Norwegian School of Economics
Bergen, Fall 2024

Chunk Smarter, Retrieve Better: Enhancing LLMs in Finance

An Empirical Comparison of Chunking Techniques in Retrieval Augmented Generation for Financial Reports

Joakim Sælemyr and Halvor Thorstensen Femdal

Supervisor: Maximilian Rohrer

Master thesis, Economics and Business Administration
Major: Business Analytics

NORWEGIAN SCHOOL OF ECONOMICS

This thesis was written as a part of the Master of Science in Economics and Business Administration at NHH. Please note that neither the institution nor the examiners are responsible – through the approval of this thesis – for the theories and methods used, or results and conclusions drawn in this work.

Acknowledgments

We express our gratitude to our supervisor, Maximilian Rohrer, for his invaluable advice, guidance and support throughout the writing of our master's thesis. We extend our gratitude to Arctic Securities for their collaborative efforts and valuable guidance in identifying the key areas where the value of our work lies.

Norwegian School of Economics
Bergen, December 2024

Joakim Sælemyr

Halvor Thorstensen Femdal

Abstract

This thesis investigates how Retrieval-Augmented Generation (RAG) improves the ability of Large Language Models (LLMs) to filter information from financial documents. For this task, we first develop *NorwegianFinanceQA*, a dataset containing 433 queries from the financial reports of 9 Norwegian companies, divided into text- and table-related queries. Next, we evaluate the retrieval accuracy and efficiency of RAG systems with different chunking techniques: character-based, recursive, and semantic splitting. Additionally, we propose a table-specific summarization approach. Our results suggest that table summaries achieve perfect accuracy for table queries while at the same time increasing efficiency. However, this improvement comes at the expense of text-query performance. Our findings highlight the importance of tailored chunking strategies when using LLMs and RAG systems for information retrieval in a financial context.

Keywords: Large Language Models (LLMs), Retrieval-Augmented Generation (RAG), Chunking Strategies, Financial Reports

Contents

List of Figures

List of Tables

1 Introduction

Large Language Models (LLMs), such as GPT, have gained widespread attention for their remarkable ability to understand and generate human-like text (Kasneci et al., 2023). This capability has sparked interest in the finance sector, where professionals process large volumes of information to make decisions (Xu, 2024). A promising approach is using LLMs to filter information from financial documents. To be specific, instead of reading a full annual report, an LLM could extract the most relevant information contained in the annual report for a specific question.

Despite their potential, LLMs face significant limitations that make them difficult to adopt in financial research. A primary concern is that LLMs are trained on static data, meaning they cannot access recent or up-to-date information (Fleischer et al., 2024). Financial research often relies on timely documents, and without access to these, LLMs may fail to extract the relevant information to answer the question. Furthermore, even when the necessary information is available, LLMs can struggle to pick out the most relevant details. Trained on vast datasets, they may have difficulty focusing on the specific information needed. This challenge is further compounded by the unique multi-modal structure of financial documents, which differs from standard text and can confuse the information filtering process. As a result, LLMs sometimes generate responses that appear coherent but are factually incorrect or misleading - a phenomenon known as hallucination (Zhang et al., 2023).

Interviews with financial analysts confirm that these limitations create significant barriers to adopting LLM tools in their workflows. In particular, analysts express concerns about the *accuracy* of LLM-generated answers, stressing the need for highly precise responses - a standard they feel LLMs often fail to meet. They also struggle to *trust* LLM tools that cannot pinpoint the original data sources for their answers, as this makes it difficult to verify whether the information is accurate.

To address these challenges, Retrieval-Augmented Generation (RAG) has emerged as a promising approach. RAG enhances traditional LLMs by retrieving relevant pieces of information from an external knowledge base before generating responses (Gao et al., 2024; Yepes et al., 2024). By integrating real-time, accurate data directly to the LLM, RAG reduces the risk of hallucination and enables more reliable outputs (Yu et al., 2024). Additionally, RAG systems can show the specific sections of documents used to generate a response. For instance, when an analyst queries a quarterly report, they not only receive an answer from the LLM but also see the relevant sections from the report that support it. This transparency builds trust by allowing users to validate the accuracy of the response themselves.

