Frankfurt University of Applied Sciences

– Faculty of Computer Science and Engineering –

Evaluating the Role of Chunk Size in Retrieval-Augmented Generation (RAG)

Thesis submitted in order to obtain the academic degree

Bachelor of Science (B.Sc.)

submitted on 21. January 2024 by

Md Jewel Rana

Student ID: 1398135

First Supervisor	:	Prof. Dr. Egbert Falkenberg
Second Supervisor	:	Prof. Dr. Eicke Godehardt

Declaration

Hereby I assure that I have written the presented thesis independently and without any third party help and that I have not used any other tools or resources than the ones mentioned/referred to in the thesis.

Any parts of the thesis that have been taken from other published or yet unpublished works in terms of their wording or meaning are thoroughly marked with an indication of the source.

All figures in this thesis have been drawn by myself or are clearly attributed with a reference.

This work has not been published or presented to any other examination authority before. I am aware of the importance of the affidavit and the consequences under examination law as well as the criminal consequences of an incorrect or incomplete affidavit.

Frankfurt, 21. January 2024

Your Signature

Md Jewel Rana

Contents

List of Figures

List of Tables

Abstract

Chapter 1

Introduction

1.1 Background and Context

Introduction to Large Language Models

Large Language Model (LLM) are advanced AI systems capable of analyzing and generating human language. They can intelligently analyze text to produce coherent responses and execute various language-related tasks. In the business world, LLMs can unlock valuable insights from vast amounts of text data, automate time-consuming tasks, and enhance customer experiences through more personalized and efficient interactions.[20]

Limitations of Standalone Large Language Models

However, despite their power, LLM face critical limitations when they are used as a standalone system. In many cases, LLM generate responses that sound plausible but are factually incorrect. This phenomenon is known as hallucination. This occurs because LLM rely on pre-trained data, which may not contain all the necessary knowledge to accurately answer domain-specific questions. [6][40] Additionally, pre-trained LLM are limited by the static knowledge they acquired during training. They cannot access recent developments or specialized domain knowledge that wasn't included in their training data. [6][40] Furthermore, many real-world applications, such as legal, medical, or technical fields, require domain-specific expertise. LLM alone may lack the depth that may necessary to handle such specialized queries effectively. [6] Moreover, in question-answering tasks, users often expect accurate, up-to-date, and contextually relevant responses. [40]

Introduction to the Retrieval-Augmented Generation

To overcome these limitations, a new paradigm called Retrieval-Augmented Generation (RAG) has been introduced. RAG is a framework which is designed to enhance the performance of LLM by incorporating an external mechanism. Instead of completely depending on the pre-trained knowledge base within the model, RAG retrieves relevant document chunks from external knowledge bases based on semantic similarity calculations. This enables LLM to access and incorporate up-to-date, domain-specific information into their responses. By referencing external knowledge, RAG addresses a significant limitation of traditional LLM: their tendency to generate factually incorrect content when they are faced with queries that are not in their training data. The retrieved information from the external knowledge base grounds the LLM's responses, reducing the problem of hallucinations. Due to this integration, RAG has become a widely adopted technology, particularly in advancing chatbot capabilities and improving the practical applicability of LLM in real-world applications.[6]

Challenges in RAG Implementation

However, with the development of RAG technique, new challenges have emerged. A reliable retrieval pipeline is crucial for building effective RAG-based applications. The quality of the final answer highly depends on the relevance of the retrieved text to the user's query. If the retriever fails to identify relevant information from the knowledge bases, the LLM may generate inaccurate or misleading responses.[29] One of the most crucial factors that directly influences the performance of semantic retrieval is chunk size.[15] In the chunking process, a large corpus of text data is divided into smaller and semantically meaningful units.[43] Smaller chunks contain more focused and context-specific information, improving the relevance of retrieved results. However, they may lack the broader context necessary to answer certain queries. Conversely, larger chunks include more context, some of which may not be relevant to the specific query. This can lead to less accurate similarity scores and potentially less useful results for RAG.[15]

1.2 Motivation

The evaluation of the impact of chunk size on RAG systems is highly significant because it directly affects the performance and reliability of applications such as chatbots and QA systems. In RAG-based systems, LLM are enhanced by integrating external knowledge sources which enable them to generate more accurate and contextually relevant responses. [30]
However, the effectiveness of this integration can be influenced by the chunk size, or how the corpus data is segmented. [6] In real-world applications like chatbots, the precision and relevance of responses are crucial. Improper chunking may lead to the retrieval of irrelevant or incomplete information which results in responses that are factually incorrect or lack coherence. [40] For example, in customer support scenarios, a chatbot must access specific sections of the knowledge base to accurately answer user queries. Because of large chunk size the system may retrieve unnecessary information which may overwhelm the user with irrelevant details. [15] On the other hand, if the chunks are too small, critical context may be missed by the retrieval system which may lead to incomplete answers. [30] Effective chunking ensures that the system can extract relevant documents without being misled by irrelevant content. [40] This precision affects the system's ability to provide more accurate and concise answers. [6]

Therefore, understanding and optimizing chunk size is essential for improving the performance of RAG-based applications. By ensuring optimal chunk sizes, developers can improve the accuracy of information retrieval and build efficient AI-driven communication tools. [30]

1.3 Research Problem

1.3.1 Main Research Question

The main question of this research is to evaluate the influence of chunk size on the accuracy, relevance, and efficiency of responses generated by RAG models in open-domain question answering.

1.3.2 Supporting Questions

In order to thoroughly address this primary question, the following supporting questions will also be explored:

- **Impact on Factual Accuracy:** How does chunk size affect the factual accuracy of RAG-generated answers?
- **Retrieval accuracy:** How does chunk size influence retrieval efficiency and accuracy? Specifically, does using smaller chunks yield more relevant context from vector search compared to larger chunks?
- **Mitigating Hallucinations:** Are shorter or longer chunk size more effective in minimizing hallucinations in generated responses?

- **Recommended Chunk Size:** What is a recommended chunk size to effectively balance retrieval efficiency and response quality in RAG models?
- **Performance Metrics:** Does chunk size impact the generation speed and real-time applicability of RAG-based systems?
- **Context vs. Irrelevance:** How do larger chunks compare to smaller ones in contributing irrelevant details that may affect response quality?

1.4 Objectives

The objectives of this thesis are designed to systematically investigate the role of chunk size on the performance of RAG based systems. Each of these objectives is specifically tailored to assess a crucial aspect of model performance:

1.4.1 Evaluate Chunk size and Accuracy

One of the primary objectives of this thesis is to assess how different chunk sizes, such as short, medium, and long, affect the quality and factual accuracy of RAG-generated responses. Chunk size directly impacts the contextual information available to the model during generation.[15] This objective focuses on quantifying the effects of chunk size on accuracy through rigorous experimentation. In the evaluation, metrics such as Precision, Recall, F1-score, BLEU, BERT score, and ROUGE will be employed to assess the impact of chunk size on RAG based system.

1.4.2 Analyze Hallucination Rates

RAG systems, like many other generative models often produce hallucinations—responses containing fabricated or unsupported or irrelevant information. These hallucinations can undermine the trustworthiness and usability of the system.[31] This objective intends to evaluate and classify the prevalence of hallucination as a function of document length. Hallucinations will be categorized into two classes:

- **Intrinsic hallucinations**, these errors may arise from the generative model, often resulting from pre-trained biases or insufficient contextual understanding.[31]
- **Extrinsic hallucinations**, these errors are a consequence of the system's reliance on incomplete, irrelevant, or noisy information extracted from its knowledge base.[31] The system's inability to accurately identify and filter out such information can lead to the generation of misleading outputs.

1.4.3 Assess Retrieval and Generation Speed

Efficiency is a vital factor in the development of RAG based systems, most importantly in real-time applications. Different chunk sizes impact the speed of retrieval and response generation.[33] The aim of this objective is, to measure the retrieval times and the generation speeds for each chunk size to find out the trade-offs between speed and quality, that are highly required in real-time applications. These assessments will provide insights into the feasibility of RAG systems for real-time use cases.

