Universiteit Leiden

Master Computer Science

Enterprise Question Answering System
on Technical Documents
with a Large Language Model

Name: Haoxiang Yuan
Student ID: s3988090
Date: 15/11/2024

Specialisation: Data Science

1st supervisor: Suzan Verberne
2nd supervisor: Zhaochun Ren

Master's Thesis in Computer Science

Leiden Institute of Advanced Computer Science
Leiden University
Niels Bohrweg 1
2333 CA Leiden
The Netherlands

Abstract

We present an approach to implement a question-answering system for enterprises's internal documents using large language models (LLMs), which also comes with capabilities to handle tabular data in the documents.

Modern enterprises often have vast amounts of internal documents, and employees need to quickly find relevant information. These documents often include a massive amount of tabular data, which carries important information that is not easily accessible through traditional search methods. Using centric high-performance infrastructure is not fesible for many enterprises considering the confidentiality and security of their data.

We use a retrieval-augmented generation (RAG) approach. Combining dense retrieval, sparse retrieval and reranking models, the application is capable of retrieving relevant document chunks from a local vector database. The retrieved chunks are then processed by a large language model to generate answers to user queries. We also implement a tabular data processing pipeline, which extracts and processes tabular data from the documents, allowing the application to generate more accurate and high-quality answers. Additionally, to evaluate the performance of our application, we create our own dataset from Wikipedia pages in technical domains, which includes a variety of document types and tabular data. Additionally, we develop a prototype application that integrates our question-answering system with a user-friendly interface, allowing users to interact with the system and ask questions about the documents.

Our experiment results show that our approach substantially improves the contextual precision, contextual recall, answer relevance and faithfulness of the answers generated by the question-answering system. Compared to a baseline RAG pipeline, our pipeline is able to handle tabular data more effectively, leading to more accurate and relevant answers.

This work demonstrates the potential of using LLMs and RAG techniques to build effective question-answering systems for enterprises, especially in scenarios where tabular data carries important information. The results suggest that our approach can substantially enhance the ability of employees to quickly find and utilize relevant information fron their documents, allowing them to improve their productivity and decision-making processes. The prototype application also provides a practical tool for enterprises to implement such systems, making it easier for employees to access and leverage internal knowledge.

Contents

1 Introduction

In large organizations, employees are often required to deal with a massive amount of technical documents, such as manuals, reports, and specifications, which contain valuable information essential for performing their tasks efficiently. For instance, new employees need to be familiar with the company's products and services by reading manuals and specifications, and software engineers consult technical documents to understand system architecture and design. A common scenario in such organizations is the need for collaboration among groups with different backgrounds and expertise. For example, software engineers may need to work closely with physicists to develop or improve a product, requiring deep understanding of physics principles, abbreviations, formulas, as well as software architecture, design, and implementation. In the meantime, the documents often contain a huge amount of tabular data, which is often used to present information more clearly and concisely. For instance, a technical document may contain a table that summarizes the specifications of a product, or values of a specific parameter in a system. These tables can be very useful for the employees to understand the system and make decisions. However, these tables are often scattered across different documents, and it is difficult to find the relevant tables that contain the information needed by the employees.

Therefore, finding relevant information within these large and complex documents can be a time-consuming and challenging task, especially when effective collaboration and knowledge sharing are crucial for success.

Additionally, in ASML, there are many regulations that restrict access to the documents. There is a huge amount of documents that are confidential and can only be accessed by the employees who have the permission, the documents should not be shared or uploaded to the cloud. Therefore, it is helpful and important to build a question-answering system that is able to ingest the user's own documents and generate answers based on the provided documents on the local machine.

Question-answering (QA) contains many techniques from the field of natural language processing (NLP), Information Retrieval (IR) and Information Extraction (IE). A typical search engine, such as Google, Bing, can be considered a QA system that can answer questions with documents from the web, i.e. given the search keyword from the user, the search engine can return the most relevant documents that contain the keywords. However, an ideal and user-friendly QA system should be able to answer questions with a natural language that can be seen as the answer directly to the user's question [1] [2].

In 2017, transformer [3] has been introduced as a new architecture for natural language processing tasks, which has shown great success in many NLP tasks. With its attention mechanism, the transformer can effectively capture the long-range dependencies in the text. This became the foundation for many state-

of-the-art models in NLP. In recent years, the development of large language models has made it possible to build such a question answering system. The transformer-based models, such as BERT [4], RoBERTa [5] and GPT-3 [6] can understand natural language and generate answers based on the user's query. With the retrieval-augmented generation (RAG) method, the system can retrieve relevant documents, and prompt the large language model to generate answers.

In this project, we aim to build a question answering system for enterprise, which can answer questions based on the organization documents. The system is built on top of a large language model, which can generate answers based on the user's query and the provided documents. The system consists of a frontend and a backend, which are responsible for the user interface and the data processing respectively.

- **RQ-1**: What is a good combination of retrieval methods for the RAG pipeline that can run on a regular laptop?
- **RQ-2**: How can we build a retrieval-augmented generation (RAG) pipeline that can handle tabular data effectively?
- **RQ-3**: What is the performance of the tabular data retrieval pipeline in our RAG application?

The main contributions of this work are as follows:

- We proposed a retrieval-augmented generation (RAG) pipeline for an enterprise question-answering system, with ability to handle tabular data effectively.
- We created a custom Wikipedia-derived dataset that contains tabular data, which can be used for end-to-end evaluation of the RAG pipeline.
- We benchmarked the performance of our RAG retrieval pipeline on the SciFact dataset, and compared it with different retrieval methods, including BM25, dense retriever, and fusion retriever.
- We conducted a series of experiments to evaluate the performance of our RAG pipeline, showing that our pipeline achieves higher scores on our custom dataset compared to the baseline approach.
- We developed a prototype application of the enterprise question-answering system, which can be deployed on a local machine and can answer questions based on the enterprise's internal documents.

The rest of the thesis is organized as follows. In Section 2, we will discuss the relevant literature in this domain. In Section 3, we will describe the datasets that we use in this project. In Section 4, we will describe the methods that we use in this project. In Section 5, we will describe the experiments that we conducted to evaluate the performance of our RAG pipeline. In Section 6, we will describe the prototype development of the enterprise question answering system. In Section 7, we will discuss the results of the experiments and the implications of the findings. Finally, in Section 8, we will conclude the paper and discuss the future work.

2 Related Work

We list the relevant literature in this chapter. In Section 2.2, we will discuss the work in retrieval-augmented generation, which is the core of our project. This project focuses on technical documents, which usually contain tabular data. Therefore, in Section 2.3, we will discuss the work in tabular data retrieval.

2.1 Question Answering

Question-answering is a widely studied task in natural language processing, the goal is to allow humans to ask questions to the existing knowledge base in natural language and get answers also in natural language. Back to last century, the topics were put up in computer science field [7, 8, 9] .

In the past decades, with the development of the computer science, especially the information retrieval and natural language processing, the question-answering system has been extended to text-to-SQL systems, which can convert natural language questions into SQL queries to retrieve relevant data from a structured database [10, 11, 12] .

There are always a huge amount of documents in modern enterprise and organization environments which are usually stored in various formats, such as PDF, Word, Excel, etc. With the rise of large language models, the question-answering system has been able to answer questions based on the content of these documents. However, these documents are not easily accessible to everyone in the organization, and they are often stored with access control and permission management [13]. An enterprise question-answering system must ensure that the sensitive information, such as patent, financial, and personal data, is securely stored and accessed during the retrieval and generation process. From the perspective of enterprise users, they often expect the question-answering system with higher quality, consistency, and interpretability compared to consumer-facing application [14].

2.2 Retrieval-Augmented Generation

There has been a lot of research on Retrieval-Augmented Generation models, which combine retrieval-based and generation-based approaches to question answering. In particular, the RAG model has shown that the retrieval-augmented model can outperform the generation-only model in knowledge-intensive NLP tasks [15]. Similarly, REALM [16] introduces a method that augments language models with a retrieval component, improving performance on open-domain QA tasks by retrieving relevant text documents during the pre-training process. Later, RAG with zero-shot prompting [17] has shown that the large language model can be used to answer questions on a wide range of topics without any fine-tuning.

As part of the RAG pipeline, the retrieval component is responsible for finding relevant documents from a large corpus. As popular and effective retrieval methods, BM25 [18] and TF-IDF [19] have been used in various retrieval-augmented generation pipelines. These are the most common sparse retrieval methods, which are based on the bag-of-words model and term frequency-inverse document frequency (TF-IDF) weighting.

Additionally, dense retrieval methods are also used in the RAG pipeline. For example, DPR [20] is a dense retrieval method that uses a bi-encoder architecture to encode the query and document into a dense vector space. Later, the dense vector space is used to retrieve relevant documents. Recently, the multilingual e5 embedding model [21] can be used to retrieve documents in multiple languages, which is useful for multilingual question answering tasks.

There has been a lot of metrics proposed to evaluate the performance of generated text in the natural language processing field. The most intuitive metric is the Exact Match (EM), which measures the percentage of generated answers that match the ground truth answers exactly.

BLEU (Bilingual Evaluation Understudy) is another popular metric that measures the similarity between the generated text and the ground truth text [22, 23]. Originally, BLEU is used to evaluate the quality of machine translation, by considering the correspondence between a machine's output and that of a human.

ROUGE (Recall-Oriented Understudy for Gisting Evaluation) is a set of metrics that measures the overlap between the generated text and the ground truth text, originally proposed for summarization tasks [24]. By comparing the n-grams of the generated text with the ground truth text, ROUGE can provide a more comprehensive evaluation in terms of the overlapping words and phrases against the ground truth text.

BERTScore is based on the pre-trained contextual embeddings from the BERT model [4, 25]. By generating the embeddings and calculating the cosine similarity, BERTScore can provide a semantic similarity score between the generated text and the ground truth text. This metric is particularly useful for evaluating the

quality of generated text in terms of semantic meaning, rather than just lexical overlap.

2.3 Tabular Data Retrieval

In addition to text retrieval, there has been research on tabular data retrieval. Traditional methods often use keyword or semantic search on the column and row names of the table [26, 27].

TINTIN [28] proposed a method to leverage the structural information in the table to retrieve relevant tables based on users' queries. Later, TableSeer [29] introduced a diagram for table detection, extraction, indexing, and ranking from PDF files.

Similar to word2vec [30], table2vec [31] proposed a method to embed tables into a dense vector space, which can be used to retrieve relevant tables based on the query. The works by Shraga et al. [32] and the work by Trabelsi et al. [33] also proposed methods for table retrieval based on the table embedding. They showed that the table embedding can be used to retrieve relevant tables based on the query and the table content.

TaPas [34] proposed a BERT-based pre-trained model to perform question answering on tabular data. Later, based on TAPAS, Herzig et al. proposed a dense retrieval method over tabular data [35], which is an end-to-end solution for tabular data retrieval and question answering.

ERATTA [36] proposed a SQL-based method to retrieve relevant data from a structural table based on the user's query. The method uses a SQL query to retrieve relevant data from the table, which can be used to answer the user's question.

More end-to-end solutions for tabular data retrieval and question answering have been proposed. For example, Pan et al. [37] proposed a transformer-based unified pipeline for table retrieval at the cell level.

3 Data

In this section, we describe the datasets that we use in this project, which are the SciFact, and our custom Wikipedia-derived dataset.

3.1 SciFact Dataset

This dataset is a scientific fact-checking dataset, which contains scientific claims verified against a corpus of abstracts. The task of this dataset is to verify the scientific claims based on the abstracts. The SciFact dataset comes from BEIR

[38], which is a heterogeneous benchmark containing diverse IR tasks. The SciFact dataset is divided into three splits: train, dev and test in the BEIR benchmark dataset.

We will use the test split of the SciFact dataset to evaluate the performance of the retrieval component in the RAG pipeline. The detailed description of the evaluation process is described in Section 5.2.

The SciFact dataset test split contains 1,409 scientific claims and a corpus of 5,183 abstracts. The statistics of the dataset are summarized in Table 1.

Table 1: Statistics of the SciFact dataset

Statistic	**Value**
Number of documents in corpus	5183
Number of queries	300
Number of qrels	300
Ave. number of documents per query	1.13
Ave. length of document	1401.08
Ave. length of query	90.34

It is worth noting that in the SciFact dataset, each query might have multiple relevant documents. As indicated in Table 1 by the average number of documents per query, which is 1.13. This means that while most queries have one relevant document, some queries have multiple relevant documents. Therefore, it requires the retriever to be able to retrieve multiple relevant documents for each query.

This also matches the real-world scenario, where informative text might be spread across multiple chunks. Therefore, the ability to retrieve top k relevant chunks is crucial for the RAG pipeline to generate accurate and informative answers.

To demonstrate the SciFact dataset, we provide an example of the query and the relevant documents in Table 2.

In this work, this dataset is used to evaluate the performance of the retrieval component in the RAG pipeline.

3.2 Custom Dataset from Wikipedia

This dataset will serve as the main evaluation set for the RAG pipeline in this project. It provides a realistic scenario for end-to-end evaluation of the RAG pipeline, including both document ingestion, retrieval, and answer generation.

