abs/2112.09118.

Izacard, G., Lewis, P., Lomeli, M., Hosseini, L., Petroni, F., Schick, T., Dwivedi-Yu, J., Joulin, A., Riedel, S. and Grave, E., (2022) Atlas: Few-shot Learning with Retrieval Augmented Language Models. [online] Available at: http://arxiv.org/abs/2208.03299.

Jiang, A.Q., Sablayrolles, A., Mensch, A., Bamford, C., Chaplot, D.S., Casas, D. de las, Bressand, F., Lengyel, G., Lample, G., Saulnier, L., Lavaud, L.R., Lachaux, M.-A., Stock, P., Scao, T. Le, Lavril, T., Wang, T., Lacroix, T. and Sayed, W. El, (2023) Mistral 7B. [online] Available at: http://arxiv.org/abs/2310.06825.

de Jong, M., Zemlyanskiy, Y., Ainslie, J., FitzGerald, N., Sanghai, S., Sha, F. and Cohen, W., (2022) FiDO: Fusion-in-Decoder optimized for stronger performance and faster inference. [online] Available at: http://arxiv.org/abs/2212.08153.

Karpukhin, V., Oğuz, B., Min, S., Lewis, P., Wu, L., Edunov, S., Chen, D. and Yih, W., (2020) Dense Passage Retrieval for Open-Domain Question Answering. [online] Available at: http://arxiv.org/abs/2004.04906.

Khandelwal, U., Levy, O., Jurafsky, D., Zettlemoyer, L. and Lewis, M., (2019) Generalization through Memorization: Nearest Neighbor Language Models. [online] Available at: http://arxiv.org/abs/1911.00172.

Khattab, O. and Zaharia, M., (2020) ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction over BERT. In: *SIGIR 2020 - Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. Association for Computing Machinery, Inc, pp.39–48.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W., Rocktäschel, T., Riedel, S. and Kiela, D., (2020) Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks. [online] Available at: http://arxiv.org/abs/2005.11401.

Li, H., Su, Y., Cai, D., Wang, Y. and Liu, L., (2022) A Survey on Retrieval-Augmented Text Generation. [online] Available at: http://arxiv.org/abs/2202.01110.

OpenAI, (2023) GPT-4 Technical Report. [online] Available at: http://arxiv.org/abs/2303.08774.

Rackauckas, Z., (2024) RAG-Fusion: a New Take on Retrieval-Augmented Generation. [online] Available at: http://arxiv.org/abs/2402.03367.

Robertson, S. and Zaragoza, H., (2009) The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends in Information Retrieval*, 34, pp.333–389.

Sun, W., Yan, L., Ma, X., Wang, S., Ren, P., Chen, Z., Yin, D. and Ren, Z., (2023) Is ChatGPT Good at Search? Investigating Large Language Models as Re-Ranking Agents. [online] Available at: http://arxiv.org/abs/2304.09542.

Touvron, H., Martin, L., Stone, K., Albert, P., Almahairi, A., Babaei, Y., Bashlykov, N., Batra, S., Bhargava, P., Bhosale, S., Bikel, D., Blecher, L., Ferrer, C.C., Chen, M., Cucurull, G., Esiobu, D., Fernandes, J., Fu, J., Fu, W., Fuller, B., Gao, C., Goswami, V., Goyal, N., Hartshorn, A., Hosseini, S., Hou, R., Inan, H., Kardas, M., Kerkez, V., Khabsa, M., Kloumann, I., Korenev, A., Koura, P.S., Lachaux, M.-A., Lavril, T., Lee, J., Liskovich, D., Lu, Y., Mao, Y., Martinet, X., Mihaylov, T., Mishra, P., Molybog, I., Nie, Y., Poulton, A., Reizenstein, J., Rungta, R., Saladi, K., Schelten, A., Silva, R., Smith, E.M., Subramanian, R., Tan, X.E., Tang, B., Taylor, R., Williams, A., Kuan, J.X., Xu, P., Yan, Z., Zarov, I., Zhang, Y., Fan, A., Kambadur, M., Narang, S., Rodriguez, A., Stojnic, R., Edunov, S. and Scialom, T., (2023) Llama 2: Open Foundation and Fine-Tuned Chat Models. [online] Available at: http://arxiv.org/abs/2307.09288.

Xu, C., Sun, Q., Zheng, K., Geng, X., Zhao, P., Feng, J., Tao, C. and Jiang, D., (2023) WizardLM: Empowering Large Language Models to Follow Complex Instructions. [online] Available at: http://arxiv.org/abs/2304.12244.

Yan, S.-Q., Gu, J.-C., Zhu, Y. and Ling, Z.-H., (2024) Corrective Retrieval Augmented Generation. [online] Available at: http://arxiv.org/abs/2401.15884.

APPENDIX B: PYTHON CODE FOR RAG EVALUATION