1.5 Thesis Structure

Provide an overview of how the thesis is organized.

Chapter 2

Literature Review

2.1 Foundational Studies

In this section the key foundational works that form the basis of RAG systems will be explained briefly. Focus will be on two particularly relevant studies. These studies provide a strong foundation for understanding the core concepts and challenges in RAG research.

Mix-of-Granularity: Optimize the Chunking Granularity for Retrieval-Augmented Generation

Zijie Zhong et al.[44] introduces the Mix-of-Granularity (MoG) approach. In this approach chunk sizes are dynamically adjusted based on the query requirements to optimize RAG performance. This study highlights the trade-offs between retrieval efficiency and response quality. A hierarchical retrieval framework is used to achieve MoG that combines smaller chunks for precision with larger chunks for context, which dynamically balances granularity to suit different tasks. This study establishes a strong evidence that chunk size directly impacts retrieval speed, hallucination rates, and the overall factual accuracy of generated responses. This research provides critical insights into the effects of chunk granularity, that forms a theoretical basis for examining chunk size systematically.

Financial Report Chunking for Effective Retrieval-Augmented Generation

Another Study [13], which was conducted by Antonio Jimeno Yepes et al. that experimented chunking strategies specifically in the context of financial documents. Their study highlights the impact of chunk size on RAG systems, particularly while dealing with complex, structured data. They propose methodologies for chunking financial reports into semantically coherent segments, which demonstrate that appropriate chunk sizes improve retrieval relevance and generative accuracy while reducing hallucinations. This work also highlights the challenges in processing lengthy financial documents, that emphases the need for tailored chunking strategies to ensure performance in domain-specific applications.

These foundational studies lay the groundwork for understanding how chunk size affects RAG models, particularly in diverse and complex domains such as open-domain QA and structured document retrieval.

2.2 Related Work

This section explores relevant literature that intersects with the research focus, which offer insights into document retrieval, generative systems, and the specific influence of document characteristics like length or chunk size.

Document Retrieval and QA Systems

Extensive research has been carried out to improve document retrieval methods for question-answering (QA) systems. Works like "End-to-End Open-Domain Question Answering with Retrieval-Augmented Generation"[10] examine how retrieval quality directly impacts the accuracy and relevance of QA outputs. Additionally, advancements in vector-based retrieval models, such as ColBERT[14], emphasize the importance of granularity in retrieval tasks. These studies highlight the importance of retrieval units, such as chunks or passages, that shape the performance and accuracy of QA systems.

Impact of Document Length on Model Performance

Research such as "Document Length and Relevance in Neural Information Retrieval"[18] and "Chunking Strategies in Long Document Processing"[4] analyze how document length and chunk size affect model performance. The findings of these studies suggest that shorter documents generally provide clearer and more focused context, whereas longer documents are more likely to include unnecessary details that can weaken the quality of responses. The highlight of these studies suggests how to find the balance between extracting rich contextual information and minimizing noise, which is crucial for determining optimal chunk sizes in RAG systems.

Hallucinations in Generative Models

The problem of hallucinations, where generative models produce incorrect or fabricated information, has drawn considerable attention. Research by Ji et al.[12] categorizes hallucinations into intrinsic and extrinsic types as well as explores strategies to mitigate them. In the context of RAG, studies such as "Mitigating Hallucinations in Retrieval-Augmented Generation"[22] show how retrieval quality and document characteristics contribute to minimize hallucination rates. These insights provide a foundation for examining how chunk size influences the probability of hallucinations in RAG-generated responses.

This literature review provides the theoretical basis for this thesis by connecting foundational studies with recent advancements in document retrieval and generative systems. The findings of these studies suggest a critical need to evaluate chunk size as a factor that influences RAG performance, to address key issues such as retrieval efficiency, response quality, and mitigation of hallucination.

2.3 Gap Analysis

Although Retrieval-Augmented Generation(RAG) has shown significant potential in improving the quality of response generated by LLM, it's unclear what effect different chunk sizes have on retrieval relevance, generative accuracy, and hallucination mitigation. Existing studies, such as those by Zijie Zhong et al. [44] and Antonio Jimeno Yepes et al.[13], focus on specific applications but do not provide an unified framework for evaluating chunk size across diverse contexts. Moreover, the relationship between chunk size and hallucination mitigation, specifically its impact on the grounding of generative outputs, requires further investigation. Furthermore, the trade-off between context richness and noise introduced by varying chunk sizes has not been systematically analyzed. This research aims to fill these gaps by offering a systematic evaluation of the impact of chunk size in RAG systems, with experiments performed on Q & A datasets to address key issues such as retrieval efficiency, response quality, contextual accuracy and the hallucination rates.

Chapter 3

Theoretical Background

3.1 RAG architecture

Retrieval-Augmented Generation (RAG) expands the abilities of Large Language models by enabling them to incorporate real-world data and dynamic knowledge.[6, 3] While language models like GPT-3 can generate impressive text, their performance is constrained by the specific information they were trained on. Additionally, their context window[17], which refers to the limited amount of text they can process at once, hinders their ability to effectively utilize vast knowledge bases. RAG addresses this limitation by adding a retrieval layer that searches a knowledge base for relevant information, which is then combined with the original query before being sent to the language model.[3]

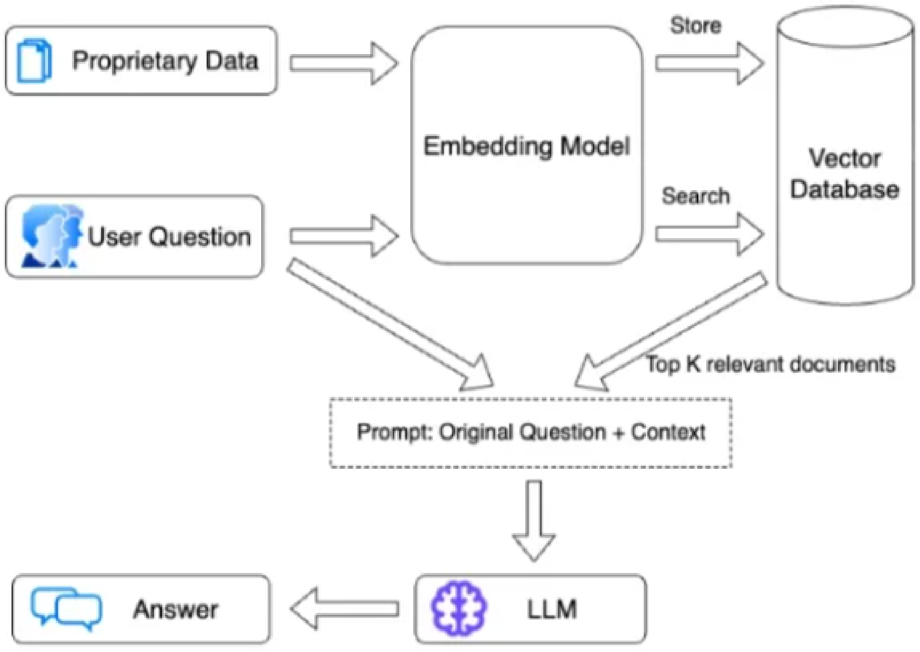

Figure 3.1: RAG architecture [28]

This allows the system to provide accurate, up-to-date, and contextually relevant responses without requiring the language model to memorize all the data. RAG utilizes vector databases to store data as numerical representations called embeddings. This enables efficient similarity searches, allowing the system to quickly identify the most pertinent information from large-scale datasets. By providing the language model with only the most relevant chunks, RAG ensures that the generated insights are accurate and relevant.

The foundation of a RAG architecture lies in two key components: the **retriever** and the **generator**.[3] To understand how it works, let's take a closer look at the roles of each.