In order to have a more realistic dataset for the application with end-to-end evaluation, we created a custom dataset from Wikipedia. The dataset is derived from technical Wikipedia pages, specifically focusing on topics on technical

Table 2: Example of the query and the relevant documents in the SciFact dataset

Field	Value
Queries	
Query ID	1
Query	0-dimensional biomaterials show inductive properties.
qrels	
Query ID	1
Document ID	31715818
Relevance	1
Documents	
Document ID	31715818
Title	New opportunities: the use of nanotechnologies to manipulate and track stem cells.
Text	Nanotechnologies are emerging platforms that could be useful in measuring, ...

products specifications, such as the details about Apple Silicon. The source material was downloaded from Wikipedia, and purged from the original websites and stored in HTML format to retain a semi-structured representation, including both textual content and tables.

From the downloaded data, we manually selected relevant sections and used ChatGPT to generate natural, human-like questions that might reasonably be asked about the content. We then prompted ChatGPT to generate multiple answers based on the provided content chunks, followed by a human review to ensure answer quality and correctness.

The resulting dataset consists of two main components: the source documents and the question-answer pairs. The source documents are stored in HTML format, which preserves the structure of the original Wikipedia pages, including both text and tables. The question-answer pairs are stored in a separate JSON file, which contains triplets of questions, generated answers, and the corresponding source documents.

After the source documents were collected and cleaned, we used ChatGPT to generate question-answer pairs. The prompt to generate the question-answer pair is listed in Appendix 1. The procedure for generating the question-answer pairs is as follows:

- Manually select relevant text and/or table segments from the cleaned documents that contain technical or factual information.

- Copy and paste the selected content into ChatGPT, along with a prompt instructing the model to generate realistic, human-like one-hop questions that one might naturally ask based on the given content.
- For each question, prompt ChatGPT to generate one or more answers grounded solely in the input chunk to simulate retrieval-augmented generation scenarios. We generate multiple candidate answers to ensure diversity and quality.
- Repeat the process for each selected content chunk, generating multiple candidate answers for each question.
- Review the generated answers by human annotators to ensure accuracy, relevance, and naturalness. The final dataset consists of tuples containing the question, the selected answer, and the name of the corresponding source document.

We collected 5 pages from Wikipedia, i.e. the pages with the following titles: 'Apple Silicon', 'Nvidia GeForce', 'Nintendo Switch', 'Intel Core' and 'ChatGPT'. For each page, we formulated 10 question-answer pairs, resulting in a total of 50 question-answer pairs. The topics were selected based on their technical nature, since the goal of these pages is to provide detailed information about a specific product. This nature of the pages makes them similar to the technical documents that employees in ASML would typically encounter.

Each entry in the question-answer dataset contains the following fields:

- **Question:** The question based on the content of the source document. The question is designed to be clear and answerable directly from the content of the source document, without requiring complex reasoning or multi-hop retrieval.
- **Answer:** The answer generated by ChatGPT and reviewed by human annotators based on the content of the source document. The answer is grounded in the source document and is expected to be accurate and relevant to the question.
- **Source:** The name of the source document from which the question and answer are derived.
- **SourceChunkType:** The type of the source chunk, which can be either *Text* or *Table*. This field indicates whether the question and answer are based on textual content or tabular data from the source document.

The following Table 3 provides an example of a question-answer pair from the custom dataset. The question is about the fabrication process used for the Apple A16 Bionic chip, and the answer is based on the information extracted from the source document. The statistics of the dataset are summarized in Table 4.

Table 3: Example Question-Answer pair from the custom Wikipedia-derived dataset

Field	**Value**
Question	How many transistors does the Apple M1 chip have?
Answer	The Apple M1 chip contains 16 billion transistors.
Source	apple_silicon.html
Source Chunk Type	Text

Table 4: Statistics of the custom Wikipedia-derived dataset

Statistic	**Value**
Number of Question	100
Ave. length of Question	65.24
Ave. length of Answer	48.84
Number of Table Questions	50
Number of Text Questions	50

Source Documents To create a clean and structured dataset suitable for downstream processing, we first extracted source documents from relevant Wikipedia pages. The raw HTML pages were downloaded, and subsequently parsed and processed using Python libraries such as BeautifulSoup. The processing pipeline focused on retaining informative content while removing unnecessary or noisy elements from the HTML. Specifically, we removed non-essential components such as images, scripts, styles, buttons, and figures. Hyperlinks were stripped down to plain text to simplify the content representation, and empty tags were removed to reduce clutter.

The result is a set of purged HTML documents preserving the core and important textual and tabular information in a semi-structured format. The cleaned HTML served as a foundation for question generation and answer grounding in the RAG evaluation pipeline.

Additionally, we also extracted a markdown version of the cleaned HTML documents without tabular data. This markdown version is used in the RAG pipeline to simplify the document ingestion process, where we aim to focus on the tabular data pipeline.

Question-Answer Pairs To generate a high quality evaluation set for our RAG system, we generated Question-Answer (QA) pairs from the cleaned Wikipedia pages.

First, we manually selected relevant text and/or table segments from the cleaned HTML documents that were likely to contain technical or factual information. These content chunks were then fed into ChatGPT with carefully designed prompts instructing the model to generate realistic, human-like questions that one might naturally ask based on the given content. To simplify the task and enforce clear answerability, we restricted generation to single-hop questions, which can be answered directly using a single chunk of text or table data without requiring information from multiple sources or complex reasoning. For each question, we prompted ChatGPT to generate one or more answers grounded solely in the input chunk to simulate retrieval-augmented generation scenarios.

To ensure the accuracy and naturalness, multiple candidate answers were generated and then reviewed by human annotators. The final dataset consists of tuples containing the question, the selected answer, and the name of the corresponding source document. The process enables a controlled interpretable evaluation of the RAG pipeline's ability to retrieve and generate grounded answers based on user queries.

4 Methods

The goal of this work is to build a RAG (Retrieval-Augmented Generation) application in an enterprise environment, which can answer questions based on the user's own documents. Following the enterprise compliance and security requirements, the system will be built to run on a local machine without any external network access. The system will be built on top of a large language model, which can generate answers based on the user's query and the provided documents. Moreover, this work consists of a tabular data retrieval pipeline that is specifically designed for retrieving relevant tables based on the user's query.

In this section, we will describe the main components of the RAG pipeline and the tabular pipeline.

4.1 Retrieval-Augmented Generation Pipeline

We built a RAG pipeline that can ingest the user's own documents and generate answers based on the provided documents.

The pipeline consists of three main components:

- Dataloader: This component is used to ingest the user's own documents and convert them into a format that can be used for the retriever (e.g. converting the documents into vectors using word embeddings and storing them in a database with their metadata).
- Retriever: This component is used to retrieve relevant documents based on the user's query. We will implement multiple retrieval methods, including BM25 and vector similarity, and even a fusion of these methods.
- Generator: This component is used to generate answers based on the retrieved documents. We will connect the generator to a large language model service such as Ollama to generate answers based on the retrieved documents.

4.1.1 Dataloader

The user provides a set of documents that they want to use for their own knowledge base (e.g. a set of technical documents, manuals, etc. in pdf, docx, or pptx format). In a typical scenario, the documents usually contain a lot of noise and irrelevant information. Besides, the user may ask questions that are relevant to tabular data in the documents. Therefore, we need to preprocess the documents to extract the relevant information and store them.

The documents are parsed and converted into text format. Then the text will be chunked into smaller pieces and stored in a database with its metadata (e.g. document id, chunk id, etc.). The metadata will also be used to retrieve the relevant chunks based on the user's query.

More formally, the dataloader can be defined as a function which can convert the user's documents into a set of chunks. Let D be the set of documents provided by the user, the dataloader can be defined as

$$L(D) = \{c_1, c_2, ..., c_n\},$$

where c_i is a chunk of the document.

The high-level structure of the dataloader is shown in Figure 1. The dataloader will first parse the documents and extract the text and tables from the documents. Then, it will chunk the text into smaller pieces and store them in a database with

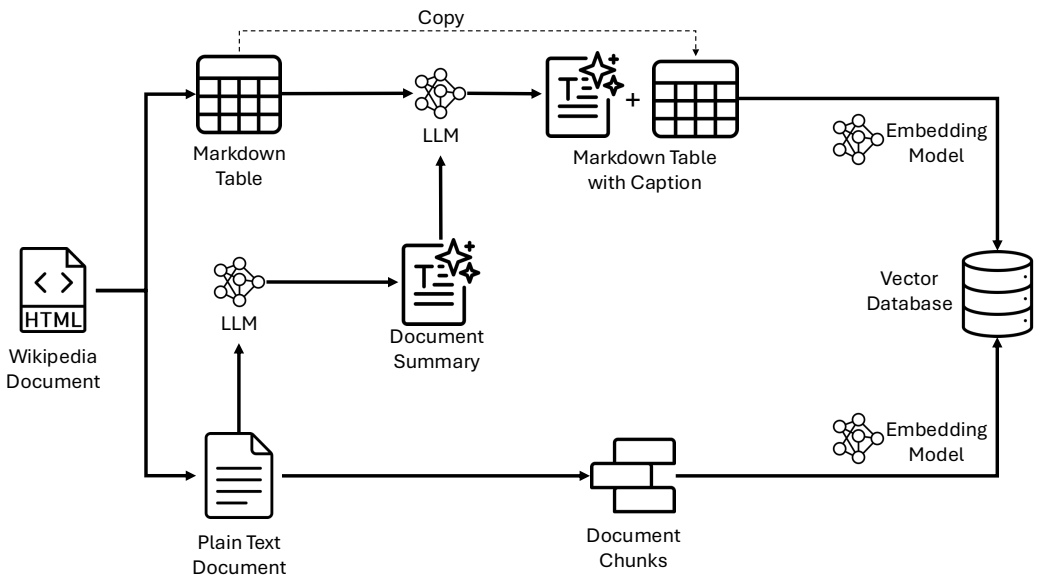

Figure 1: An overview structure of the dataloader

its metadata. The embedding model will be used to convert the chunks into vectors, which can be used for retrieval.

The dataloader will also handle the tabular data in the documents. The tables will be extracted from the documents and processed with the tabular dataloader pipeline before being stored in the vector database. The detailed description of the tabular dataloader pipeline is provided in Section 4.2.

4.1.2 Retriever

The retriever is used to retrieve relevant documents based on the user's query. Once the query from the user is received, the retriever will search through the documents and return the most relevant top-k chunks. Multiple retrieval methods will be implemented and compared, including BM25, vector similarity, and a fusion of these methods.

The retriever can be formally defined as a function which can retrieve the top-k chunks based on the user's query. Let Q be the user's query, the retriever can be defined as follows:

$$R(Q) = \{c_1, c_2, ..., c_k \mid c_i \in L(D)\}$$

where c_i is the top-k chunks retrieved by the retriever based on the user's query Q.

Sparse Retrieval Sparse retrieval methods are based on the bag-of-words model and term frequency-inverse document frequency (TF-IDF) weighting. This method more focuses on the lexical matching between the query and the document.

The term frequency (TF) is the number of times a term appears in a document, in our case, a chunk. It measures the relevance of the term to the document; if a term appears more frequently in a document, it is more likely to be relevant to the document.

The inverse document frequency (IDF) is the logarithm of the total number of documents divided by the number of documents containing the term. It measures how rare a term is across all documents; if a term appears in fewer documents, it is more likely to be relevant to the query.

These measurements are formally defined as follows:

$$\mathrm{TF}(t, d) = \frac{f_{t,d}}{\sum_{t' \in d} f_{t',d}}$$

where $f_{t,d}$ is the frequency of term t in document d.

$$\mathrm{IDF}(t, D) = \log \frac{|D|}{|\{d \in D : t \in d\}|}$$

where $|D|$ is the total number of documents in the corpus, and $|\{d \in D : t \in d\}|$ is the number of documents containing term t.

Therefore, we can have the TF-IDF score as follows:

$$\mathrm{TF-IDF}(t, d, D) = \mathrm{TF}(t, d) \times \mathrm{IDF}(t, D)$$

In this work, we will use the BM25 method, which is an extension of the TF-IDF method. Given a query Q and a document D, the BM25 score is defined as follows:

$$\mathrm{BM25}(Q, D) = \sum_{t \in Q} \mathrm{IDF}(t, D) \times \frac{f_{t,D} \times (k_1 + 1)}{f_{t,D} + k_1 \times (1 - b + b \times \frac{|D|}{avgdl})}$$

where $f_{t,D}$ is the frequency of term t in document D, k_1 and b are hyperparameters, and $avgdl$ is the average document length in the corpus.

BM25 is a popular sparse retrieval method that is based on the TF-IDF method. This algorithm calculates the relevance score between the query and the document based on the query terms appearing in the document. It can return the top k documents that are most relevant to the query based on lexical matching.

Dense Retrieval Dense retrieval methods are based on the dense vector space. This method focuses on the semantic matching between the query and the document by comparing the dense vector representations of the query and the document.

The distributional hypothesis [39] in the information retrieval domain states that the words that appear in similar contexts are likely to have similar meanings. Therefore, the dense vector representations of the words can capture the semantic meaning of the words.

With the development of language models, deep learning approaches [40] [41] [42] [43] have become popular in the information retrieval domain. By mapping text into low-dimensional latent space, where the words with semantic similarity will be close, we can use the dense vector representations of the text to calculate the similarity between the query and the document.