A critical factor in the performance of RAG systems is *how* external documents are divided into smaller, manageable pieces - referred to as **chunks**. The choice of chunking technique significantly impacts the accuracy of information retrieval from these documents (Yepes et al., 2024). In this paper, we evaluate various chunking methods to determine which approaches optimize retrieval performance from financial reports. Our goal is to develop best practices for using RAG with financial documents. By addressing the challenges of accuracy and trust, we aim to make LLM-based tools more reliable and useful for financial professionals.

To achieve this goal, we first develop *NorwegianFinanceQA*, a dataset designed to evaluate RAG systems in Norwegian financial research. The dataset is built from annual and quarterly reports of *9* Norwegian companies listed on Oslo Børs, covering a variety of document sizes and structures. It includes *433* queries, each paired with the necessary keywords needed to answer the query, extracted directly from the reports. For example, one query asks, "What was the operational income of Fish Farming Central Norway for Salmar for Q2 2024?" with the necessary keywords: "['Fish Farming Central Norway', 'Q2 2024', 'Operating income', 'NOK million', '2,656']". *NorwegianFinanceQA* is unique in its focus on Norwegian financial documents, distinguishing it from international datasets like FinanceBench (Islam et al., 2023).[1] It also considers both text-based and table-based queries, enabling evaluations of how RAG systems handle different content types. The dataset is publicly available on GitHub to support further research in financial document RAG systems.[2]

Next, we use three chunking techniques to divide the *9* financial reports from *NorwegianFinanceQA* into smaller pieces (chunks): **character-based splitting**, which splits text into fixed-length segments; **recursive splitting**, which divides based on document structures like paragraphs and line breaks; and **semantic splitting**, which uses natural language processing (NLP) techniques to divide text when sentences are semantically different. In addition, we propose a strategy tailored to the multi-modal nature of financial reports, chunking tables differently than text. When the algorithm identifies a table, it incorporates a portion of the preceding text to provide additional context. An LLM then generates a concise summary that integrates information from both the table and its surrounding text. These table summaries are treated as distinct chunks, while the remaining text is processed using the aforementioned chunking methods.

[1] FinanceBench encompasses a comprehensive dataset of 10,238 questions focused on publicly traded companies, primarily derived from American financial documents such as SEC filings (10-Ks, 10-Qs) and earnings call transcripts. Notably, the dataset underscores the current challenges of applying LLMs to financial documents. For instance, GPT-4-Turbo struggled to answer 81% of the questions accurately. Moreover, the authors of FinanceBench observed significant hallucination issues across all tested LLM models, concluding that these limitations hinder their use for corporate applications.

[2] The test corpus can be found at: https://github.com/Joakim-Sael/NorwegianFinanceQA

We evaluate the performance of the different chunking strategies in retrieving the correct information based on queries from the *NorwegianFinanceQA* dataset. For each query, we retrieve the most relevant sections (chunks) found by each chunking strategy. **Accuracy** is measured by determining whether at least one of these chunks contains all the necessary keywords required to answer the query correctly. To address the need for validation, we analyze the **efficiency** for each strategy by calculating the number of tokens that an analyst would need to review before encountering all the necessary keywords.

Our analysis shows that among the three default strategies, recursive chunking with the largest tested split (4,000 characters and a 1,000-character overlap) delivers the highest accuracy, achieving 94% across all queries in the corpus. However, when it comes to efficiency, the best-performing approach combines recursive text chunking with table summaries, retrieving the relevant keywords within an average of 2,423 tokens ($\approx$ 3.7 pages).[3] This method retains a high accuracy, achieving 90% accuracy, which we consider more balanced compared to the former method's inefficiency, requiring an average of 174.37% more tokens (6,648 tokens $\approx$ 10 pages) to identify all necessary keywords.