Retriever

The Retriever is a component in RAG Architecture that searches through a large database of vector embeddings using similarity search techniques to find the most relevant information for a given query. The initial step involves organizing the data through indexing. This process includes loading documents, dividing them into smaller chunks, and transforming these chunks into numerical representations known as vector embeddings. The vector embeddings are stored in a knowledge base for future reference. When a query is submitted, a similar vectorization process is performed, where the query is transformed into a numerical representation. This process ensures compatibility with the vectorized documents, which enables the system to retrieve relevant documents based on the similarity of their vector representations. To represent words numerically various techniques employed. Sparse vector methods for instance TF-IDF assign numerical values based on word frequency and importance. Dense embedding models, such as BERT which generates more complex numerical representations. These techniques can be combined to achieve even more accurate results.[3]

Generator

The Generator uses the retrieved relevant chunks to produce a final answer by synthesizing and expressing the information in natural language. It depends on a LLM such as GPT, BART which are trained on massive datasets to generate human-like text. The generator takes both the query and the relevant documents which was retrieved by the retriever as an input to produce a coherent and informative response.[3] Effective prompt engineering plays a crucial role in order to provide the generator proper guidelines to focus on the right context and produce precise outputs, especially when complex or ambiguous queries are processed. By carefully designing prompts, developers can better guide the LLM to utilize the retrieved knowledge effectively and ensure the generated response aligns with the intended requirements.[1]

3.2 Sentence Embeddings

Sentence embeddings are numerical representations of sentences, where each sentence is assigned to a point in a high-dimensional vector space. Unlike traditional methods like bag-of-words or TF-IDF, which focus mostly on the presence or frequency of individual words, sentence embeddings capture the semantic meaning of the entire sentence. These embeddings enable the comparison of sentences based on their semantic similarity, which facilitates tasks such as text similarity, clustering, semantic search, and question-answer retrieval.[35]Sentence embeddings are generated by transforming text into high-dimensional vectors. The closer two sentences are in this vector space, the more semantically similar they are. For instance, the sentences 'What is AI?' and 'Explain artificial intelligence' would produce embeddings that are close together because they express similar meanings.[26] Sentence embeddings are essential for many NLP applications that require an understanding of the semantic meaning of sentences.[34]

3.3 Bag of N-grams

The Bag of N-Grams Model is a method in NLP, which is used to represent text as a vector, which are contiguous sequences of words or characters. When n=1, these are referred to as uni-grams, representing individual words from the text. Let's consider two sentences "the cat sat on the mat" and "the dog sat on the mat". The Vector representation of these two sentence is following.[11]

Uni-gram Vocabulary	**The cat sat on the mat**	**The dog sat on the mat**
The	1	1
cat	1	0
dog	0	1
sat	1	1
on	1	1
the	1	1
mat	1	1

Table 3.1: Uni-gram Frequency Representation of Two Texts.

3.4 Sentence Transformer

One of the popular Python libraries for computing dense vector representations of sentences is SentenceTransformer, which facilitates various NLP tasks.[25] It is equipped with the pre-trained model **"all-MiniLM-L6-v2"**, which ensures a balance between computational efficiency and performance. This model is based on a distilled version of BERT and outputs 384-dimensional vector embeddings that capture the semantic meaning of entire sentences.[32][39]

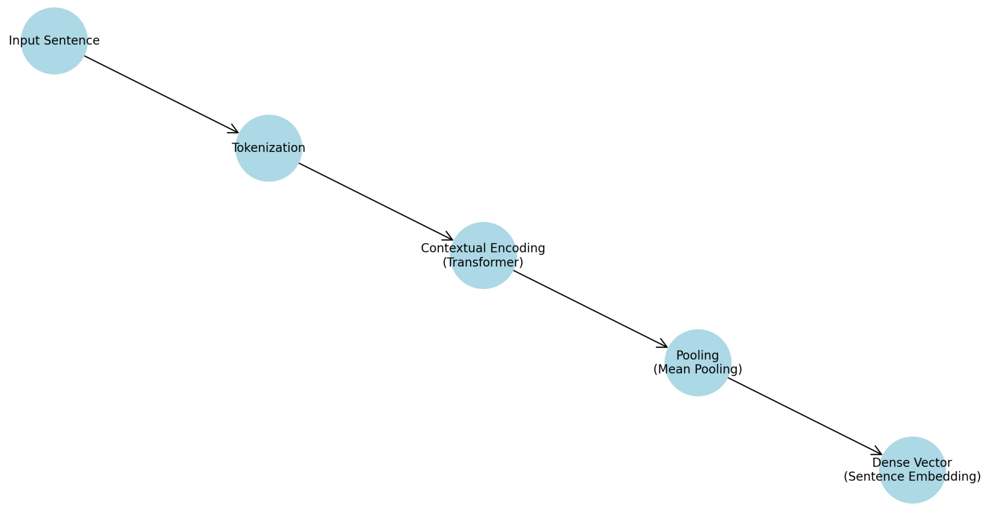
Input Sentence
Tokenization
Contextual Encoding (Transformer)
Pooling (Mean Pooling)
Dense Vector (Sentence Embedding)

Figure 3.2: Working Procedure of SentenceTransformer

The model starts by breaking the sentence into tokens and using contextual encoding to understand semantic relationships. Afterward, it applies mean pooling to the token embeddings, resulting in a concise and meaningful representation of the sentence.[24]

3.5 Vector Search

Vector search is a method for exploring unstructured data by measuring the similarity between vector embeddings. These vector embeddings are numerical representations of data, such as text, documents, images, or audio. Vector search uses similarity measures to identify patterns and retrieve relevant information, which is useful for text-based applications like semantic search, recommendation systems, or information retrieval systems. Vector Search relies on similarity measures such as Euclidean Distance, Dot Product, or Cosine Similarity to compare vector embeddings. The choice of method depends on the nature of the dataset and the specific task requirements.[19] In the following section, we will discuss Cosine Similarity in more detail, as it is also used in the implementation of the retrieval system for this study.

3.5.1 Cosine Similarity

Cosine Similarity measures how similar two vectors are by looking at their direction, not their length. It is particularly suitable for text embeddings as it can capture semantic similarities while ignoring vector magnitude, which may vary due to sentence length or structure.[36] This measure produces a similarity score between -1 and 1, where higher values indicate closer alignment, which makes it ideal for text-based applications like document search.[19] For vector a and b with dimensions n, cosine similarity can be defined as:

$$\cos\theta = \frac{(a_1 \cdot b_1) + (a_2 \cdot b_2) + \cdots + (a_n \cdot b_n)}{\sqrt{(a_1^2 + a_2^2 \ldots a_n^2) \cdot (b_1^2 + b_2^2 \ldots b_n^2)}}$$

3.6 Chunking in RAG

3.6.1 Introduction to Chunking

Chunking, also called LLM chunking, which involves breaking down large amounts of text into smaller, more manageable chunks.[21] By breaking down textual data into coherent paragraphs, sentences, or token-limited segments, chunking facilitates precise query matching while maintaining logical coherence and reducing noise.[2] This allows each chunk to be individually indexed and easily retrieved when required. This technique significantly improves RAG systems by making retrieval faster, more accurate, and by ensuring that information is evenly distributed across the entire dataset. In practice, as Lewis et al.[16] showed, splitting a massive dataset like Wikipedia into 100-word chunks creates millions of smaller, searchable documents for retrieval process.[21] This highlights the importance of chunking for handling large datasets, maintaining context, and providing relevant answers.[2]

3.6.2 Chunking Strategies

RAG systems use various chunking strategies to optimize information retrieval and processing. Some of the chunking strategies, which are most commonly used in RAG system are, Fixed-Size Chunking, Recursive-Based Chunking, Document-Based Chunking, Semantic Chunking, and Hybrid Chunking. For this study, Recursive-Based Chunking is used. In the following section, this chunking strategy will be discussed in detail.