In this work, we will use the embedding model to create the dense vector representations of the chunks and the query, and then use the vector similarity to retrieve the relevant chunks based on the user's query.

Given a query Q and a document D, the cosine similarity score is calculated as follows:

$$\text{CosineSimilarity}(Q, D) = \frac{v_Q \cdot v_D}{||v_Q|| \times ||v_D||}$$

where v_Q and v_D are the dense vector representations of the query Q and the document D respectively.

By calculating the cosine similarity across all the chunks, we can retrieve the top k chunks that are most relevant to the query based on the semantic matching.

4.1.3 Embedding Model

In this work, we use the embedding model to create the dense vector representations of the chunks and the query. The embedding model is a large language model that can generate the dense vector representations of the text. The embedding model is pre-trained on a large corpus of text data and can generate the embeddings for the text data.

$$E(c_i) = v_i$$

where E is the embedding model, c_i is the chunk of the document, and v_i is the dense vector representation of the chunk c_i.

In this work, we compare the performance on multiple embedding models, including the BGE-m3, MiniLM and Mixedbread. By calculating the cosine similarity between the query and the document, we can retrieve the top k chunks that are most relevant to the query based on the semantic matching. Then, we

compare the performance of the embedding models based on the retrieval metrics in terms of nDCG@k, MAP@k, Recall@k and Precision@k.

BGE-m3 BGE-m3 is a large language model developed by Beijing Academy of Artificial Intelligence. This model is designed for information retrieval tasks, ranging from dense retrieval, multi-vector retrieval, and sparse retrieval with multilingual support for more than 100 languages. It can also be used for not only short sentences but also long documents of up to 8192 tokens. [44]

MiniLM MiniLM is a small and efficient sentence transformer model that can be used for generating embeddings for sentence-level or short paragraph text. [45] The model is based on the BERT architecture and is pre-trained on a large corpus of text data from multiple datasets.

Mixedbread Mixedbread AI is a company that focuses on text mining and information retrieval technologies. They have developed a collection of models and tools for various tasks, including embedding generation, text classification, and information extraction. The Mixedbread embedding model series is a collection of models that are designed to generate high-quality embeddings for text data. More specifically, in this project we use the Mixedbread AI small embedding model mixedbread-ai/mxbai-embed-xsmall-v1 [1] to generate the embeddings for the text data. The model is based on the BERT architecture, upon sentence-transformers/all-MiniLM-L6-v2, the model is pre-trained with AnglE loss [46] and Espresso [47] to generate high-quality embeddings to be used for information retrieval tasks.

The reason why we use this model is that the expected scenario for our application is to run on a local machine, where the resources are limited. The model is designed to be lightweight and efficient. With only 22.7 million parameters, the model is small enough to run on a regular laptop while still providing high-quality embeddings.

Fusion Retrieval With the help of both sparse and dense retrieval methods, we can perform both lexical and semantic matching between the query and the document. The sparse retrieval method can capture the lexical matching between the query and the document, while the dense retrieval method can capture the semantic matching between the query and the document.

By fusing these two methods, we can retrieve the top k chunks that are most relevant to the query based on both lexical and semantic matching.

[1] `https://www.mixedbread.ai/blog/mxbai-embed-xsmall-v1`

After retrieving the top k chunks from both retrievers, we can combine the results by ranking the chunks based on the relevance score.

The reciprocal rank fusion method [48] can be used to achieve this goal by first deduplicating the chunks and then ranking the chunks based on the relevance score. The reciprocal rank fusion score is defined as follows:

$$\text{RRFScore}(d \in D) = \sum_{i=1}^{k} \frac{1}{\text{rank}(d) + j}$$

where $\text{rank}(d)$ is the rank of the document d in the list of retrieved documents D, and j is a hyperparameter fixed to 60 to control the impact of outlier rankings.

4.1.4 Reranker

The reranker is used to rerank the top-k chunks retrieved by the retriever. It takes both the query and the retrieved chunks as input and generates a relevance score for each chunk. The reranker can be based on a large language model or a transformer-based model that can generate the relevance score based on the query and the retrieved chunks.

With a reranker model, we can first retrieve a large number of chunks from the retriever with low cost in terms of time and resources. Then, we can use the reranker model to rerank the top-k chunks based on the relevance score to help provide the generator with more relevant chunks.

Mixedbread Reranker The Mixedbread Reranker family is a collection of open-source models that are designed to generate relevance scores for query-document pairs, similar to the Mixedbread embedding model, they are all released by Mixedbread AI. The reranking models are based on the transformer-based cross-encoder architecture and are pre-trained on a large corpus of text data with guided reinforcement prompt optimization, contrastive learning and preference learning [2]. More specifically, in the project we use the smallest model in the family, which is the mixedbread-ai/mxbai-rerank-xsmall-v1. With only 70.8 million parameters, the model is small enough to integrate into the pipeline and run on a laptop without a large increase in the computation cost.

Jinaai Reranker Developed by Jina AI, the Jinaai Reranker family is a collection of language models based on their own JinaBERT architecture that supports the symmetric bidirectional variant of ALiBi [49, 50]. This architecture allows the models to process significantly longer sequences of text, up to an impressive 8,192 tokens. In this project, the model from the family that we use is the

[2] `https://www.mixedbread.ai/blog/mxbai-rerank-v1`

jinaai/jina-reranker-v1-turbo-en, which is also a small model with only 37.8 million parameters.

DistilRoBERTa Reranker The DistilRoBERTa Reranker is a transformer-based model that is distilled from the RoBERTa-base model. The model is pre-trained on the OpenWebTextCorpus dataset, which is a reproduction of OpenAI's WebText dataset. Compared to the original RoBERTa model, the DistilRoBERTa model is smaller and faster, with only 82 million parameters, while the original RoBERTa model has 125 million parameters [51]. We use the DistilRoBERTa model as a baseline for the reranker in this project, so that we can compare the performance of the reranker with the other models and without the reranker in the pipeline.

4.1.5 Generator

The generator is used to generate answers based on the retrieved documents. Once the retriever returns the top-k chunks, the generator will use a large language model to generate answers by providing the chunks and the query to the model within the prompt. The generated answers will be ranked based on the model's confidence score and returned to the user.

This can be formally defined as a function which can generate answers based on the retrieved chunks and the user's query. Let G be the generator, the generator can be defined as follows:

$$G(Q, R(Q)) = A$$

where A is the answer generated by the generator based on the user's query Q and the retrieved chunks $R(Q)$ as context.

We demonstrate the high-level structure of the generator in Figure 2. The generator will take the user's query and the retrieved chunks as input, and then generate the answer based on the context provided by the retrieved chunks. The generator can be based on a large language model such as LLaMA, which can generate answers based on the context provided by the retrieved chunks. The generator also contains a special tabular data retrieval pipeline that is specifically designed for retrieving relevant tables based on the user's query. The details of the tabular data retrieval pipeline are provided in Section 4.2.

LLaMA Models LLaMA (Large Language Model Meta AI) is a series of large language models developed by Meta AI [52]. The first generation of LLaMA models, LLaMA 1, was released in February 2023, which aimed to provide a more efficient and competitive alternative to existing language models with extremely

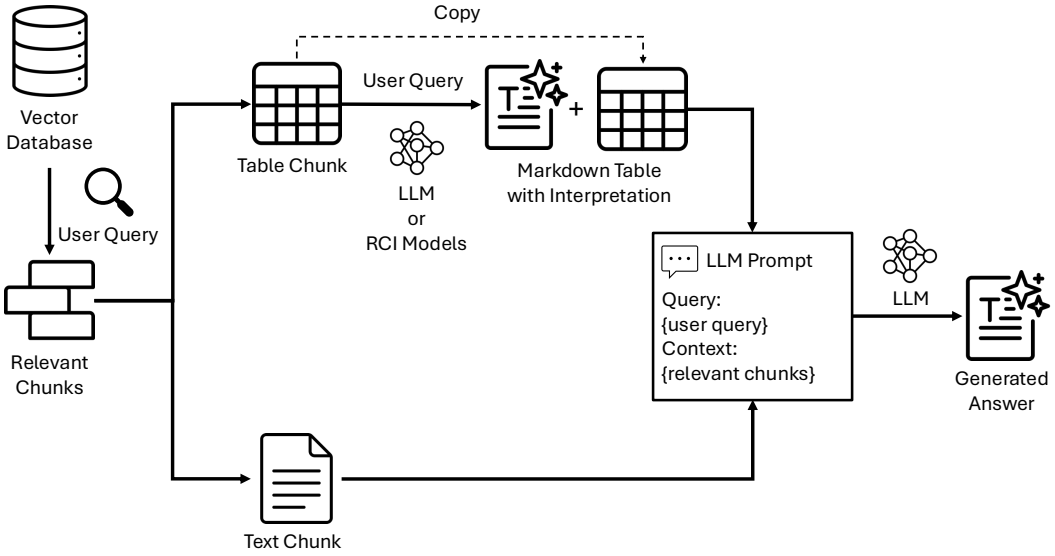

Figure 2: An overview structure of the generator

large parameter sizes. Later, LLaMA 2 was released in the same year, which is a more advanced version of the original LLaMA model in terms of performance, safety, and alignment [53]. In 2024, LLaMA 3 was released, which further improved the capabilities of the model in terms of natural language understanding, reasoning, and tool use [54]. These models are commonly used in a variety of applications, including Retrieval-Augmented Generation (RAG), where their strong performance in understanding context and generating coherent, relevant responses has made them a popular choice.

In this project, we use the LLaMA 3.2, the 3 billion parameter version of the model, which is a smaller version in the LLaMA 3 family. The model is distilled from the LLaMA 3.1 model with 70 billion parameters. With the knowledge distillation method, the model is trained to mimic the behavior of the larger model while being significantly smaller and more efficient. The LLaMA 3.2 model is designed to be lightweight and efficient, making it suitable for deployment in resource-constrained environments.

Additionally, for the RAGAS evaluation, we also use the new LLaMA 4 Scout 17B 16E model [3], which is a MoE (Mixture of Experts) model with a 17 billion parameter model with 16 experts. The model is designed to be more efficient and effective in terms of performance, safety, and alignment. The model is trained on a large corpus of text data and can generate high-quality answers based on the

[3] `https://ai.meta.com/blog/llama-4-multimodal-intelligence/`

context provided by the retrieved chunks.

4.2 Tabular Data Pipeline

Apart from the text retrieval, we also implemented a tabular data pipeline that is specifically designed for retrieving relevant tables based on the user's query.

4.2.1 Table Dataloader

From the documents provided by the user, when the pipeline detects a table, the table will be extracted and handled separately. Following in Table 5 is an example of a table extracted from the documents. The mock table is about the specifications of Apple's M-series chips, which includes the RAM type, width, data rate and the number of Thunderbolt controllers. The table is in markdown format, which can be easily parsed and processed by the pipeline. In the following sections, we will describe how the table data is processed and used in the retrieval and generation steps.

Table 5: Example Tabular Data: Specifications of M1 series chips.

Model	**RAM (-MT/s)**	**Width**	**Data rate**	**TB Controller**
M1	LPDDR4X-4266	128 bit	68.3 GB/s	2×TB3
M1 Pro	LPDDR5-6400	256 bit	204.8 GB/s	2×TB4
M1 Max	LPDDR5-6400	512 bit	409.6 GB/s	4×TB4
M1 Ultra	LPDDR5-6400	1024 bit	819.2 GB/s	8×TB4

A summary of the whole document will be generated by a large language model to describe the content of the document. The summary will be used in the following steps to generate a caption for the table. The summary can provide a high-level overview of the document, which can help the large language model to understand the context of the table better. The prompt of the document summary generation is shown in Appendix 2.

Then, a caption will be generated by a large language model to describe the table content. The column names, row names and along with the table data will be sent to the model to generate a caption. The caption will be concatenated to the table text, which will be sent to the embedding model to generate the embedding for the table. The caption can provide a more detailed description of the table content in terms of semantics and lexical matching, which can be useful for the retrieval of the table based on the user's query. The prompt of the table caption generation is shown in Appendix 3.

Instead of generating the embedding for the table data itself, we generate the embedding for the table caption and the table data. The idea is to use the table caption populate the table content, making the table chunk more informative. This process can help the retriever to retrieve the relevant table chunk more accurately and easily.

Following in Table 6 is an example of how the table caption is generated by the large language model:

Table 6: Example of the table caption generation.

Field	**Value**
Document Text	Apple silicon is a series of system on a chip (SoC) and system in a package (SiP) ... (Wikipedia Text)
Document Summary	The A series is a family of SoCs used in the iPhone, iPad, and Apple TV. The M series is a family of SoCs used in Mac computers... (Summary for the whole document)
Table Data	Table 5
Table Caption	The table lists the specifications of Apple's M-series SoCs, including: - M1: 128-bit, 4266 MT/s RAM, 68.3 GB/s data rate, and 2×TB3 controller... (Caption for the whole table)

To demonstrate the process more clearly, we also show the process visually in Figure 3. The table caption generation process is shown in the figure, where the document text, document summary, table data and table caption are used to generate the table caption. The table caption will be used to retrieve the relevant tables based on the user's query.