Building on current RAG research, we find that combining table-summarization for tables with default chunking for text greatly improves the retrieval performance on table-related queries in financial reports. On average, this approach achieves perfect accuracy (100%) on table queries, compared to 80% when using the conventional chunking strategies. Additionally, the table-summarization approach only requires 546 tokens ($\approx$ 0.82 pages) on average to capture all relevant keywords - significantly fewer compared to the default approach's 13,795 tokens ($\approx$ 20.8 pages). However, we discover that the improved retrieval performance on tables comes at a cost for text-based queries. The table-summarization approach reduces accuracy for text queries, dropping from 86% to 70%. Furthermore, token usage rises to 11,666 ($\approx$ 17.6 pages) compared to 10,789 tokens ($\approx$ 16.3 pages) for the default strategies. Thereby, our main contribution to the existing literature is demonstrating that treating tables and text differently has a substantial impact on information retrieval for financial documents, underlining its importance in future chunk strategy developments.

The rest of the thesis is structured as follows: **Section 2** provides a review of the relevant theory and literature, including an overview of RAG, its applications in finance, and the hypotheses guiding this study. **Section 3** describes the methodology, including the creation of the *NorwegianFinanceQA* dataset, the chunking strategies applied, and the approach used for evaluating retrieval performance. **Section 4** presents the results and analysis, focusing on the accuracy and efficiency of various chunking strategies.

[3]The number of pages was calculated by dividing the total tokens by the average tokens per page ($\approx$ 663) in the test corpus (see Table 3.1).

2 Theory and Literature Review

This section begins with an overview of a typical RAG pipeline, focusing on its core processes: indexing, retrieval, and generation. Next, we explore two fundamental components of RAG: embeddings and similarity search. These components draw attention to the critical role of chunking in improving retrieval performance. We also review relevant literature on chunking techniques in RAG, with special attention to their application in financial contexts. Finally, we use these insights to develop hypotheses that form the foundation for our research.

2.1 RAG overview

The concept of RAG was first introduced in 2021 with the paper *Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks* by Lewis et al. (2021). Since then, RAG has gained significant attention, leading to hundreds of studies and reviews, largely due to its ability to enhance LLM capabilities (Bansal & Suddala, 2024; H. Li et al., 2022; Woo et al., 2024). A simplified overview of the RAG process, inspired by Gao et al. (2024) and S. Wu et al. (2024), is shown in Figure 2.1.