Recursive-Based Chunking

This method uses hierarchical separators, such as paragraphs, sentences, or specific markers, to divide text into meaningful chunks. It repeatedly applies these rules until the chunks reach a certain size, without changing the original structure. Although it offers flexibility and maintains semantic integrity, its complexity and computational requirements make it less efficient for processing large datasets.[21]

3.6.3 Impact of chunk size on RAG

The chunks size has a pivotal role in both the retrieval and generation stages of a RAG system. Finding the right balance in chunk size is essential to ensure that the system performs in an optimal way. As chunk size has direct impact on the granularity of retrieved information and the coherence of generated responses.[5, 38]

Impact on Information Retrieval

Larger Chunks: Larger chunks contain more context, which makes them effective for the retrieval of comprehensive information relevant to answer complex queries. This large context can include extra details that may be critical for understanding.[44, 38] But the inclusion of extra information may reduce retrieval granularity, which may lead to the retrieval of unnecessary content. This makes larger chunks less precise for specific or targeted queries.[13, 38]

Smaller Chunks: Smaller chunks allow the retrieval of more precise and specific pieces of information. This makes them ideal for the scenarios where accurate and targeted information is needed.[44, 37] However, context within smaller chunks is limited, which may result in incomplete information retrieval. This reduces their usefulness for broad-context queries.[5, 38]

Impact on Generation

Larger Chunks: Larger chunks provide the model more context during the generation, as a result the risk of hallucinations or incomplete responses can be reduced. This is especially useful for generating detailed outputs.[13] But the larger context may sometimes overwhelm the model, which may lead to exceeding the input token limit of the language model. This can result in truncation or inefficiencies in processing.[44, 38]

Smaller Chunk: Smaller chunks provide precise and focused information to the model, which can improve the precision of generated responses. They are effective for answering simple queries requiring straightforward answers.[5, 38] However, due to lack of sufficient context, smaller chunks may lead to less accurate or simplistic answers.[13, 38]

Impact on retrieval time and response time

Response time is directly influenced by the chunk size in RAG. Larger chunk sizes provide more context for the LLM, improving response quality but potentially increasing processing time. This trade-off between deeper contextual understanding and responsiveness becomes more pronounced with larger chunk sizes.[37]

The size of the chunks also highly affects the retrieval speed. Because it influences how frequently the system needs to search and the complexity of those searches. Larger chunks are more resource-intensive, whereas smaller chunks can lead to more frequent searches.[38]

3.7 Few Shot Prompting

Few Shot Prompting provides the LLM with multiple examples to clarify the task and enhance its understanding, which improves the quality of its responses. This approach demonstrates the expected input-output relationship through concrete examples, which enables the model to align its output with the desired format and context.[27]

3.8 Evaluation Metrics

In order to evaluate RAG systems nuanced metrics to measure the effectiveness of both the retrieval and generation components are required. The choice of metrics directly impacts how well these systems align with real-world requirements. Below is a detailed overview of evaluation metrics that will be used to evaluate our RAG system.

3.8.1 Retrieval Metrics

Precision

Measures the fraction of retrieved chunks that are relevant to answer the query.[41]

$$\text{Precision} = \frac{\text{Number of relevant retrieved chunks}}{\text{Total number of retrieved chunks}} \tag{3.1}$$

Recall

Recall calculates the system's ability to get all relevant chunks for a given query, ensuring completeness.[41]

$$\text{Recall} = \frac{\text{Number of relevant retrieved chunks}}{\text{Total number of relevant chunks in the knowledge base}} \tag{3.2}$$

3.8.2 Generation Metrics

BLEU

The BLEU score calculates the quality of LLM generated text by comparing it with reference text. It evaluates n-gram overlap between reference text and generated text with brevity penalties for overly short responses.[41, 8]

$$p_n = \frac{\text{Count of overlapping n-grams}}{\text{Total n-grams in candidate}} \tag{3.3}$$

$$B.P = \exp(1 - \frac{r}{c}) \text{ if } (c < r) \tag{3.4}$$

$$\text{BLEU} = BP \cdot \exp\left(\frac{1}{N}\sum_{n=1}^{N} \log P_n\right) \tag{3.5}$$

Here, B.P = Brevity penalties, c = candidate length, r = reference length, P_n = Precision for n-grams

ROUGE

ROUGE assesses the overlap between system-generated text and human reference texts, frequently employed in summarization tasks.[41] There are various ROUGE variants, such as ROUGE-1, ROUGE-S, ROUGE-L, and ROUGE-N. However, for our experiment, we will focus exclusively on ROUGE-1. ROUGE-1 measures overlap of uni-grams between the generated text and reference[9]

$$ROUGE - 1 = \frac{\text{Overlapping unigrams}}{\text{Total unigrams in candidate}} \tag{3.6}$$

BERTScore

Unlike traditional metrics like BLEU or ROUGE, which rely on n-gram overlaps, BERTScore uses contextual embeddings from transformer-based models like BERT to capture the semantic similarity between the two texts.[42, 41]

$$R = \frac{1}{|R|}\sum_{r\in R} \max_{c\in C} \text{cosine\_similarity}(E_r, E_c) \tag{3.7}$$

$$P = \frac{1}{|C|}\sum_{c\in C} \max_{r\in R} \text{cosine\_similarity}(E_c, E_r) \tag{3.8}$$

$$F_1 \text{ score} = \frac{2 \cdot P \cdot R}{P + R} \tag{3.9}$$

Where, C = Tokens in the candidate, R = Tokens in the reference, E_c, E_r : Embeddings of candidate and reference tokens, P = Precision, and R = Recall.

Human Evaluation

Human evaluation remains the best standard for assessing generative quality, Human can manually judge coherence, relevance, and fluency.

LLM as a Judge

LLM can also be used to evaluate generated text based on detailed prompts and criteria such as coherence and fluency.[41]

3.8.3 Speed Metrics

Retrieval time

This measures the time taken to fetch relevant chunks from the knowledge base.[38]

Response time

This evaluates the time taken for the language model to produce a response after retrieving the context for a given query.[37]

Chapter 4

Methodology

4.1 High Level Overview

This study investigates the influence of chunk size on the performance of RAG system, with a focus on key metrics including response accuracy, retrieval accuracy, hallucination rate, response time, and retrieval time. To facilitate this analysis, a knowledge base was constructed using the **"christti/squad-augmented-v2"** dataset, where the first 820 unique contexts were concatenated in order to form a single corpus. After analyzing the length of all the contexts, four distinct chunking strategies—page, small, medium, and large—were developed, and a series of text pre-processing steps were performed on the resulting chunks. Vector embeddings were calculated for each chunk by using `Sentence Transformer` and stored them in a local database. The RAG architecture was implemented by selecting the most relevant chunks based on cosine similarity with the user query's embedding and generating responses for each chunk type. An Evaluation dataset was prepared by using a sample of 90 randomly selected questions from the test dataset. The performance of RAG system was assessed across metrics such as accuracy, retrieval time, and response time for evaluating the effect of chunking strategy on generated responses. In the following sections, each step of the methodology will be elaborated in detail to provide a comprehensive understanding of the research process.

4.2 Preparing the Knowledge Base

4.2.1 Dataset Selection

The **"christti/squad-augmented-v2"** dataset, which is an extension of the Stanford Question Answering Dataset (SQuAD), is an excellent choice for evaluating RAG system. This dataset contains a diverse collection of human-created contexts, each of them is paired with a question and a corresponding answer. These contexts cover a wide range of topics and are semantically rich and coherent. With so many different kinds of context, different chunk sizes can be experimented without losing the semantic meaning. Furthermore, the dataset is aligned with real-world question-answering scenarios and it is highly adopted in NLP research, which establish it as a trustworthy and dependable benchmark for assessing retrieval accuracy, contextual response quality, and hallucination rate. Moreover, its structured format simplifies text pre-processing and embedding generation.[23]

4.2.2 Corpus Construction

The dataset contains a total of 100,132 distinct contexts. For the purpose of this study, the first 820 contexts were selected as the basis for analysis. In order to construct the knowledge base, a corpus was formed, which is defined as a structured collection of texts.[7] This corpus was generated by concatenating all the selected contexts into a single unified text. The following code demonstrates the process of corpus construction.

```python
# Load the dataset
ds = load_dataset("christti/squad-augmented-v2")
df = pd.DataFrame(ds["train"])

# Select first 820 unique contexts
contexts = df['context'].unique()[:820]

# construct the corpus
corpus = ' '.join([context for context in contexts])
```

4.2.3 Chunking Strategy

To facilitate RAG experiments, the corpus was divided into different chunk sizes. In order to evaluate the impact of chunk size on RAG this study considers four types of chunking strategies: **page**, **small**, **medium**, and **large**.