4.2.2 Table Retrieval

The table retrieval is similar to the text retrieval, but instead of retrieving the chunks, we retrieve the tables based on the user's query. The table retrieval is based on the table caption, which is generated by the LLM. The table caption is used to retrieve the relevant tables based on the user's query.

4.2.3 LLM-based Table Interpreter

The LLM can be used to interpret the table data and generate a draft answer based on the table data; in the meantime, the LLM can also be used to filter

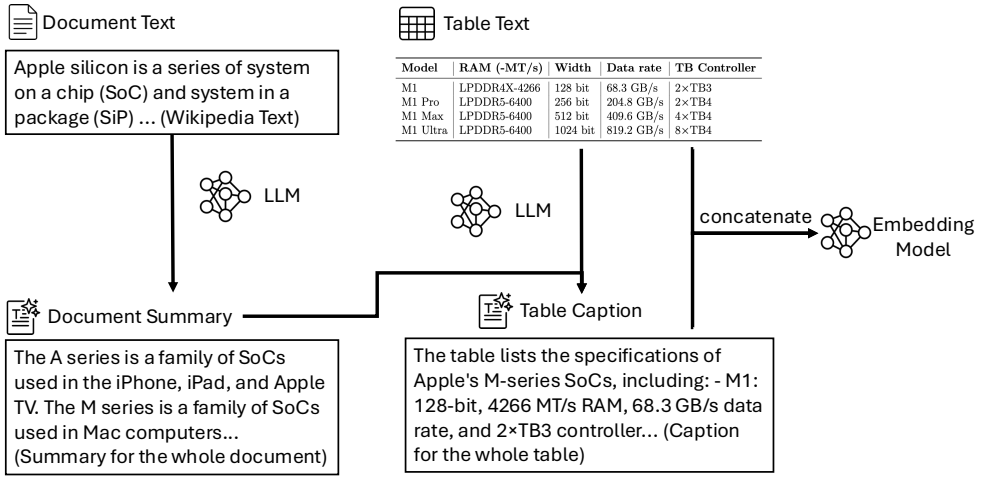

Figure 3: Visualization of the table caption generation process.

out the irrelevant tables based on the user's query. Therefore, to (1) enrich the relevant information for the generator, and (2) filter out the irrelevant tables to reduce the noise, we use the large language model to interpret the table data and generate a draft answer based on the table data.

Table 7: Example of the LLM-based table interpreter output.

Field	**Value**
Question	Which chip has TB3 controllers?
Table Data	Table 5
Prompt	Appendix 4
LLM Output	'M1'

In the following procedure, the draft answer generated by the LLM will be concatenated with the retrieved table chunks. In the generator, the draft answer will be used as part of the context to help the generator to highlight the relevant information from the table data.

In case of an irrelevant table, the LLM will return an **UNKNOWN** answer, which will be used to filter out the irrelevant tables. The table text will be completely set to empty, and the table will not be used in the generator, so that the noise from the irrelevant tables can be reduced.

4.2.4 RCI Model Table Interpreter

If a table is retrieved from the database, the table data will be sent to a row-column intersection (RCI) model to extract the possible rows, columns or cells that are relevant to the user's query [55] . There are two types of RCI models, one to classify question-column pairs and one to classify question-row pairs [4]. The RCI model will generate the probabilities per-cell based on the user's query and the table data. The probabilities will be used to help the large language model to understand the table in the context better and generate the answer based on the table data.

With the tool calling feature of the language model, we can set up a special tool to be used to interpret the table data.

The RCI models are based on the albert-base-v2 [5], which is a transformer-based model that can extract the rows, columns or cells from the table data based on the user's query . The RCI models are trained on WikiSQL, TabMCQ and WikiTableQuestions datasets. There are two variants of the RCI models, one to classify question-column pairs and one to classify question-row pairs.

The retrieved tables, along with the user's query, will be sent to the RCI model to generate the probabilities per-cell. The probabilities are used to help the LLM to understand the table in the context better and generate the answer based on the table data.

The RCI models take the table header and the table data as input, along with the user's query. The data will be applied to the row model and the column model separately. And then, each model will generate the probabilities per-cell based on the user's query and the table data in terms of the row and column. The final probabilities are the sum of the probabilities from both models.

The RCI model can be defined as follows:

$$P_{\text{row}} = \text{RCI}_{\text{row}}(Q, T_{header}, T_{data})$$
$$P_{\text{col}} = \text{RCI}_{\text{col}}(Q, T_{header}, T_{data})$$

$$P_{\text{final}} = P_{\text{row}} + P_{\text{col}}$$

where P_{row} is the probabilities per-cell generated by the row model, P_{col} is the probabilities per-cell generated by the column model, and P_{final} is the final probabilities per-cell.

Table 8 is an example of how the RCI model can be used to extract the relevant rows, columns or cells from the table data based on the user's query.

[4] https://github.com/IBM/row-column-intersection

[5] https://huggingface.co/albert/albert-base-v2

Table 8: Example of the RCI model output based on the user's query and the table data.

Field	**Value**
Question	Which chip has TB3 controllers?
Table Data	Table 5
RCI Model Output	'row_ndx': 0, 'col_ndx': 0, 'confidence_score': -4.436323642730713, 'text': 'M1'

However, the RCI model is not perfect. It requires the arguments to be in a specific format, and it may not be able to handle all the cases. In our implementation, to correctly format the table data, we use LLM to extract the table header and the table columns from the text data, and then format the data into json format. This process needs an extra step to call the LLM, and the practical performance of the RCI model is not as good as expected. Therefore, we will not use the RCI model in the final implementation, but we will keep it as an option for future work. The actual implementation for the table interpreter is replaced by the LLM-based table interpreter, which is more versatile and can handle more cases.

5 Experiments and Results

In this section, we describe the experiments that we conducted to evaluate the performance of our RAG pipeline. We first elaborate the evaluation metrics that we used in the experiments. Then we describe the experimental setup and the results of the experiments.

5.1 Evaluation Metrics

In this work, we use the following evaluation metrics to evaluate the performance of the RAG pipeline.

The metrics are divided into two categories: retrieval metrics and generation metrics. The retrieval metrics are used to evaluate the performance of the retriever, while the generation metrics are used to evaluate the performance of the generator. The retrieval metrics are used to measure how well the retriever can retrieve relevant documents based on the user's query, while the generation metrics are used to measure how well the generator can generate answers based on

the retrieved documents.

These metrics for the generation stage are proposed by RAGAS and are widely used in the retrieval-augmented generation tasks.

5.1.1 Retrieval Metrics

nDCG@k Discounted Cumulative Gain (DCG) is a ranking quality metric that measures the effectiveness of a retrieval system. This metric is often normalized by the ideal DCG (IDCG) to obtain the normalized DCG (nDCG) to make it comparable between different queries [56] . With a graded relevance scale of documents in the top-k retrieved documents.

DCG is the sum of the usefulness of the results at each rank position, discounted by the rank position. The DCG is calculated as follows:

$$\text{DCG@}k = \sum_{i=1}^{k} \frac{rel_i}{\log_2(i+1)} = rel_1 + \sum_{i=2}^{k} \frac{rel_i}{\log_2(i+1)}$$

where rel_i is the relevance score of the document at rank i. The relevance score is a number between 0 and 1, where 0 means not relevant and 1 means relevant.

The ideal DCG (IDCG) is the maximum possible DCG for a given query. The IDCG is calculated as follows:

$$\text{IDCG@}k = \sum_{i=1}^{|REL_k|} \frac{2^{rel_i} - 1}{\log_2(i+1)}$$

where $|REL_p|$ is the number of relevant documents in the top-k retrieved documents. The IDCG is calculated by sorting the relevance scores in descending order and then calculating the DCG for the sorted relevance scores.

And the nDCG is calculated as follows:

$$\text{nDCG@}k = \frac{\text{DCG@}k}{\text{IDCG@}k}$$

MAP@k Mean Average Precision (MAP) is a metric that measures the average precision of the retrieved documents. Or we can say that it measures how many relevant documents are retrieved by the retriever.

The MAP is calculated as follows:

$$\text{MAP@}k = \frac{1}{|Q|} \sum_{q=1}^{|Q|} \text{AveragePrecision@}k(q)$$

where $|Q|$ is the number of queries, and AveragePrecision@$k(q)$ is the average precision for query q. The average precision is calculated as follows:

$$\text{AveragePrecision@}k(q) = \frac{1}{|R|} \sum_{i=1}^{k} \text{Precision@}i(q)$$

where $|R|$ is the number of relevant documents for query q, and Precision@$i(q)$ is the precision at rank i for query q. The precision at rank i is calculated as follows:

$$\text{Precision@}i(q) = \frac{1}{i} \sum_{j=1}^{i} rel_j$$

where rel_j is the relevance score of the document at rank j. The relevance score is a number between 0 and 1, where 0 means not relevant and 1 means relevant.

Recall@k Recall@k is a metric that measures the number of relevant documents retrieved by the retriever. The recall is calculated as the number of relevant documents retrieved by the retriever divided by the total number of relevant documents in the corpus. The recall@k is calculated as follows:

$$\text{Recall@}k = \frac{1}{|Q|} \sum_{q=1}^{|Q|} \frac{\text{TP@}k(q)}{|R|}$$

where $|Q|$ is the number of queries, $TP@k(q)$ is the number of true positives for query q, and $|R|$ is the number of relevant documents for query q. The true positives are the relevant documents retrieved by the retriever.

This metric is a key metric for evaluating the performance of the retriever. A high recall indicates that the retriever can retrieve most of the relevant documents based on the user's query within the top-k retrieved documents.

Precision@k Precision@k is a metric that measures the number of relevant documents retrieved by the retriever divided by the total number of documents retrieved by the retriever. The precision@k is calculated as follows:

$$\text{Precision@}k = \frac{1}{|Q|} \sum_{q=1}^{|Q|} \frac{\text{TP@}k(q)}{\text{TP@}k(q) + \text{FP@}k(q)}$$

where $|Q|$ is the number of queries, TP@$k(q)$ is the number of true positives for query q, and FP@$k(q)$ is the number of false positives for query q. The true positives are the relevant documents retrieved by the retriever, and the false positives are the irrelevant documents retrieved by the retriever.

5.1.2 RAGAS Metrics

In this section, we will describe the LLM-based metrics that are used to evaluate the performance of the RAG pipeline. These metrics are proposed by RAGAS [57] and are widely used in the retrieval-augmented generation tasks. The metrics are designed to evaluate the quality of the generated answers based on the retrieved documents and the user's query. In our RAG application, we do not have the ground truth relevant chunks for the user's query, to mitigate the lack of ground truth answers and relevant chunks, an LLM-based metric is used to evaluate the quality of the retrieved chunks and generated answers.

Contextual Precision@k This is a metric that measures the precision of the retrieved chunks based on the user's query. The precision is calculated as the number of relevant chunks retrieved by the retriever divided by the total number of chunks retrieved by the retriever. The contextual precision@k is calculated as follows:

$$\text{ContextualPrecision@}k = \frac{\sum_{k=1}^{K}(\text{Precision@}k \cdot v_k)}{\text{Number of relevant chunks in top k}}$$

where K is the number of chunks retrieved by the retriever, $Precision@k$ is the precision of the top k chunks, and $v_k \in 0, 1$ is the indicator at rank k.

The indicator v_k is 1 if the chunk at rank k is relevant to the user's query, and 0 otherwise.

The metric is an LLM-based metric with reference. During the evaluation, user's query, the retrieved chunks and the answer from the dataset are used as input to the LLM. This method uses the LLM to judge if one retrieved chunk is relevant to the reference answer. The judgement from the LLM is v_k in the formula.

Contextual Recall@k This is a metric that measures how many relevant chunks are retrieved by the retriever based on the user's query. It tries to judge if the retriever can find all important information from the database. The context recall is computed with user's query, reference answer, and the retrieved chunks.

The reference answer will be broken down into multiple reference contexts, or claims. Each claim is a sentence or a phrase that contains a piece of important information from the reference answer. And then the claim will be analyzed by the LLM to determine if it can be attributed to the retrieved context or not. In an ideal situation, all claims can be attributed to the retrieved context.

The context recall@k is calculated as follows:

$$\text{ContextualRecall@k} = \frac{\text{Number of claims supported by retrieved context}}{\text{Number of claims}}$$

The process of calculating the number of claims is also done by the LLM. The LLM will extract the claims from the reference answer. Then, the LLM will be prompted to judge if the claims can be supported by the retrieved context. If a claim can be supported by the retrieved context, it is considered as a relevant context.

Answer Relevance This metric is to measure how relevant the generated answer is to the user's query. A higher relevance score indicates that the generated answer has a better alignment with the user's query.

The answer relevance is calculated by comparing the similarity of the dense representations between user's query and multiple generated questions from the LLM based on the LLM generated answer. If an answer has a high relevance score, it means the answer can be considered as a good answer. It evaluates how well the answer matches the user's query, and it penalizes the answer that is not relevant or contains too much irrelevant information.

The answer relevance is calculated as follows:

$$\text{AnswerRelevance} = \frac{1}{N}\sum_{i=1}^{N}\text{CosineSimilarity}(E_Q, E_{Q_i})$$

where N is the number of generated questions, E_Q is the dense representation of the user's query, and E_{Q_i} is the dense representation of the generated question i.