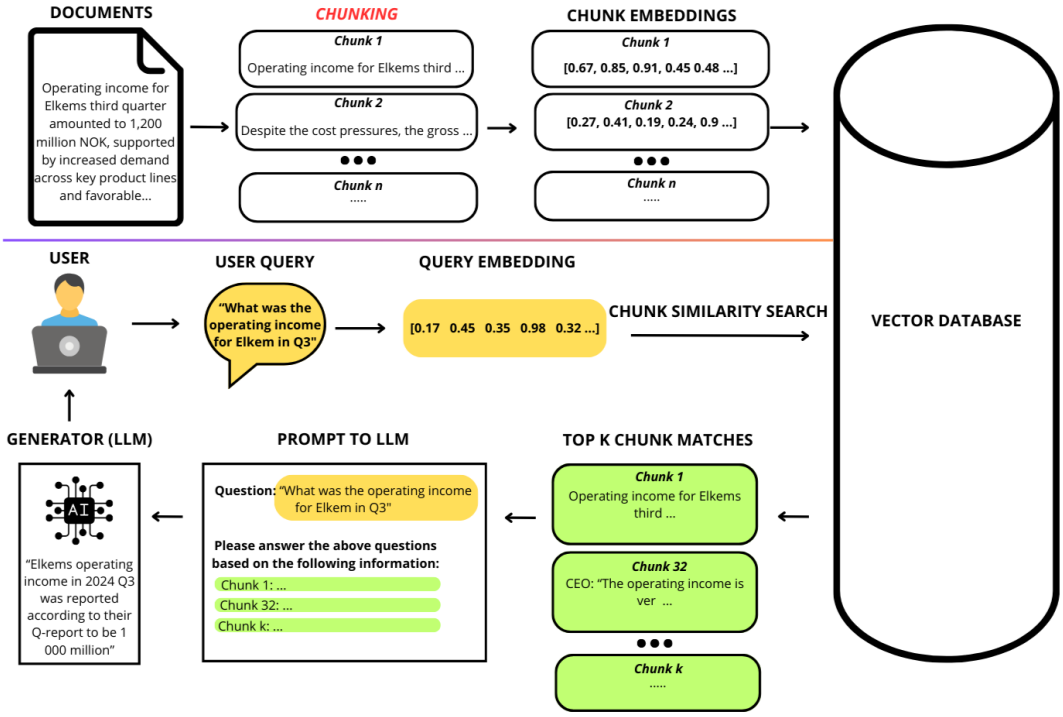


Figure 2.1: Overview of the Retrieval-Augmented Generation (RAG) pipeline

The RAG process can be divided into three main steps:

Indexing: The first step is to create a knowledge base containing all the relevant infor-

mation the system might need. This could include financial reports, market data, news articles, and regulatory documents. The knowledge base serves as the external source of information that the model can draw upon to generate accurate responses (Guu et al., 2020). Before the knowledge base can be used effectively, it needs to be broken down into smaller, manageable pieces called "chunks". Each chunk is then converted into an embedding - a numerical representation that captures the meaning of the chunk (Balažević et al., 2019).

***R**etrieval*: When a user inputs a query, the system embeds this query using the same embedding model used for the chunks. This ensures that both the query and the chunks are represented in the same vector space, making it possible to compare them directly. The RAG system calculates the similarity between the embedded user query and all the embedded chunks in the knowledge base, and retrieves the top k chunks that are most similar to the query. These chunks are considered the most relevant pieces of external information related to the user's query.

***A**ugmented and **G**eneration*: The retrieved chunks are combined with the original user query and fed into an LLM. The generative model uses this combined information to produce a response that is informed by the external knowledge and tailored to the user's specific question.

Thus, by grounding the model on an external knowledge base, RAG systems significantly reduce the risk of hallucination - a common issue where generative models produce inaccurate or unsupported responses. Additionally, when equipped with recent and domain-specific information, the LLM can provide more accurate and contextually relevant answers, even for emerging or specialized topics.

A key advantage of RAG systems is their cost efficiency, particularly in domain-specific applications. Training large generative AI models with new data is prohibitively expensive and resource-intensive due to their size and complexity (Zhao et al., 2023). Moreover, domain-specific GPT models have been found to perform worse on datasets within their intended domain compared to general-purpose models. For instance, Bloomberg GPT underperformed GPT-4 on financial tasks (X. Li et al., 2023). Fine-tuning existing pretrained GPT models with additional domain-specific knowledge has been proposed as a solution, but it does not address the issue of hallucinations (Donavan, 2024) and, in some cases, has been shown to exacerbate the problem (Gekhman et al., 2024). RAG, in contrast, enables updates to the knowledge base without requiring extensive model retraining or fine-tuning. This makes RAG particularly useful in dynamic domains where information changes frequently, such as finance.

However, implementing RAG systems involves several challenges due to variability in each

step of the pipeline. Research highlights that decisions around chunk strategies(Smith & Troynikov, 2024), selecting embedding models (Caspari et al., 2024), determining the number of retrieved chunks (Anthropic, 2024), and choosing appropriate generative models (Databricks, 2024) all substantially affect performance. Among these steps, chunking has been identified as especially critical to the success of RAG systems (Yepes et al., 2024).