Page Chunking

Each row in the dataset represents an unique context, which is treated as a single page. This approach aligns with one of the commonly used chunking strategies in NLP tasks, where each context is treated as an independent unit for retrieval and generation.

Determination of Chunk Sizes

For the sake of defining chunk sizes for small, medium, and large, the lengths of contexts in the dataset were analyzed. A descriptive statistical analysis of the character lengths was performed, which reveals the following:

Statistic	**Value**
Count	820.000000
Mean	731.360976
Standard Deviation (std)	358.499497
Minimum (min)	154.000000
25th Percentile (25%)	468.750000
Median (50%)	667.000000
75th Percentile (75%)	912.500000
Maximum (max)	3076.000000

Table 4.1: Statistical Analysis of Context Lengths in the Dataset.

Based on this analysis:

- **Small Chunks:** Defined as 250–500 characters, representing shorter passages near the 25th percentile of context lengths.
- **Medium Chunks:** Defined as 500–800 characters, corresponding to documents near the mean and median lengths.

- **Large Chunks:** Defined as 800–1,200 characters, encompassing longer documents that approach the 75th percentile or exceed the mean.

Chunk Overlaps

For ensuring the continuity of information across chunk boundaries, overlapping text was introduced. Overlaps were set proportionally to the chunk size, where the need to preserve semantic coherence was considered.

- **Small chunks:** 50–100 characters overlap.
- **Medium chunks:** 100–150 characters overlap.
- **Large chunks:** 150–200 characters overlap.

Recursive Chunking Strategy

For dividing the corpus into smaller chunks recursive chunking strategy were used, which ensures that the meaning of the text is preserved within each chunk. The `RecursiveCharacterTextSplitter` module from LangChain was utilized to implement the recursive chunking strategy.

4.2.4 Embedding Generation

After dividing the corpus into chunks of various sizes, each chunk was converted into a vector. However, before performing this conversion, comprehensive text pre-processing was applied to ensure the quality and consistency of the input data, as text pre-processing is crucial for enhancing the efficiency and accuracy of retrieval-based tasks. The pre-processing phase involved several steps, including removing HTML tags, URLs, punctuation, and stopwords, followed by applying stemming to standardize word forms and lemmatization to ensure linguistic consistency. After pre-processing, each chunk was converted into a vector using the `SentenceTransformer`, a Python module widely used for generating high-quality sentence embeddings.

4.2.5 Vector Database Construction

To facilitate efficient vector search in the RAG pipeline, a database table named `text_vectors` was created. This table functions as a storage repository for all chunked text and their associated vector representations, which facilitates efficient and streamlined retrieval processes. The table is created on a local database and is structured as follows:

Field	Type
id	int
chunk_text	text
chunk_type	enum('PAGE', 'SMALL', 'MEDIUM', 'LARGE')
embedding	blob

Table 4.2: Schema of the `text_vectors` table for vector database construction.

- **id:** A unique integer identifier for each entry, serving as the primary key.
- **chunk_text:** A column of type text that stores the chunked text data.
- **chunk_type:** An enumerated column indicating the type of chunk (PAGE, SMALL, MEDIUM, LARGE), which facilitates the identification and retrieval of specific chunk types.
- **embedding:** A binary large object (BLOB) column containing the vector representation of the corresponding chunk text.

This table design ensures that chunks of various types can be effectively retrieved, and their embeddings can be used for similarity-based operations in the RAG pipeline. The distribution of chunks by chunk type in the vector database is shown in the bar diagram below, highlighting the frequency of each chunk type (SMALL, MEDIUM, LARGE, PAGE).

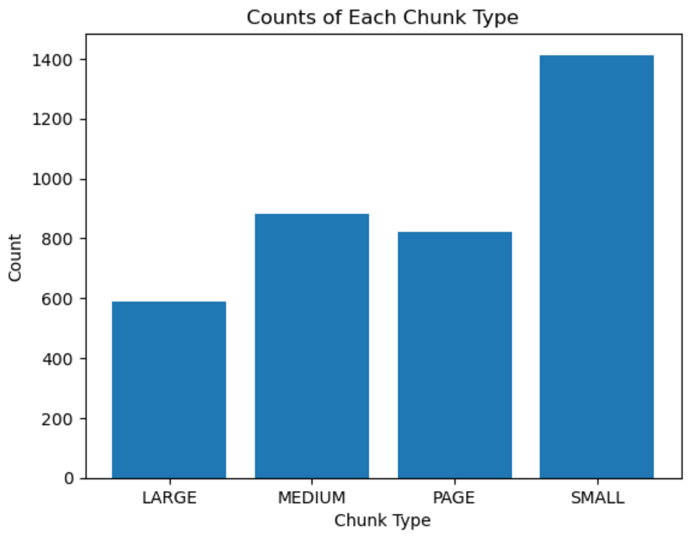

Figure 4.1: Distribution of Chunks by Chunk Type in the Vector Database

4.3 Implementation of RAG Architecture

The implementation of RAG architecture consists of two key components: **retrieval** and **generation**. These components work together to fetch relevant contextual information from a pre-constructed knowledge base and generate high-quality responses to user queries. The RAG pipeline is triggered whenever a user submits a query, initiating the retrieval and generation processes to deliver a contextually accurate response.

4.3.1 Retriever

The retrieval process identifies and retrieves the most relevant chunks from the vector database based on the user query and specified chunk type. The following code snippet demonstrates the retrieval process:

Listing 1 Python code snippet: Implementation of retrieval process

```python
def get_context_by_chunk_type(chunk_type, query_text):
    start_time = time.time()

    # Get database connection
    connection = database_connection()
    cursor = connection.cursor()

    # get chunks by chunk type
    cursor.execute("SELECT * FROM text_vectors WHERE chunk_type=%s", (chunk_type,))

    # Retrieve chunks
    rows = cursor.fetchall()

    # Convert user query into vector
    clean_query = preprocess_text(query_text.lower())
    model = SentenceTransformer("all-MiniLM-L6-v2")
    query_vector = model.encode(clean_query)

    # Perform vector search
    similar: list[Similar_Text] = []
    for row in rows:
        # Calculate cosine similarity
        sim_score = cosine_similarity(pickle.loads(row[3]), query_vector)
        similar.append(Similar_Text(row[1], sim_score=sim_score))

    # Sort chunks by similarity score
    similar.sort(key=lambda similar: similar.sim_score, reverse=True)

    # Combine most 3 relevant chunks
    text = "/n".join([sm.text for sm in similar[:3]])
    end_time = time.time()

    # calculate retrieval time
    delta= end_time - start_time
    # return relevant chunks and retrieval time
    return {"retrieval_time": delta, "text": text}
```

This process, implemented in the `get_context_by_chunk_type()` function, follows these steps:

- **Database Query:** The function connects to the vector database and retrieves entries matching the specified chunk type (PAGE, SMALL, MEDIUM, or LARGE).
- **Query Preprocessing:** The user query is preprocessed to ensure consistency with the stored chunks by converting it to lowercase and cleaning it using text preprocessing techniques.
- **Query Vectorization:** The preprocessed query is converted into a vector representation using the `SentenceTransformer`.
- **Cosine Similarity Calculation:** The similarity score between the query vector and each chunk's vector is computed to determine relevance.
- **Sorting and Selection:** The chunks are sorted by similarity scores in descending order, and the top three chunks are concatenated to form the context for the generation component.