Faithfulness This metric is to measure how factually consistent the generated answer is with the retrieved context. A higher faithfulness score indicates that the generated answer is more factually consistent with the retrieved context.

First, all claims from the generated answer will be extracted by an LLM model. Then, the claims will be judged by the LLM model to determine if they can be supported by the retrieved context. If a claim can be supported by the retrieved context, it is considered as a faithful claim.

The faithfulness is calculated as follows:

$$\text{Faithfulness} = \frac{\text{Number of claims supported by retrieved context}}{\text{Number of claims}}$$

5.1.3 Non LLM-based Metrics

In addition to the LLM-based metrics, we also use some non LLM-based metrics to evaluate the performance of the RAG pipeline. In our custom Wikipedia-derived dataset, we have a reference answer for each query, generated by an LLM and verified by a human annotator. These non LLM-based metrics are computed based on the generated answer and the reference answer to give a more objective evaluation of the generated answer.

BERTScore BERTScore is a metric that measures the similarity between the generated answer and the reference answer. [25] It uses the BERT model to compute the similarity between the two answers. The BERTScore is calculated as follows:

$$\text{BERTScore}(E_{A_i}, E_{R_j}) = \frac{1}{N}\sum_{i=1}^{N} \max_j \text{CosineSimilarity}(E_{A_i}, E_{R_j})$$

where N is the number of generated answers, E_{A_i} is the dense representation of the generated answer i, and E_{R_j} is the dense representation of the reference answer j. The BERTScore is a measure of how similar the generated answer is to the reference answer.

Even though a language model is used to compute the BERTScore, it is not like the RAGAS metrics that require a conversation with the LLM to generate the verdicts. Instead, it is a one-shot evaluation metric that uses the BERT model to compute the similarity between the generated answer and the reference answer.

ROUGE ROUGE [24] is a widely used metric in the natural language processing domain to evaluate the quality of the generated text, its original goal is to calculate the overlap between machine-generated summaries and human-written reference summaries. In this work, we use the ROUGE metric to measure the overlap between the RAG generated answer and the human-reviewed reference answer. The formula for the ROUGE metric is as follows:

$$\text{ROUGE-N} = \frac{\sum_{S \in \text{Reference}} \sum_{gram_n \in S} \text{Count}_{\text{match}}(gram_n)}{\sum_{S \in \text{Reference}} \sum_{gram_n \in S} \text{Count}(gram_n)}$$

where $gram_n$ is the n-gram of the generated answer, $Count_{match}(gram_n)$ is the number of times the n-gram appears in the reference answer, and $Count(gram_n)$ is the number of times the n-gram appears in the generated answer. Specifically, ROUGE-L is used to measure the longest common subsequence between the generated answer and the reference answer, where L is the longest common sequence.

We are going to use the ROUGE-1, ROUGE-2 and ROUGE-L metrics to evaluate the performance of the RAG pipeline.

5.2 Experimental Setup

In this section, we will describe the experimental setup that we used to evaluate the performance of the RAG pipeline. The experiments are conducted on a standard ASML laptop, ThinkPad P15 Gen1, with an Intel Core i7-10850H CPU, 32GB DDR4 RAM, and an NVIDIA Quadro RTX 3000 GPU. The pipeline is implemented in Python 3.12, with PyTorch, LlamaIndex, Ollama and HuggingFace models as the main foundation of the RAG pipeline.

Retrieval Component The retrieval component is responsible for retrieving relevant documents based on the user's query. The retriever is implemented using multiple retrieval methods, including BM25, vector similarity, and a fusion of these methods. The retriever is evaluated based on the retrieval metrics described in Section 5.1.

To evaluate the retriever, we used the SciFact dataset as the benchmark dataset, which contains queries, document corpus and relevance labels, the detailed information of this dataset is described in Section 3.1. Therefore, we can compute the retrieval metrics based on the retrieved documents and the relevance labels, including nDCG@k, MAP@k, Recall@k and Precision@k.

The experiments are conducted with multiple retrieval methods, including BM25, vector similarity, and a fusion of these methods, as described in Section 4.1.2. Then, we also implemented a reranker that is used to re-rank the retrieved documents based on the user's query, as described in Section 4.1.4.

The experiments include the following parts:

- **Embedding Model Comparison**: Ingest the SciFact dataset into the LlamaIndex vector database with different embedding models. This part is to compare the performance of the different embedding models in terms of the ability to retrieve relevant documents based on the user's query and the time cost of the embedding process. The retrieval algorithm used in this stage is cosine similarity to give a straightforward comparison of the embedding models.

- **Retrieval Method Comparison**: Compare the performance of different retrieval methods, including BM25, vector similarity and a fusion of these methods. The fusion method is implemented with the RFF score, which is a combination of the BM25 score and the vector similarity score.

- **Reranker Comparison**: Compare the performance of different rerankers, including the mixedbread reranker, JinaAI reranker and DistilRoBERTa reranker. The reranker is used to re-rank the top-k chunks retrieved by the retriever based on the user's query.

RAG Pipeline Evaluation The RAG pipeline is evaluated based on the end-to-end performance of the RAG pipeline, including the dataloader, retriever and generator. The RAG pipeline is implemented using the `LlamaIndex` framework, which provides a simple and efficient way to build a RAG pipeline. The RAG pipeline is evaluated based on the RAGAS metrics described in Section 5.1.2 and the non LLM-based metrics described in Section 5.1.3.

For the RAG pipeline evaluation, the dataset we are going to use is the custom Wikipedia-derived dataset, which is described in Section 3.2. The dataset contains queries, reference answers and the Wikipedia pages documents that are used to generate the answers. The content of the dataset makes it suitable for the end-to-end evaluation of the RAG pipeline.

In this experiment, we are going to first observe how the chunk size affects the performance of the RAG pipeline. The chunk size is the size of the document chunks that are stored in the vector database. In general, a larger chunk size can provide more context to the generator, but it can also bring more noise to the generator. Therefore, a moderate chunk size is preferred to balance the context and noise.

Once the chunk size is determined, we will use the composition of the pipeline as a baseline. Then, we will add the table enhancements to the RAG pipeline to see how the table enhancements can improve the performance of the RAG pipeline. The table enhancements are described in Section 4.2.

Models In order to demonstrate the models we used in the experiments, we will list the models' names we mentioned in this thesis, along with their identifiers in Table 9.

To make the experiments more reproducible, we set the temperature of the LLMs to 0.1 to make the answers more deterministic. If applicable, we also set the random seed to 42 to ensure the reproducibility of the experiments.

5.3 Results

5.3.1 Retrieval Evaluation

We first evaluated the performance for the retriever component in the RAG pipeline. The retriever is the first stage of the RAG pipeline that is responsible for retrieving relevant documents based on the user's query. The retriever is evaluated based on the retrieval metrics described in Section 5.1.

We implemented multiple retrieval methods, including BM25, vector similarity, and a fusion of these methods. Additionally, we also implemented a Reranker that is used to re-rank the retrieved documents based on the user's query.

Table 9: List of models used in the experiments

Model Name	**Identifier**
Embedding Models	
BGE-m3	BAAI/bge-m3
MiniLM	sentence-transformers/all-MiniLM-L6-v2
Mixedbread	mixedbread-ai/mxbai-rerank-xsmall-v1
Reranking Models	
Mixedbread Reranker	mixedbread-ai/mxbai-rerank-xsmall-v1
JinaAI	jinaai/jina-reranker-v1-turbo-en
DistilRoBERTa	cross-encoder/qnli-distilroberta-base
LLMs	
Generator LLM	(Ollama) llama3.2:3b
RAGAS Evaluator LLM	meta-llama/Llama-4-Scout-17B-16E-Instruct

Dense Retrieval After the documents are chunked, the next step is to generate the embeddings for the chunks. The embedding model is used to generate the dense vector representations of the chunks. We conducted the experiments with multiple embedding models, including BAAI/bge-m3, sentence-transformers/all-MiniLM-L6-v2 and mixedbread-ai/mxbai-embed-xsmall-v1. The results of the experiments are shown in Table 10.

Table 10: Performance of different embedding models on the SciFact dataset

Model	**nDCG@5**	**MAP@5**	**Recall@5**	**Precision@5**	**Time**
BGE-m3	0.615	0.577	0.708	0.155	751
MiniLM	0.633	0.593	0.742	0.163	**33**
Mixedbread	**0.638**	**0.598**	**0.743**	**0.164**	44

We can see that the mixedbread-ai/mxbai-embed-xsmall-v1 model has the best performance on the SciFact dataset, with an nDCG@5 score of 0.638, a MAP@5 score of 0.5984, a Recall@5 score of 0.743, and a Precision@5 score of 0.164. In terms of time, the MiniLM model is the fastest, taking only 33 seconds to finish the embedding process for the whole dataset. However, the mxbai model is only slightly slower than the MiniLM model, taking 44 seconds to finish the embedding process, while reaching the best performance.

Therefore, we choose the mixedbread-ai/mxbai-embed-xsmall-v1 model as the embedding model for the retriever component in the RAG pipeline.

Fusion Retrieval The fusion retriever can be used to combine the results from both sparse and dense retrieval methods. We conducted the experiments to compare the performance between the sparse retrieval method (BM25), the dense retrieval method (Embedding model from Mixedbread AI) and the fusion retrieval method that combines the results from both methods with RFF score.

Table 11: Performance of different retrieval methods on the SciFact dataset

Retriever	**nDCG@5**	**MAP@5**	**Recall@5**	**Precision@5**
BM25	0.659	0.627	0.738	0.159
Dense	0.638	0.598	0.743	0.164
Fusion	**0.696**	**0.657**	**0.794**	**0.174**

After the embedding models comparison, we conducted the retrieval methods comparison in Section 5.3.1 as well. We compared the performance of three retrieval methods, including BM25, dense retrieval with embedding models, and hybrid retrieval with both BM25 and embedding models. The results are shown in Table 11.

Following the previous experiment, we used the mixedbread-ai/mxbai-embed-xsmall-v1 model as the embedding model for the dense retrieval method. The BM25 method is a traditional yet effective lexical search method that is widely used in information retrieval tasks. The fusion retrieval method combines the results from both BM25 and dense retrieval methods, which can benefit from the strengths of both methods.

We can see that the fusion retrieval method achieved the best performance on the retrieval task, with an nDCG@5 of 0.696, MAP@5 of 0.657, Recall@5 of 0.794 and Precision@5 of 0.174. The performance is better than both the BM25 and dense retrieval methods, which indicates that the fusion retrieval method can effectively improve the retrieval performance by combining the strengths of both methods.

Reranker The reranker is used to rerank the top-k chunks retrieved by the retriever. We conducted the experiments to compare the performance between the mixedbread reranker, JinaAI reranker and DistilRoBERTa reranker. The results of the experiments are shown in Table 12.

The results of the rerank models comparison are shown in Table 12. We compared the performance of three rerank models, including the mixedbread-ai/mxbai-rerank-xsmall-v1 model, jinaai/jina-reranker-v1-turbo-en and the cross-encoder/qnli-distilroberta-base model. In general, a rerank model is used to rerank the retrieved chunks based on their relevance to the query.

Table 12: Performance of different rerankers on the SciFact dataset

Reranker	**nDCG@5**	**MAP@5**	**Recall@5**	**Precision@5**
Mixedbread rerank	0.680	0.644	0.768	0.169
JinaAI	**0.722**	**0.690**	**0.795**	**0.177**
DistilRoBERTa	0.445	0.396	0.576	0.125

With the help of the rerank model, we can provide a more useful context for the generator by ranking the higher relevant chunks to the top. The results show that the jinaai/jina-reranker-v1-turbo-en model achieved the best performance on the rerank task, with an nDCG@5 of 0.722, MAP@5 of 0.690, Recall@5 of 0.793 and Precision@5 of 0.177. Compared to the previous fusion retrieval method, the performance of the rerank model is slightly better, which indicates that the rerank model can improve the retrieval performance without consuming too much resources and time.

5.3.2 Original RAG Pipeline

With the retriever component in place, we can now evaluate the performance of the end-to-end RAG pipeline. The end-to-end evaluation is to evaluate the performance of the whole RAG pipeline, including the retriever and the generator. We first evaluate the original RAG pipeline without any table enhancements.

Table 13: Original RAG pipeline performance on the custom Wikipedia-derived dataset, without table enhancements.

Chunk Size	**Precision**	**Recall**	**Relevance**	**Faithfulness**
Without Reranker				
384	0.502	0.605	**0.642**	0.857
512	0.567	0.620	0.611	**0.887**
768	**0.618**	**0.685**	0.628	0.822
1024	0.565	0.630	0.592	0.845
With Reranker				
384	0.630	0.638	0.653	0.846
512	0.656	0.656	**0.667**	**0.854**
768	**0.667**	**0.707**	0.647	0.786
1024	0.636	0.685	0.578	0.824

The results of the end-to-end evaluation are shown in Table 13. This experiment is conducted on the custom Wikipedia-derived dataset across multiple chunk

Table 14: Evaluation scores (BERTScore and ROUGE) of the RAG pipeline across different chunk sizes.