4.3.2 Generator

The generation component builds on the retrieved chunks by using them as input to a language model to produce a response to the user query. The following code snippet demonstrates the generation process:

Listing 2 Python code snippet: Generating response for a query.

```python
def chat_with_ai(user_query: str, chunk_type: str, ref_ans: str):
    # Get OpenAI API key
    openai_api_key = os.environ["OPENAI_API_KEY"]

    # Get relevant chunks
    result = get_context_by_chunk_type(chunk_type, user_query)

    # Format the prompt
    prompt_template = ChatPromptTemplate.from_template(PROMPT_TEMPLATE)
    prompt = prompt_template.format(context=result["text"], question=user_query)

    # Initialze LLM( modle gpt-4o )
    model = ChatOpenAI(openai_api_key=openai_api_key, model_name="gpt-4o")

    # Generate response
    start_time = time.time()
    response_text = model.invoke(prompt)
    end_time = time.time()

    # Calculate response time
    delta = end_time - start_time

    # Return response time and generated response
    return {"response_time": delta, "response": response_text.content}
```

The function includes the following steps:

- **Chunk Integration:** The retrieved chunks are integrated into a predefined prompt template that combines context, user query, and additional instructions.
- **Prompt Construction:** The `ChatPromptTemplate` from LangChain dynamically formats the prompt to include the context and query seamlessly.
- **Response Generation:** The prompt is sent to OpenAI's GPT-4 model (gpt-4o) through the `ChatOpenAI` class to generate response based on the given context.
- **Response Timing:** The function measures and returns the time taken by the model to generate the response.
- **Output Formatting:** The response text and timing information are returned for further evaluation.

4.4 Preparing Evaluation Dataset

To facilitate the analysis of how varying chunk sizes impact the responses generated by RAG system, a structured evaluation dataset was prepared. To prepare the evaluation dataset, 90 questions and their corresponding reference answers were selected from the test dataset. For each question, the system generated four distinct responses, each corresponding to a different chunking strategy: **PAGE**, **SMALL**, **MEDIUM**, and **LARGE**. The generated responses, along with additional metadata, were systematically stored in a database table named `evaluate_answer`. This table was designed with the following schema:

- **id:** A unique identifier for each record in the table, serving as the primary key.
- **ref_answer:** The reference answer corresponding to the question, serving as a benchmark for comparison.

Field	Type
id	int
ref_answer	text
chunk_type	enum('PAGE', 'SMALL', 'MEDIUM', 'LARGE')
llm_response	text
response_time	double
retrieval_time	double

Table 4.3: Schema of the `evaluate_answer` table for constructing the evaluation dataset.

- **chunk_type:** The type of chunking strategy employed for generating the LLM response.
- **llm_response:** The response generated by the LLM for the specific chunking strategy.
- **response_time:** The time taken by the LLM to generate the response, measured in seconds.
- **retrieval_time:** The time taken for the retrieval process during the RAG pipeline, also measured in seconds.

4.5 Designing an LLM Evaluation Framework

In the evaluation dataset, questions, reference answers, and four LLM-generated responses corresponding to each chunk type (small, medium, large, page) were stored. To analyze the impact of chunk size on the performance of RAG system, a series of experiments will be conducted focusing on retrieval accuracy, retrieval speed, response time, hallucination rate, and contextual accuracy. To support these experiments, the LLM-generated responses were thoroughly examined manually.

Question	Reference Answer	LLM Response
When did Beyoncé release her single 'Formation'?	On February 6, 2016, one day before her Super Bowl performance.	Formation was released by Beyoncé in February 2016.
When did Beyoncé release her single 'Formation'?	Unknown based on the Context provided.	Formation was released by Beyoncé in February 2016.
Which music video sparked speculation about Beyoncé's relationship with Jay Z?	03 Bonnie & Clyde	The music video for '03 Bonnie & Clyde' where Beyoncé appeared as Jay Z's girlfriend.
Which music video sparked speculation about Beyoncé's relationship with Jay Z?	Drunk in Love	The music video for '03 Bonnie & Clyde' where Beyoncé appeared as Jay Z's girlfriend.

Table 4.4: Examples of Questions, Reference Answers, and LLM Responses

The table above shows that in some cases, the LLM did not even attempt to answer the questions due to the lack of relevant context provided to it. In other instances, the LLM provided partial answers, while in some cases, it gave completely incorrect answers. However, there are also many instances where the LLM answered the questions accurately and completely. Based on these observations, the generated responses can be categorized into four distinct groups. This categorization provides a systematic framework for analyzing LLM performance across key metrics, such as retrieval accuracy, contextual accuracy, and hallucination rate.

- **Category 1:** This category includes responses where the LLM did not attempt to answer the question due to the absence of relevant context. This indicates inaccuracies in the retrieval process, as the similarity search failed to retrieve the necessary context for answering the question. These responses will be used to measure retrieval inaccuracy. However, they will be excluded when measuring contextual accuracy, as contextual accuracy assumes that relevant context has been retrieved. Without relevant context, contextual accuracy cannot be meaningfully assessed. Additionally, this category will not be included in calculating the hallucination rate, as the LLM did not attempt to answer the question, there is no generated content to evaluate for hallucination.

- **Category 2:** Responses in this category are partially correct but they do not provide the complete information required to fully answer the question. This suggests that either the provided context was only partially relevant, or hallucination occurred during the response generation. These responses will be considered as a retrieval success, as some relevant context was retrieved. They will also contribute to measure the hallucination rate, as the incomplete answer may indicate that the model depended on irrelevant information.
- **Category 3:** This category includes responses where the LLM answered the question completely and accurately. These responses will be treated as retrieval successes and will not be counted while measuring hallucination rate.
- **Category 4:** Responses in this category are incorrect or unrelated to the reference answer. Since the LLM attempted to answer the question, it suggests that some relevant context was retrieved, but it might have been noisy or not sufficiently relevant. Additionally, these responses suggest that hallucination occurred during the generation process. Therefore, this category will be considered as a retrieval success (as some context was retrieved) and will also contribute to calculate the hallucination rate, as the model failed to produce a meaningful response based on the given context.

To categorize responses and identify the type of hallucination, a detailed analysis of each response is needed, along with a clear understanding of the definitions for intrinsic and extrinsic hallucinations. If this is done manually, it becomes impractical and inefficient for large-scale datasets. To address this challenge, an LLM with an appropriately designed prompt was used as a judge to automatically categorize responses and determine the type of hallucination.

4.6 Implementation of LLM Evaluation Framework

A few-shot structured prompt is used to give the LLM proper guidelines to categorize the responses and to identify the hallucination type based on predefined criteria. This few-shot approach provides the model with sufficient context and examples, which enables consistent and accurate evaluations across the dataset. Below is the prompt used: **should I include the prompt here?**

```
prompt_template ="""
   You are tasked with evaluating the quality of a single response generated by a language model (LLM).
   Each evaluation involves a question, a reference answer, and one LLM-generated response. Your job is to evaluate the
   response based on the criteria provided below.
   ---
   ### **Evaluation Criteria**
   1. **Answer Accuracy**:
   - **1**: The response answers the question accurately.
   - **2**: The response partially answers the question.
   - **3**: The response cannot answer the question from the given context.
   - **4**: The response answers the question but is completely unrelated to the reference answer.

   2. **Hallucination Type**:
   - **N/A** if **Answer Accuracy** = 1 or 3.
   - If **Answer Accuracy** = 2 or 4, classify the hallucination type as:
     - **Intrinsic**: The response is incomplete or incorrect due to limitations in the generative model (e.g.,
     biases, insufficient reasoning).
     - **Extrinsic**: The response is incomplete or incorrect due to reliance on irrelevant, incomplete, or
     noisy retrieval from the context.
     - **Intrinsic Extrinsic**: Both intrinsic and extrinsic factors contribute to the hallucination.
     ---
    ### **Examples**
    #### **Example 1**
    - **Question:** What title did People magazine award Beyoncé in 2012?
    - **Reference Answer:** People magazine named her the 'World's Most Beautiful Woman' in 2012.
    - **LLM Response:** "Based exclusively on the Context provided, the title People magazine awarded Beyoncé
    in 2012 is not mentioned."

    **Example Output 2**:
    {{
    "answer_accuracy": 3,
    "hallucination_type": "N/A"
    }}
    #### **Example 2**
    - **Question:** What title did People magazine award Beyoncé in 2012?
    - **Reference Answer:** People magazine named her the 'World's Most Beautiful Woman' in 2012.
    - **LLM Response:** "Sexiest Woman of the 21st Century""

    **Example Output 2**:
    {{
        "answer_accuracy": 4,
        "hallucination_type": "Intrinsic"
    }}
    ### Your output must be in JSON Format.
    ### Output Format:
    {{
        "answer_accuracy": 1/2/3/4,
        "hallucination_type": "Intrinsic"/"N/A"/"Extrinsic"/"Extrinsic Intrinsic"
    }}
    ### Inputs:
    - **Question:** {question}
    - **Reference Answer:** {reference_answer}
    - **LLM Responses:**
        {response}

"""
```

In the prompt, a brief description of the task was provided. The criteria for evaluating answer accuracy and hallucination type were clearly explained. For better clarification, some examples with a question, reference answer, and LLM response, along with the expected output format, were included. For categorizing and identifying hallucination in each response, this prompt was dynamically populated with inputs (question, reference answer, and LLM response). The formatted prompt was passed to the LLM, which generated a JSON output containing the response accuracy and hallucination type. This consistent output format was later helped to extract the answer accuracy and hallucination type. After extracting all the answer accuracies and hallucination types, the `evaluate_answer` table (Table 4.3) was extended. Two new columns, named `hallucination_type` and `answer_accuracy`, were added to the table, which will serve as key factors in further experiments.

4.7 Evaluation Metrics: Application and Measurement

4.7.1 Retrieval & Response Time

Retrieval time was calculated in seconds during the retrieval process, as shown in (code snippet 1). It represents the time taken to retrieve all the chunks from the vector database, calculate cosine similarity, and sort the chunks based on similarity scores to extract the top 3 relevant chunks for a given user query. The numerical mean was calculated, grouped by chunk type, to compare the impact of chunk size on retrieval time.

Response time was measured in seconds during the response generation by the LLM, as shown in (code snippet 2) which represents the time taken by the LLM to generate a response for a given context and user query. The numerical mean of the response time was calculated for each chunk type for further analysis.

4.7.2 Retrieval Accuracy

Retrieval accuracy was measured using two key metrics: Precision (defined in Equation 3.8.1) and Recall (defined in Equation 3.2). To calculate precision, the number of irrelevant and relevant chunks for each chunk type was determined. Irrelevant chunks were identified as those where `answer_accuracy` was categorized as 3 (discussed in Section 4.5), indicating the retrieval of non-relevant context. Relevant chunks included those where `answer_accuracy` was categorized as 1, 2, or 4 (discussed in Section 4.5).

4.7.3 Hallucination Rate

4.7.4 Contextual Accuracy

4.8 Tech-stack

This section outlines the key tools, libraries, and hardware environment used to perform the experiments.

Tool/Framework	**Usage in the Project**
`Python`	Primary programming language used to develop and implement the project.
`datasets`	For loading the test dataset.
`pandas`	For data manipulation and data analysis.
`numpy`	For array manipulations and numerical computations.
`langchain`	To interact with OpenAI models for response generation. In order to format Prompt template. To implement recursive chunking strategy.
`flask`	For implementing REST APIs in order to make CRUD services available from other programms.
`mysql.connector`	For connecting and managing the MySQL database.
`sentence_transformers`	For generating vector embeddings from text.
`python-dotenv`	For loading environment variables stored in `.env` files.
`nltk`	For natural language processing tasks, such as BLEU score computation.
`rouge_score`	For evaluating generated text using ROUGE metrics.
`bert_score`	For evaluating the semantic similarity of text using BERT embeddings.
`requests`	For making HTTP requests in various processes.
`matplotlib`	For visualizing data and analysis results.
Ubuntu/Linux	Operating system with 8 GB Storage capacity was used to run all tools and frameworks.
VS-Code	IDE used for writing, debugging, and managing project code.
Jupyter Notebook	Interactive environment for running and visualizing Python code during development and experimentation.

Table 4.5: Tools and frameworks used in the project and their purposes.

Chapter 5

Results and Analysis

5.1 Chunk size and Factual Accuracy

5.2 Chunk size and Retrieval Accuracy

5.3 Hallucination Analysis

5.4 Retrieval and Generation Speed

Chapter 6

Discussion

6.1 Analysis of Results

6.2 Insights on Chunk Size

6.3 Practical Implications

6.4 Limitations

Chapter 7

Conclusion

7.1 Summary of Findings

7.2 Contributions

7.3 Future Work

List of Abbreviations

AI Artificial Intelligence. 2, 3, 8

API Application Programming Interface. 24

BART Bidirectional and Auto-Regressive Transformer. 8

BERT Bidirectional Encoder Representations from Transformers. 4, 8, 9, 12, 24

BLEU Bilingual evaluation understudy. 4, 12, 24

CRUD Create, Read, Update, and Delete. 24

GPT Generative pre-trained transformer. 7, 8, 19

HTML HyperText Markup Language. 16

HTTP HyperText Transfer Protocol. 24

IDE Integrated development environment (IDE). 24

JSON JavaScript Object Notation. 22

LLM Large Language Model. 2, 3, 6, 8, 10–13, 20–23

MoG Mix-of-Granularity. 5

NLP natural language processing (NLP). 8, 9, 14, 15

QA Questions & Answers. 3, 5, 6

RAG Retrieval-Augmented Generation. 2–8, 10, 11, 14–17, 19, 20

REST Representational State Transfer. 24

ROUGE Recall-Oriented Understudy for Gisting Evaluation.. 4, 12, 24

SQL Structured Query Language. 24

TF-IDF Term Frequency - Inverse Document Frequency. 8

URL Uniform Resource Locator. 16

Bibliography

[1] Xavier Amatriain. “Prompt Design and Engineering: Introduction and Advanced Methods.” In: *arXiv preprint* 2401.14423v3 (2024). Accessed on November 27, 2024. URL: https://arxiv.org/html/2401.14423v3.

[2] Sharmila Ananthasayanam. *7 Chunking Strategies in RAG You Need To Know*. Accessed October 2, 2024. 2024. URL: https://www.f22labs.com/blogs/7-chunking-strategies-in-rag-you-need-to-know/?utm_source=chatgpt.com.

[3] Sabrina Aquino. *What is RAG: Understanding Retrieval-Augmented Generation*. Accessed: 2024-03-19. 2024. URL: https://qdrant.tech/articles/what-is-rag-in-ai/.

[4] Junwei Chen, Yuxuan Xu, and Qi Zhang. “Chunking Strategies in Long Document Processing.” In: *Journal of Computational Linguistics* (2023).

[5] Paulo Finardi et al. “The Chronicles of RAG: The Retriever, the Chunk and the Generator.” In: *arXiv preprint arXiv:2401.07883* (2024).

[6] Yunfan Gao et al. “Retrieval-Augmented Generation for Large Language Models: A Survey.” In: *arXiv preprint arXiv:2301.12730* (2023). URL: https://arxiv.org/abs/2301.12730.

[7] Geeks for Geeks Teams. *NLP — Benutzerdefiniertes Korpus*. Accessed February 20, 2019. 2019. URL: https://www.geeksforgeeks.org/nlp-custom-corpus/.

[8] Geeks for Geeks Teams. *NLP – BLEU-Score zur Bewertung neuronaler maschineller Übersetzung – Python*. Accessed March 8, 2024. 2024. URL: https://www.geeksforgeeks.org/nlp-bleu-score-for-evaluating-neural-machine-translation-python/.

[9] Geeks for Geeks Teams. *Verständnis des BLEU- und ROUGE-Scores zur NLP-Bewertung*. Accessed October 4, 2024. 2024. URL: https://www.geeksforgeeks.org/understanding-bleu-and-rouge-score-for-nlp-evaluation/.