Chunk Size	**BERTScore**	**ROUGE-1**	**ROUGE-2**	**ROUGE-L**
Without Reranker				
384	0.879	0.320	0.184	0.301
512	0.873	0.303	0.174	0.278
768	**0.882**	**0.357**	**0.213**	**0.325**
1024	0.877	0.324	0.197	0.298
With Reranker				
384	**0.885**	**0.366**	**0.218**	**0.340**
512	0.878	0.322	0.196	0.303
768	0.879	0.342	0.206	0.312
1024	0.877	0.329	0.201	0.299

sizes, ranging from 384 to 1024, with and without the reranker. Four metrics are computed to evaluate the quality of the generated answers: Precision, Recall, Relevance and Faithfulness.

In order to give more insights of the performance, we also computed the BERTScore and ROUGE metrics for the generated answers, as shown in Table 14.

We conducted a chunk size analysis to determine the moderate chunk size for the end-to-end RAG pipeline in the document ingestion step. The chunk size is an important hyperparameter, since it can affect the length of the context provided to the generator and the grain of the context.

For the precision and recall, we can see that the best performance for both with or without the reranker is achieved with a chunk size of 768. This could be due to the fact that a larger chunk size can provide more context. The RAGAS metrics are based on the LLM judge, hence, it is possible that a large chunk size can be seen as a more relevant context to the user's query since it contains more information.

The relevance is the most direct metric to evaluate whether the generated answer is sufficiently relevant to the user's query. The best performance is achieved with a chunk size of 512, with reranker, which is 0.667. Compared to the results without the reranker, the performance is better, regardless of the chunk size. It shows that with the help of the reranker, the RAG retriever can provide more useful and relevant chunks to the generator, allowing the generator to give a more relevant answer. Apart from the reranker, we can also notice that a large chunk size can lead to a lower relevance score. Ideally, a large chunk size should provide

more context to the generator. However, our goal is to build a RAG application that runs on a laptop, which limits the LLM's capabilities to understand the context. Therefore, a large chunk size may introduce too much noise to the generator, leading to a lower relevance score.

The results with the reranker are generally better than the results without the reranker. It is worth noting that the BERTScore and ROUGE metrics are less sensitive to the chunk size compared to the RAGAS metrics. This might be due to the fact that retrieved chunks are not directly used to compute the BERTScore and ROUGE metrics, but rather the generated answer is compared with the reference answer. In our Wikipedia-derived dataset, the reference answer is relatively short and might be directly related to the retrieved chunks. We can assume that once the important information is retrieved, the generated answer can be considered as a good answer.

Case Study We conducted a case study to analyze the differences between the LLM-based metrics and the non LLM-based metrics.

Here are two examples of a query-answer pair, generated answer and the metrics for the query. These examples are shown in Table 15 as a showcase of the analysis between RAGAS answer relevance and BERTScore.

To further understand the inconsistency between the RAGAS metrics and the BERTScore, we conducted a case study on two examples from the Wikipedia-derived dataset. The results are shown in Table 15.

Example 1 shows a good performance of the RAG pipeline, where the generated answer is relevant to the user's query. Even though it contains some additional information, it still can be seen as a good answer to the user's query. The answer relevance score is 0.943, which indicates that the generated answer is highly relevant to the user's query. The BERTScore is 0.858, which also indicates that the generated answer is similar to the reference answer.

However, in Example 2, the generated answer only partially answers the user's query, while introducing redundant information that is not relevant to the user's query. The answer relevance score is 0.000, because it was not considered as a relevant answer from the RAGAS metrics. However, the BERTScore is 0.811, this might be due to the fact that the generated answer still contains some keywords, and the topic is still relevant to the user's query, such as 'Core 2 Solo Merom-L', 'cache size', and 'TDP'. The BERTScore is still high, which indicates that the generated answer is still similar to the reference answer. This shows that the BERTScore is not sensitive to the relevance of the generated answer, but rather to the semantic similarity between the generated answer and the reference answer.

Table 15: Example of a query-answer pair, generated answer and the metrics for the query.

Example 1	
Query	What naming scheme did Intel introduce in 2023 for mainstream processors?
Reference	Core 3, Core 5, and Core 7.
Response	In 2023, Intel introduced a new naming scheme for mainstream processors, consisting of Core 3/5/7 branding.
Answer Relevance	0.943
BERTScore	0.858
ROUGE-1	0.320
ROUGE-2	0.087
ROUGE-L	0.320
Example 2	
Query	For Core 2 Solo Merom-L, what is the cache size and TDP?
Reference	1 MB L2 cache, 5.5 W TDP.
Response	The provided context does not mention the cache size for Core 2 Solo Merom-L. However, it does mention that the original U2xxx series Merom-L used a special version of the Merom chip with CPUID number 10661 (model 22, stepping A1) that only had a single core. As for the TDP, the context states that the Core 2 Solo, introduced in September 2007, is the successor to the Core Solo and is available only as an ultra-low-power mobile processor with 5.5 Watt thermal design power.
Answer Relevance	0.000
BERTScore	0.811
ROUGE-1	0.082
ROUGE-2	0.021
ROUGE-L	0.062

5.3.3 Table Pipeline

We also evaluated the performance of the RAG application with the table pipeline. The results of these experiments are shown in Table 16.

Table 16: Performance of the RAG pipeline with table enhancements on the custom Wikipedia-derived dataset.

Pipeline	**Precision**	**Recall**	**Relevance**	**Faithfulness**
Baseline	0.656	0.656	0.667	0.854
Table Enhancements	**0.750**	**0.783**	**0.740**	**0.928**
Ablation Study				
No Table Answer	0.765	0.775	0.762	0.895
No Doc Summary	**0.781**	**0.815**	**0.778**	0.928
Only Table Caption	0.765	0.794	0.762	**0.964**

We can see that the table enhancements can improve the performance compared to the original RAG pipeline. On our custom Wikipedia-derived dataset, the table enhancements can improve the precision, recall, relevance and faithfulness from the RAGAS metrics. This indicates that our table enhancements can help the RAG pipeline to understand the table structure better and generate more accurate answers based on the table data.

Additionally, we also conducted an ablation experiment to analyze the impact of each enhancement in the table pipeline.

There are three main steps in the table pipeline, including (1) generating the document summary, (2) generating the table caption, and (3) generating the draft answer. Our ablation study is designed to analyze the impact of each step in the table pipeline. Following is the detailed description of the ablation study:

- **No Table Answer**: This setting is to remove the table draft answer generation step from the table pipeline. The table will be retrieved and sent to the generator as-is, without generating the draft answer or removing the irrelevant tables.
- **No Doc Summary**: This setting is to remove the document summary generation step from the table pipeline. The table caption will be generated based on the table content only, without using the document summary to generate the table caption.
- **Only Table Caption**: This setting is the combination of the previous two settings, where the table draft answer generation step and the document

summary generation step are both removed. The table caption will be generated based on the table content only. And the table will be retrieved as-is, without generating the draft answer or removing the irrelevant tables.

It is worth noting that the pipeline without using the document summary to generate the table caption can achieve the best performance among all these experiment settings, it even outperforms the result with all enhancements. This indicates that the document summary can bring some noise to the table caption, which can affect the performance of the RAG pipeline.

6 Prototype Development

In this section, we describe the prototype development of the enterprise question answering system. The system is built on top of a large language model, which can generate answers based on the user's query and the provided documents. The system consists of a frontend and a backend, which are responsible for the user interface and the data processing respectively.

6.1 Frontend Development

The frontend is built using React, which is a popular JavaScript/TypeScript library for building user interfaces. The frontend is responsible for the user interface, which allows the user to interact with the system. It has a simple and intuitive design that allows the user to upload their own documents, ask questions, and view the answers generated by the system. It follows the design pattern of popular online large language model services, such as ChatGPT.

The frontend consists of multiple components, including the knowledge base, the conversation panel and the chat window. The knowledge base displays the user's own documents that are uploaded to the system. The conversation panel displays the conversation history between the user and the system. The chat window allows the user to ask questions and view the answers generated by the system.

6.2 Backend Development

The backend is built using FastAPI, which is a modern Python web framework for building APIs. The backend is responsible for the data processing, which includes the dataloader, the retriever, and the generator. It connects the frontend to the large language model service, which is used to generate answers based on the user's query and the provided documents.

The backend consists of multiple components, including the dataloader, the retriever, and the generator. When the users upload their own documents from the frontend via the API, the dataloader processes these documents and performs the necessary transformations to make them suitable for the retrieval and generation tasks. When the users interact with the system via the frontend and ask questions, the retriever in the backend retrieves relevant information from the knowledge base based on the user's query. The generator then uses this information to generate answers to the user's questions. During the generation, the backend also streams the response to the frontend, which allows the user to see the answer being generated in real-time.

7 Discussion

In our work, we developed an RAG application for enterprise environments, which allows users to upload organization-internal documents and ask questions about the content of the content of these documents. Our research questions are focused on the effectiveness of the RAG application that can be used on a laptop to ensure the confidentiality and compliance of the data. And we also explored the table handling capabilities of the RAG application, which is a common requirement in enterprise environments.

In this section, we discuss the results of our experiments for each component of the RAG application, including the retrieval component, the LLM-based and non-LLM-based metrics, the chunk size, the tabular data handling, the RCI models, and the ablation study. We also discuss the limitations and challenges of our RAG application, as well as future work that can be done to improve the RAG application.

Retrieval Component To build up the foundation of our RAG application, we conducted a series of experiments to evaluate the composition of the retrieval component.

With the SciFact dataset, we found that the mixedbread-ai/mxbai-embed-xsmall-v1 as the embedding model and jinaai/jina-reranker-v1-turbo-en as the reranker model can achieve the best performance. With the fusion retrieval method, our retrieval component can perform both semantic and lexical search. Combining the strengths of dense retrieval, sparse retrieval, and reranking, our retrieval component can effectively retrieve relevant documents on SciFact dataset. The best results for our retrieval component were an nDCG@5 of nearly 72%, MAP@5 of about 69%, Recall@5 of close to 80%, while Precision@5 was lower at 18%.

It is worth mentioning that even though the Precision@5 is low, this is due to

the nature of the SciFact dataset. In SciFact, the average number of relevant documents for each query is 1.13, this means in most of good cases, the top 5 retrieved documents only contain one relevant document. Therefore, the Precision@5 looks low, but it is actually a relatively good performance.

One possible alternative metric is to use Precision@1, which is the precision of the top 1 retrieved document. This can solve the problem of low Precision@5, since in most cases, there is only one relevant document in the top 5 retrieved documents. Using Precision@1 will alter the evaluation of the retrieval component into a strict and binary evaluation: if the top 1 retrieved document is relevant or not. This will be useful when the downstream task (for the generator, in our case) only needs one relevant document to generate the answer. However, in our implementation, the retrieval component will pass the top k retrieved documents to the generator, where k is less likely to be 1. The LLM itself is able to extract useful information as long as the information is present in the retrieved documents. Therefore, we consider to use the Precision@5 metric because it is able to evaluate whether the relevant documents can be retrieved in the top k retrieved documents, which is more close to the use-case in the RAG application.

LLM-based Metrics vs. Non-LLM-based Metrics We chose RAGAS metrics as the main evaluation metrics for our RAG application, additionally, we also evaluated the RAG performance on BERT Score and ROUGE metrics. For each experiment setting, we used the RAGAS metrics, BERT Score, and ROUGE metrics to evaluate the performance of the RAG application. We found that the RAGAS metrics can better reflect the performance of our RAG application than the BERT Score and ROUGE metrics in terms of evaluating the quality of the generated answers.

By using LLM-as-a-judge, RAGAS metrics designed a chain-of-thought evaluation process to evaluate the generated answers from multiple perspectives, such as contextual precision, contextual recall, answer relevance, and faithfulness. Among these metrics, we think the answer relevance is the most important metric for our RAG application, since it is the direct measure of how well the generated answer is. The contextual precision and contextual recall are metrics to evaluate how well retrieved documents are used in the generated answer. While the faithfulness is to measure whether the generated answer is faithful to the retrieved documents, this is more relevant to the LLM model itself, to check whether the LLM model is generating hallucinations or not. Therefore, to determine the composition of the RAG application, we made our decision based on the answer relevance metric.

We also used BERT Score and ROUGE metrics to evaluate our pipeline. We found that the BERT Score and ROUGE metrics might not be as good as RAGAS metrics in evaluating the RAG application. We provided examples from our pipeline and their metrics scores. We can see that the BERT Score is still high

even when the generated answer is insufficient to answer the question. Similar to embedding, BERT Score is based on the semantic similarity to the ground truth answer reviewed by human, it does not consider if the answer is useful or not. Considering the nature of the RAG application, the generated answers are always based on the retrieved documents, therefore, the generated answers and the ground truth answers are always discussing the same topic, which leads to high BERT Score.

The ROUGE metric has its own limitations as well, it is based on the n-gram overlap between the generated answer and the ground truth answer. Ideally, the ROUGE metric can reflect the lexical similarity between the generated answer and the ground truth answer, however, in our case, we notice that the ROUGE metric is easily affected by the word choice, phrasing and structure of the generated answer. For example, if the reference answer is 'Core 3/5/7', while the generated answer is 'Intel Core i3, Core i5 and Core i7'. In this case, the generated answer is actually correct, but the ROUGE metric will be low because of the different expressions of the same information.

Therefore, we consider that the BERT Score and ROUGE metrics can only be used as supplementary metrics for our RAG application, but not the main evaluation metrics.