[10] Gautier Izacard and Edouard Grave. “End-to-End Open-Domain Question Answering with Retrieval-Augmented Generation.” In: *Advances in Neural Information Processing Systems (NeurIPS)*. 2021.

[11] Javatpoint. *Bag of N-Grams Model*. https://www.javatpoint.com/bag-of-n-grams-model?utm_source=chatgpt.com. Accessed: December 2, 2024. 2024.

[12] Zhijing Ji, M Lee, and et al. “Categorizing and Mitigating Hallucinations in Generative Models.” In: *Proceedings of the Association for Computational Linguistics (ACL)*. 2023.

[13] Antonio Jimeno Yepes et al. “Financial Report Chunking for Effective Retrieval-Augmented Generation.” In: *ArXiv preprint* (2024). URL: https://unstructured.io.

[14] Omar Khattab and Matei Zaharia. “ColBERT: Efficient and Effective Passage Search via Contextualized Late Interaction Over BERT.” In: *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*. 2020.

[15] Lam Hoang. *How Chunk Sizes Affect Semantic Retrieval Results*. Mar. 11, 2024. URL: https://ai.plainenglish.io/investigating-chunk-size-on-semantic-results-b465867d8ca1.

[16] Patrick Lewis et al. “Retrieval-augmented generation for knowledge-intensive nlp tasks.” In: *Advances in Neural Information Processing Systems* 33 (2020), pp. 9459–9474.

[17] Hugo Lu. *Understanding the Context Window in OpenAI Models*. Accessed: 2024-07-25. 2023. URL: https://www.getorchestra.io/guides/understanding-the-context-window-in-openai-models#:~:text=The%20context%20window%20refers%20to%20the%20maximum%20amount,models%2C%20each%20with%20different%20capabilities%20and%20context%20windows..

[18] Sean MacAvaney, Robbie McDonald, and Andrew Yates. "Document Length and Relevance in Neural Information Retrieval." In: *Journal of Information Retrieval* (2020).

[19] Nathan Crone Neil Kanungo. *How Vector Databases Search by Similarity: A Comprehensive Primer*. Accessed: 2023-08-25. 2023. URL: https://medium.com/kx-systems/how-vector-databases-search-by-similarity-a-comprehensive-primer-c4b80d13ce63.

[20] Patricia Kelbert, Dr. Julien Siebert, Lisa Jöckel. *Was ist ein Large Language Model (LLM)?* Dec. 12, 2023. URL: https://www.iese.fraunhofer.de/blog/large-language-models-ki-sprachmodelle/ (visited on 11/22/2024).

[21] Rahul. *A Guide to Chunking Strategies for Retrieval Augmented Generation (RAG)*. Accessed May 15, 2024. 2024. URL: https://zilliz.com/learn/guide-to-chunking-strategies-for-rag?utm_source=chatgpt.com.

[22] Vatsal Raina and Mark Gales. "Mitigating Hallucinations in Retrieval-Augmented Generation." In: *ArXiv preprint* (2024). URL: https://arxiv.org/abs/2405.12363.

[23] P Rajpurkar. "Squad: 100,000+ questions for machine comprehension of text." In: *arXiv preprint arXiv:1606.05250* (2016).

[24] Nils Reimers and Iryna Gurevych. *SentenceTransformers: Architecture and Pooling Strategies*. Accessed November 27, 2024. 2019. URL: https://www.sbert.net/docs/usage.html#sentence-embeddings.

[25] Nils Reimers and Iryna Gurevych. *SentenceTransformers: Multilingual Sentence Embeddings using BERT and RoBERTa*. Accessed November 27, 2024. 2019. URL: https://www.sbert.net.

[26] Oscar Sanseviero. *Introduction to Sentence Embeddings*. Accessed November 27, 2024. 2023. URL: https://osanseviero.github.io/hackerllama/blog/posts/sentence_embeddings/.

[27] Sander Schulhoff. *Few Shot Prompting*. Accessed August 7, 2024. 2024. URL: https://learnprompting.org/de/docs/basics/few_shot.

[28] Hazal Şimşek. *RAG architecture*. Accessed: 2024-11-24. 2024. URL: https://research.aimultiple.com/retrieval-augmented-generation/#easy-footnote-bottom-1-745351.

[29] Ishneet Sukhvinder Singh et al. "ChunkRAG: Novel LLM-Chunk Filtering Method for RAG Systems." In: *arXiv preprint arXiv:2410.19572* (2024). DOI: 10.48550/arXiv.2410.19572. URL: https://doi.org/10.48550/arXiv.2410.19572.

[30] Ishneet Sukhvinder Singh et al. "ChunkRAG: Novel LLM-Chunk Filtering Method for RAG Systems." In: *arXiv preprint arXiv:2410.19572* (2024). DOI: 10.48550/arXiv.2410.19572. URL: https://doi.org/10.48550/arXiv.2410.19572.

[31] DL Staff. "Understanding Intrinsic and Extrinsic Hallucinations in NLP." In: *Communications of the ACM* 66.12 (2023), pp. 202–215. URL: https://dl.acm.org/doi/pdf/10.1145/3632410.3633297.

[32] Hugging Face Team. *Pretrained Model: all-MiniLM-L6-v2*. Accessed November 27, 2024. 2023. URL: https://www.sbert.net/docs/pretrained_models.html#all-minilm-l6-v2.

[33] LlamaIndex Team. *Evaluating the Ideal Chunk Size for a RAG System Using LlamaIndex*. Accessed: 2024-11-23. 2023. URL: https://www.llamaindex.ai/blog/evaluating-the-ideal-chunk-size-for-a-rag-system-using-llamaindex-6207e5d3fec5.

[34] SBERT Team. *Sentence Transformers Documentation*. Accessed November 27, 2024. 2023. URL: https://www.sbert.net/.

[35] TAUS Blog Team. *What Are Sentence Embeddings and Their Applications?* Accessed November 27, 2024. 2023. URL: https://www.taus.net/resources/blog/what-are-sentence-embeddings-and-their-applications.

[36] Zilliz Blog Team. "Similarity Metrics for Vector Search." In: *Zilliz Blog* (2024). Accessed on November 27, 2024. URL: https://zilliz.com/blog/similarity-metrics-for-vector-search.

[37] Ravi Theja. *Evaluating the Ideal Chunk Size for a RAG System using LlamaIndex*. Accessed October 5, 2023. 2023. URL: https://www.llamaindex.ai/blog/evaluating-the-ideal-chunk-size-for-a-rag-system-using-llamaindex-6207e5d3fec5?utm_source=chatgpt.com.

[38] Ravi Theja. *Evaluating the ideal chunk size for a rag system*. Accessed February 20, 2024. 2024. URL: https://vectorize.io/evaluating-the-ideal-chunk-size-for-a-rag-system/#:~:text=Additionally%2C%20the%20chunk%20size%20impacts%20the%20efficiency%20of,sizes%20may%20result%20in%20more%20frequent%20retrieval%20queries..

[39] Wenhui Wang et al. "Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers." In: *Advances in Neural Information Processing Systems* 33 (2020).

[40] Hao Yu et al. "Evaluation of Retrieval-Augmented Generation: A Survey." In: *arXiv preprint arXiv:2301.07297* (2023). URL: https://arxiv.org/abs/2301.07297.

[41] Hao Yu et al. "Evaluation of Retrieval-Augmented Generation: A Survey." In: *arXiv preprint arXiv:2405.07437* (2024).

[42] Tianyi Zhang et al. "Bertscore: Evaluating text generation with bert." In: *arXiv preprint arXiv:1904.09675* (2019).

[43] Tong Zhang, Fred Damerau, and David Johnson. "Text chunking based on a generalization of Winnow." In: *Journal of Machine Learning Research* 2.4 (2002).

[44] Zijie Zhong et al. "Mix-of-Granularity: Optimize the Chunking Granularity for Retrieval-Augmented Generation." In: *ArXiv preprint* (2024). URL: https://arxiv.org/abs/2406.00456.