Chunk Size We used our custom Wikipedia-derived dataset to evaluate the chunk size of the end-to-end performance of our RAG application. We found that the chunk size of 512 with reranker model can achieve the best performance among all the chunk sizes we tested in terms of the answer relevance.

There is a trade-off between the chunk size and the performance of the RAG application. A smaller chunk size can provide a short and less noisy context for the LLM model, while it might not be able to capture the full context of the informative content. A larger chunk size can provide a rich context, but it might introduce more noise and irrelevant information to the LLM model.

Our experiments can also reflect this pattern. The contextual precision and contextual recall are higher when the chunk size is 768, when we increase or decrease the chunk size, the contextual precision and contextual recall became lower. This indicates that when we evaluate the context with RAGAS, a longer and richer context is more relevant to the answer.

We also observed that the faithfulness is higher when the chunk size is 512, compared to the chunk size of 384, 768 and 1024, regardless of the reranker model. For the smaller chunk size, such as 384, the faithfulness is lower, this might be due to the fact that the context is too short and the relevant sentences might be truncated. This might cause the LLM to make something to answer the question, leading to lower faithfulness. On the one hand, the fact that 512 chunk size achieves the best faithfulness score shows that increasing the context size can help

the LLM to get more information to generate faithful answers. On the other hand, increasing the chunk size to 1024 does not improve the faithfulness even further, this could be because the long context might introduce more noise and irrelevant information to the LLM. As context grows longer, the LLM's ability to attend accurately to relevant information may decrease, resulting in misinterpretation or hallucination. Therefore, we consider the chunk size of 512 is the best choice for our RAG application, making a good balance between the context richness and the noise.

Tabular Data Handling By adding more information to the chunk and removing irrelevant table chunks from the context, our tabular data handling process can improve the performance of the RAG application. Compared to the baseline, our RAG application with table enhancements achieves a higher answer relevance, contextual precision, contextual recall, and faithfulness.

Before sending the table to the embedding model, we prompt the LLM to generate a caption for the table, this caption brings more information to the chunk, making the context more informative. In the process for generating the caption, we also add the document summary to the prompt, this is to ensure the LLM model can understand the context of the table. It is not only providing a large chance for the retrieval component to retrieve the relevant table chunk, but also providing more information to the LLM model to understand the table data.

After the table chunk is retrieved, we prompt the LLM model to generate a draft answer based on the table chunk and the query, this draft answer is used to highlight the relevant table data. By highlighting the relevant table data, the generator model can focus on the relevant information in the table. Furthermore, if the LLM notices that the table chunk is not relevant to the query, it can simply ignore the table chunk and generate the answer based on the other retrieved documents. Our RAG application is expected to be used in a laptop environment, with limited resources. Therefore, this step is designed to shrink the context size, making the LLM model focus on the relevant information, instead of feeding a long context to the LLM model. This is a more flexible way to handle the table data, since it allows the LLM model to focus on the relevant information and ignore the irrelevant information.

RCI Models Initially, we implemented a table interpreter with an RCI model, which can extract the most important value from the tabular data. However, in the later experiments, we found that to correctly feed the data to the RCI model, we need to parse the tabular data from the table with the LLM, which can be a time-consuming and unstable process since the table can be in various formats. For instance, the table might have merged cells, or to present the table in HTML, the table might have its own style. Even though we can use `pandas` to parse the

table data and convert it into a markdown format, it is not always guaranteed that the table is well-formatted and can be parsed correctly by the RCI model. Therefore, we decided to use an LLM as the replacement to the RCI model, which is used to highlight the important values in the table data. The results show that the idea of highlighting important values in the table data is still valid.

Ablation Study We conducted an ablation study to evaluate how each process contributes to the overall performance of the RAG application with table enhancement.

The results are interesting: all the results in our ablation study are better than not only the baseline, but also the RAG application with all these capabilities enabled.

Comparing the results of the baseline and our Only Table setting, we can observe a substantially higher metrics score. This is an evidence showing that the idea of splitting the table chunk from the other text chunk is valid. The contextual precision and contextual recall are higher, indicating that these chunks are more relevant to the query. The faithfulness is also higher, showing that the context is less noisy and clearer to the LLM. And most importantly, the answer relevance is higher, indicating that the idea can help the LLM to understand the context both on the table chunk and the other text chunk.

The impact of using document summary is not as significant as we expected. In our No Doc Summary setting, the document summary is disabled, while the other capabilities are still enabled as the Table Enhancements setting. On the contrary, the result shows that removing the document summary can improve the overall performance. This could be because when the document summary is used, the generated table caption carries more information about the whole document. This might introduce more semantic and lexical similarity to the table that is not relevant to the query. These irrelevant table chunks are more likely to be retrieved during the retrieval process, which leads to lower performance.

Our draft answer generation process shows a similar pattern. In the No Table Answer setting, the result has a slightly improvement compared to the Table Enhancements setting. This might be because the LLM is sufficient to capture the information in the context, and the draft answer generation process is not necessary to make the context more clear. However, we can notice that the No Doc Summary has a higher overall performance compared to the Only Table setting. The difference between these two settings is that the No Doc Summary setting has the draft answer generation process enabled, while the Only Table setting does not. This might imply that by removing irrelevant table chunks, the contextual precision and contextual recall can be improved, leading to a higher answer relevance. Therefore, the draft answer generation process is still useful to improve the performance of the RAG application by removing irrelevant table

chunks.

According to our ablation study, we can conclude that these table enhancements have the following impacts on the RAG application:

- **Table caption:** This is the most important and useful enhancement, improving the relevance of the retrieved documents.
- **Document summary:** This can provide more context to the LLM when the table caption is generated, but it might introduce more noise to the context, making the retrieval process less effective.
- **Draft answer generation:** This can help the LLM to focus on the relevant information in the table chunk, but it might not be necessary for the LLM. However, removing irrelevant table chunks has the potential to improve the overall performance of the RAG application.

Hence, we can conclude that the best setting for our RAG application is to use the generated table caption without the document summary involved, and follow by the draft answer generation process to remove irrelevant table chunks. This provides a clear and relevant context for the LLM to improve the overall performance of the RAG application.

Limitations and Challenges One limitation comes from the dataset used for our experiments. Our custom Wikipedia-derived data set served as the end-to-end dataset for this project, but is not a standard or widely recognized benchmark. There is no direct comparison with other RAG architectures or applications, which makes it difficult to evaluate the performance of our RAG application in a broader range.

Additionally, our custom dataset is not large enough to cover all possible scenarios in enterprise environments, which may limit the generalizability of our findings. In real scenarios, documents can be in various formats, such as PDF, Word, Excel, and HTML, which makes parsing and processing more challenging. Our experiments assumed that documents are well-formatted, semi-structured HTML files, providing a controlled environment. However, in real-world cases, especially in PDF documents, tables can be more complex and difficult to parse, with merged cells, different styles, or tables split across pages. This presents challenges for the table enhancement pipeline, which may not be able to handle all possible scenarios in real-world documents.

8 Conclusion

In this thesis, we have developed a question-answering RAG application that integrates table enhancements for enterprise use cases. The application is designed to be deployed on a regular laptop, making it accessible for employees to use in their daily work. The application is capable of running on a local machine, without Internet connection; this makes it suitable for enterprise use cases where data privacy is a concern. We also integrated a frontend and a backend to provide a user-friendly and intuitive interface for users to interact with the application.

In our method, we used the SciFact dataset to determine the composition of the retrieval component. We used an embedding model from Mixedbread-AI to generate embeddings for the documents, and used a fusion retrieval method to combine the results from BM25 and the embedding. Then, we also used a reranking model from Jina AI to rerank the results from the retrieval component to improve the quality of the retrieved documents.

We created a custom Wikipedia-derived dataset to evaluate the end-to-end performance of the RAG application. The dataset consists of 5 documents in the technical domain, each document contains text and multiple tables. We formulated 10 questions per document, resulting in a total of 50 question-answer pairs. We proposed a table enhancement method to improve the ability of the RAG application to handle tables. The method splits the table chunk from the other text chunks, and uses an LLM to generate document summaries, then combines the table chunk and the document summary to generate a table caption. The caption is then used to add information to the table chunk.

We evaluated the RAG application with table enhancements on the custom Wikipedia-derived dataset. The results show that the RAG application with table enhancements outperforms the baseline RAG application in terms of precision, recall, relevance, and faithfulness from the RAGAS metrics.

Additionally, we conducted an ablation study to evaluate how each process contributes to the overall performance of the RAG application with table enhancement. The results show that the idea of splitting the table chunk from the other text chunk and generating a caption for the table chunk is valid. The ideas of using document summaries to add information to the table chunk, and the idea of generating draft answers for the table chunk have their own advantages, but they are not as effective as generating a caption for the table chunk. Even though these ideas are not as good as we expected, they still show potential to improve the performance of the RAG application in handling tables.

Based on our experiments and results, we can answer our research questions as follows:

- **RQ-1: What is a good combination of retrieval methods for the RAG pipeline that can run on a regular laptop?** Combining dense

retrieval and sparse retrieval methods, our experiments show that the fusion retrieval with Reciprocal Rank Fusion (RRF) is a good combination of retrieval methods. The fusion retrieval method can benefit from the strengths of both methods, resulting in good performance in retrieving relevant documents. Following the fusion retrieval, we also used a reranking model to improve the quality of the retrieved documents even further. The reranking model can help to rank the most relevant documents higher, which can improve the performance of the RAG application. This implementation can run on a regular laptop, making it suitable for our use case.

- **RQ-2: How can we build a retrieval-augmented generation (RAG) pipeline that can handle tabular data effectively?** We proposed a table enhancement method to improve the ability of the RAG application to handle tables. The method consists of the following steps: (1) Splitting the table chunk from the other text chunk, (2) Use an LLM to generate table captions, optionally with document summaries in this process. The table chunk contains table text and caption will be sent to the embedding model. (3) When a table chunk is retrieved, an LLM will generate a draft answer to make the table chunk more clear to the query, and remove the table chunk if the table data is not relevant to the query. By designing a tabular pipeline and using LLM to populate the table chunk with more context, this method can help the RAG application to handle tabular data more effectively compared to the baseline RAG application. Moreover, the albation study shows that disabling the document summary during the caption generateion step can make the performance even better.

- **RQ-3: What is the performance of the tabular data retrieval pipeline in our RAG application?** The performance of the best RAG application with table enhancements is evaluated on our custom Wikipedia-derived dataset. The results show that with the table enhancements, the end-to-end RAG application achieves 78.1%, 82.5%, 78.8% and 92.8% on contextual precision, contextual recall, answer relevance, and faithfulness respectively. Compared to the baseline RAG application, the best scores of the RAG application with table enhancements are 12.5%, 15.9%, 11.1% and 7.4% higher on these metrics. These improvements indicate that our tabular pipeline can substantially improve the performance of the RAG application on documents with tabular data.

Our future work can focus on addressing the limitations and challenges mentioned above.

One is to evaluate the RAG application on a larger dataset. This might involve collecting more documents to create a more comprehensive dataset that

covers a wider range of scenarios in enterprise environments. This can help to better evaluate the performance of the RAG application and its table handling capabilities.

Another is to improve the table handling capabilities of the RAG application. This can involve developing a more robust table parsing and processing pipeline, so that a document with more noise and complex table structures can still be handled effectively. This might include using more advanced table parsing libraries or methods. One approach could be to use OCR (Optical Character Recognition) techniques to extract table data from documents.

One possible improvement might be re-designing the way of using RCI models. The RCI models we used in this project still have their potential to be used in the RAG application, considering their lightweight and fast inference speed. We can explore how to better parse, format, and store the table data, so that the RCI models are able to interpret the table more stably.

Finally, we can also explore a more fine-grained knowledge management that can better handle the documents. Our current implementation is storing all the ingested chunks together, so the search is based on the whole knowledge base. However, in enterprise environments, the application might need to handle a large number of documents. Therefore, it might be more efficient to implement a more fine-grained knowledge management approach, where documents are categorized and indexed based on their content, metadata, or other relevant attributes. This can be helpful to improve the precision and recall of the retrieval component.

Appendix

Prompts

Listing 1: "Custom Dataset Generation Prompt for ChatGPT"

```
1.
[USER:]
Below is a piece of technical documentation. Your task is to ask a
question about the content. The question should be answerable based on
the information provided in the document. Do not ask question about date
. The question should be clear and concise.

Document:
{document}

2.
[USER:]
Based on the provided document, generate the answer to the question.
```

Listing 2: "Document Summary Prompt"

```
[SYSTEM:]
You will be given a document, your task is to summarize the document in
a concise manner.

[USER:]
Document:
{document}

Please only return the summary of the document.
```

Listing 3: "Table Summary Prompt"

```
[SYSTEM:]
You will be given a table in markdown format along with the summary of
the whole document, your task is to summarize the table in a concise
manner.

[USER:]
Document Summary:
{document_summary}

Table:
{table_text}

Please only return the summary of the table.
```

Listing 4: "LLM Table Interpreter Prompt"

```
[SYSTEM:]
You will be given a table in markdown format, your task is to answer the
 question based on the table. If the answer is not present in the table,
 return *UNKNOWEN*

[USER:]
Question:
{user_query}
Table:
{table_text}
Please only return the answer to the question.
```

References

[1] Lynette Hirschman and Robert Gaizauskas. Natural language question answering: the view from here. *Natural Language Engineering*, 7(4):275–300, 2001.

[2] Dell Zhang and Wee Sun Lee. A web-based question answering system. 2003.

[3] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. Attention is all you need. In *Proceeddings of the 30th Advances in Neural Information Processing Systems: Annual Conference on Neural Information Processing Systems, (NeurIPS)*, pages 5998–6008, 2017.

[4] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. BERT: pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, (NAACL-HLT)*, pages 4171–4186, 2019.

[5] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. Roberta: A robustly optimized BERT pretraining approach. *arXiv preprint arXiv:1907.11692*, 2019.

[6] Luciano Floridi and Massimo Chiriatti. GPT-3: Its nature, scope, limits, and consequences. *Minds and Machines*, 30(4):681–694, 2020.

[7] Bert F Green Jr, Alice K Wolf, Carol Chomsky, and Kenneth Laughery. Baseball: an automatic question-answerer. In *Papers presented at the western joint IRE-AIEE-ACM Computer Conference*, pages 219–224, 1961.

[8] C Cordell Green and Bertram Raphael. The use of theorem-proving techniques in question-answering systems. In *Proceedings of the 23rd ACM National Conference*, pages 169–181, 1968.

[9] William A Woods. Transition network grammars for natural language analysis. *Communications of the ACM*, 13(10):591–606, 1970.

[10] Navid Yaghmazadeh, Yuepeng Wang, Isil Dillig, and Thomas Dillig. Sqlizer: query synthesis from natural language. *Proceedings of the ACM on Programming Languages*, 1(OOPSLA):1–26, 2017.

[11] Fei Li and Hosagrahar V Jagadish. Constructing an interactive natural language interface for relational databases. *Proceedings of the VLDB Endowment*, 8(1):73–84, 2014.

[12] Ana-Maria Popescu, Oren Etzioni, and Henry Kautz. Towards a theory of natural language interfaces to databases. In *Proceedings of the 8th International Conference on Intelligent User Interfaces, (IUI)*, pages 149–157, 2003.

[13] Majid Latifi and Miquel Sànchez-Marrè. The use of NLP interchange format for question answering in organizations. In *Proceedings of the 16th International Conference of the Catalan Association for Artificial Intelligence*, pages 235–244, 2013.

[14] Alejandro Barredo Arrieta, Natalia Díaz-Rodríguez, Javier Del Ser, Adrien Bennetot, Siham Tabik, Alberto Barbado, Salvador García, Sergio Gil-López, Daniel Molina, Richard Benjamins, et al. Explainable artificial intelligence (XAI): concepts, taxonomies, opportunities and challenges toward responsible AI. *Information Fusion*, 58:82–115, 2020.

[15] Patrick Lewis, Ethan Perez, Aleksandra Piktus, Fabio Petroni, Vladimir Karpukhin, Naman Goyal, Heinrich Küttler, Mike Lewis, Wen-tau Yih, Tim Rocktäschel, Sebastian Riedel, and Douwe Kiela. Retrieval-augmented generation for knowledge-intensive NLP tasks. In *Proceedings of the 33rd Advances in Neural Information Processing Systems: Annual Conference on Neural Information Processing Systems, (NeurIPS)*, pages 9459–9474, 2020.

[16] Kelvin Guu, Kenton Lee, Zora Tung, Panupong Pasupat, and Mingwei Chang. Retrieval augmented language model pre-training. In *Proceedings of the 37th International Conference on Machine Learning, (ICML)*, pages 3929–3938. PMLR, 2020.

[17] Alec Radford, Jeffrey Wu, Rewon Child, David Luan, Dario Amodei, Ilya Sutskever, et al. Language models are unsupervised multitask learners. *OpenAI blog*, 1(8):9, 2019.

[18] Stephen Robertson, Hugo Zaragoza, et al. The probabilistic relevance framework: BM25 and beyond. *Foundations and Trends in Information Retrieval*, 3(4):333–389, 2009.

[19] Karen Sparck Jones. A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 28(1):11–21, 1972.

[20] Vladimir Karpukhin, Barlas Oğuz, Sewon Min, Patrick Lewis, Ledell Wu, Sergey Edunov, Danqi Chen, and Wen-tau Yih. Dense passage retrieval for open-domain question answering. *arXiv preprint arXiv:2004.04906*, 2020.

[21] Liang Wang, Nan Yang, Xiaolong Huang, Linjun Yang, Rangan Majumder, and Furu Wei. Multilingual e5 text embeddings: A technical report. *arXiv preprint arXiv:2402.05672*, 2024.

[22] Kishore Papineni, Salim Roukos, Todd Ward, and Wei-Jing Zhu. Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th annual meeting of the Association for Computational Linguistics, (ACL)*, pages 311–318, 2002.

[23] Chin-Yew Lin and Franz Josef Och. ORANGE: A method for evaluating automatic evaluation metrics for machine translation. In *Proceedings of the 20th International Conference on Computational Linguistics*, pages 501–507, 2004.

[24] Chin-Yew Lin. ROUGE: A package for automatic evaluation of summaries. In *Text Summarization Branches Out*, pages 74–81, 2004.

[25] Tianyi Zhang, Varsha Kishore, Felix Wu, Kilian Q Weinberger, and Yoav Artzi. BERTScore: Evaluating text generation with BERT. In *Proceedings of the 8th International Conference on Learning Representations, (ICLR)*, 2020.

[26] Rakesh Pimplikar and Sunita Sarawagi. Answering table queries on the web using column keywords. *arXiv preprint arXiv:1207.0132*, 2012.

[27] Petros Venetis, Alon Halevy, Jayant Madhavan, Marius Pasca, Warren Shen, Fei Wu, Gengxin Miao, and Chung Wu. Recovering semantics of tables on the web. *Proc. VLDB Endow.*, 2011.

[28] Pallavi Pyreddy and W Bruce Croft. TINTIN: A system for retrieval in text tables. In *Proceedings of the 2nd ACM International Conference on Digital Libraries,*, pages 193–200, 1997.

[29] Ying Liu, Kun Bai, Prasenjit Mitra, and C Lee Giles. Tableseer: automatic table metadata extraction and searching in digital libraries. In *Proceedings of the 7th ACM/IEEE Joint Conference on Digital Libraries, (JCDL)*, pages 91–100, 2007.

[30] Kenneth Ward Church. Word2vec. *Natural Language Engineering*, 23(1):155–162, 2017.

[31] Li Zhang, Shuo Zhang, and Krisztian Balog. Table2vec: Neural word and entity embeddings for table population and retrieval. In *Proceedings of the 42nd international ACM SIGIR conference on research and development in information retrieval*, pages 1029–1032, 2019.

[32] Roee Shraga, Haggai Roitman, Guy Feigenblat, and Mustafa Cannim. Web table retrieval using multimodal deep learning. In *Proceedings of the 43rd International ACM SIGIR conference on research and development in Information Retrieval, (SIGIR)*, pages 1399–1408, 2020.

[33] Mohamed Trabelsi, Brian D Davison, and Jeff Heflin. Improved table retrieval using multiple context embeddings for attributes. In *Proceeddings of the IEEE International Conference on Big Data, (IEEE BigData)*, pages 1238–1244, 2019.

[34] Jonathan Herzig, Paweł Krzysztof Nowak, Thomas Müller, Francesco Piccinno, and Julian Martin Eisenschlos. TaPas: Weakly supervised table parsing via pre-training. *arXiv preprint arXiv:2004.02349*, 2020.

[35] Jonathan Herzig, Thomas Müller, Syrine Krichene, and Julian Martin Eisenschlos. Open domain question answering over tables via dense retrieval. *arXiv preprint arXiv:2103.12011*, 2021.

[36] Sohini Roychowdhury, Marko Krema, Anvar Mahammad, Brian Moore, Arijit Mukherjee, and Punit Prakashchandra. ERATTA: Extreme RAG for table to answers with large language models. *arXiv preprint arXiv:2405.03963*, 2024.

[37] Feifei Pan, Mustafa Canim, Michael Glass, Alfio Gliozzo, and Peter Fox. CLTR: an end-to-end, transformer-based system for cell-level table retrieval and table question answering. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing: System Demonstrations*, pages 202–209, 2021.

[38] Ehsan Kamalloo, Nandan Thakur, Carlos Lassance, Xueguang Ma, Jheng-Hong Yang, and Jimmy Lin. Resources for Brewing BEIR: Reproducible reference models and statistical analyses. In *Proceedings of the 47th International ACM SIGIR Conference on Research and Development in Information Retrieval, (SIGIR)*, pages 1431–1440, 2024.

[39] Zellig S Harris. Distributional structure. *Word*, 10(2-3):146–162, 1954.

[40] Po-Sen Huang, Xiaodong He, Jianfeng Gao, Li Deng, Alex Acero, and Larry Heck. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, pages 2333–2338, 2013.

[41] Jiafeng Guo, Yixing Fan, Qingyao Ai, and W Bruce Croft. A deep relevance matching model for ad-hoc retrieval. In *Proceedings of the 25th ACM International Conference on Information and Knowledge Management, (CIKM)*, pages 55–64, 2016.

[42] Bhaskar Mitra and Nick Craswell. Neural models for information retrieval. *arXiv preprint arXiv:1705.01509*, 2017.

[43] Jiafeng Guo, Yixing Fan, Liang Pang, Liu Yang, Qingyao Ai, Hamed Zamani, Chen Wu, W Bruce Croft, and Xueqi Cheng. A deep look into neural ranking models for information retrieval. *Information Processing & Management*, 57(6):102067, 2020.

[44] Jianlyu Chen, Shitao Xiao, Peitian Zhang, Kun Luo, Defu Lian, and Zheng Liu. M3-embedding: Multi-linguality, multi-functionality, multi-granularity text embeddings through self-knowledge distillation. In *Findings of the Association for Computational Linguistics, (ACL)*, pages 2318–2335, 2024.

[45] Nils Reimers and Iryna Gurevych. Sentence-BERT: sentence embeddings using siamese BERT-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing, (EMNLP-IJCNLP)*, pages 3982–3992, 2019.

[46] Xianming Li and Jing Li. AoE: Angle-optimized embeddings for semantic textual similarity. In *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics*, pages 1825–1839, 2024.

[47] Xianming Li, Zongxi Li, Jing Li, Haoran Xie, and Qing Li. 2d matryoshka sentence embeddings. *arXiv preprint arXiv:2402.14776*, 2024.

[48] Gordon V Cormack, Charles LA Clarke, and Stefan Buettcher. Reciprocal rank fusion outperforms condorcet and individual rank learning methods. In *Proceedings of the 32nd international ACM SIGIR conference on Research and development in information retrieval*, pages 758–759, 2009.

[49] Michael Günther, Jackmin Ong, Isabelle Mohr, Alaeddine Abdessalem, Tanguy Abel, Mohammad Kalim Akram, Susana Guzman, Georgios Mastrapas, Saba Sturua, Bo Wang, et al. Jina embeddings 2: 8192-token general-purpose text embeddings for long documents. *arXiv preprint arXiv:2310.19923*, 2023.

[50] Ofir Press, Noah A Smith, and Mike Lewis. Train short, test long: Attention with linear biases enables input length extrapolation. *arXiv preprint arXiv:2108.12409*, 2021.

[51] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*, 2019.

[52] Hugo Touvron, Thibaut Lavril, Gautier Izacard, Xavier Martinet, Marie-Anne Lachaux, Timothée Lacroix, Baptiste Rozière, Naman Goyal, Eric Hambro, Faisal Azhar, et al. Llama: Open and efficient foundation language models. *arXiv preprint arXiv:2302.13971*, 2023.

[53] Hugo Touvron, Louis Martin, Kevin Stone, Peter Albert, Amjad Almahairi, Yasmine Babaei, Nikolay Bashlykov, Soumya Batra, Prajjwal Bhargava, Shruti Bhosale, et al. Llama 2: Open foundation and fine-tuned chat models. *arXiv preprint arXiv:2307.09288*, 2023.

[54] Aaron Grattafiori, Abhimanyu Dubey, Abhinav Jauhri, Abhinav Pandey, Abhishek Kadian, Ahmad Al-Dahle, Aiesha Letman, Akhil Mathur, Alan Schelten, Alex Vaughan, et al. The llama 3 herd of models. *arXiv preprint arXiv:2407.21783*, 2024.

[55] Michael Glass, Mustafa Canim, Alfio Gliozzo, Saneem Chemmengath, Vishwajeet Kumar, Rishav Chakravarti, Avi Sil, Feifei Pan, Samarth Bharadwaj, and Nicolas Rodolfo Fauceglia. Capturing row and column semantics in transformer based question answering over tables. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, (NAACL-HLT)*, pages 1212–1224, 2021.

[56] Kalervo Järvelin and Jaana Kekäläinen. Cumulated gain-based evaluation of IR techniques. *ACM Transactions on Information Systems, (TOIS)*, 20(4):422–446, 2002.

[57] Shahul Es, Jithin James, Luis Espinosa Anke, and Steven Schockaert. RAGAS: Automated evaluation of retrieval augmented generation. In *Proceedings of the 18th Conference of the European Chapter of the Association for Computational Linguistics: System Demonstrations*, pages 150–158, 2024.