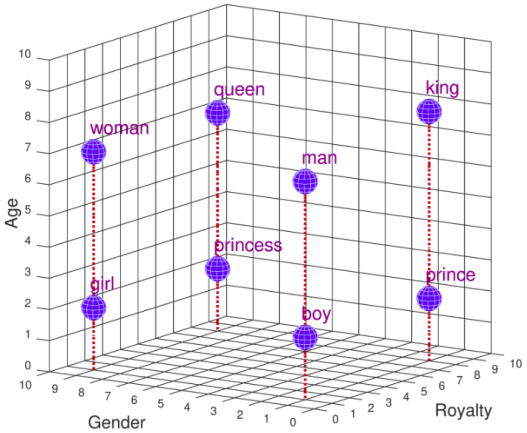
For this thesis, we adopt standard approaches for most steps in the RAG pipeline and focus our efforts on experimenting with and evaluating different chunk strategies. As shown in Figure 2.1, the chunking process is highlighted in red to indicate its importance in our research.

2.1.1 Embeddings and Similarity Search

To understand why chunking is important in RAG systems, we explore the concepts of embeddings and similarity search.

Embeddings are a way to represent text - words, sentences, or entire documents - as numbers. These numbers (or vectors) capture the meaning of the text in a way that models can use. Figure 2.2 illustrates a simplified example of a vector space, where words such as "boy" and "girl" are positioned closer together than "boy" and "queen".

Figure 2.2: Word embeddings in a vector space



Word Coordinates

	Gender	Age	Royalty
man	1,	7,	1
woman	9,	7,	1
boy	1,	2,	1
girl	9,	2,	1
king	1,	8,	8
queen	9,	7,	8
prince	1,	2,	8
princess	9,	2,	8

The figure is sourced from: (CMU School of Computer Science, n.d.)

In practice, state-of-the-art embeddings are much more complex. They exist in multi-dimensional spaces created by advanced models trained on large amounts of text.[4] Early methods like Word2Vec (Mikolov et al., 2013) created static embeddings by analyzing

[4]HuggingFace provides a leaderboard ranking various embedding models. See https://huggingface.co/spaces/mteb/leaderboard. At the time of writing, embedding models range from 64 dimensions to as many as 30,522(!).

word co-occurrences, capturing only the isolated meaning of words. However, the introduction of the transformer architecture in the seminal *Attention Is All You Need* paper (Vaswani et al., 2023) revolutionized the field. The self-attention mechanism it introduced allows models to understand the relationships between words in a sentence, capturing contextual meaning with unprecedented precision.

This technology forms the foundation of modern Natural Language Processing (NLP) models like BERT (Devlin et al., 2019) and GPT (Brown et al., 2020), enabling them to excel at various NLP tasks by generating embeddings that reflect the context in which words appear. The transformer's ability to process and integrate contextual information is a key reason these models outperform earlier approaches and have become cornerstones of NLP and LLM advancements.

Once embeddings are created, **similarity search** is used to measure how closely related two vectors are. A common method for this is *cosine similarity*, which calculates the cosine of the angle between two vectors. The formula for cosine similarity between two vectors, $\mathbf{A}$ and $\mathbf{B}$, is:

$$\cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2} \sqrt{\sum_{i=1}^{n} B_i^2}}$$

In this equation, the numerator calculates the dot product of the two vectors, which reflects how much they align, while the denominator normalizes this value by the magnitudes of the vectors. The cosine similarity value ranges from -1 to 1, corresponding to the angle (θ) between the vectors. When the angle is 0°, the vectors are perfectly aligned, and the cosine similarity equals 1, indicating maximum similarity. An angle of 90° results in a cosine similarity of 0, meaning no similarity, while an angle of 180° gives a cosine similarity of -1, signifying complete dissimilarity. As the angle decreases, the cosine similarity increases, reflecting greater alignment between the vectors.

Figure 2.3 illustrates the concept of cosine similarity using three example sentences. The angle between the vectors representing the sentences `'Did the earnings rise for the previous quarter?'` and `'The profit last quarter increased.'` is smaller than the angle between `'Did the earnings rise for the previous quarter?'` and `'The bananas were not edible.'`. This indicates greater semantic similarity between the first pair of sentences, since they relate to the same financial concept, compared to the unrelated context of the latter example.

In a RAG system, as described in Section 2.1, Figure 2.4 demonstrates how cosine similarities are distributed between the embedding of an input query and the embeddings

Figure 2.3: Visualization of sentence embeddings and their cosine angles

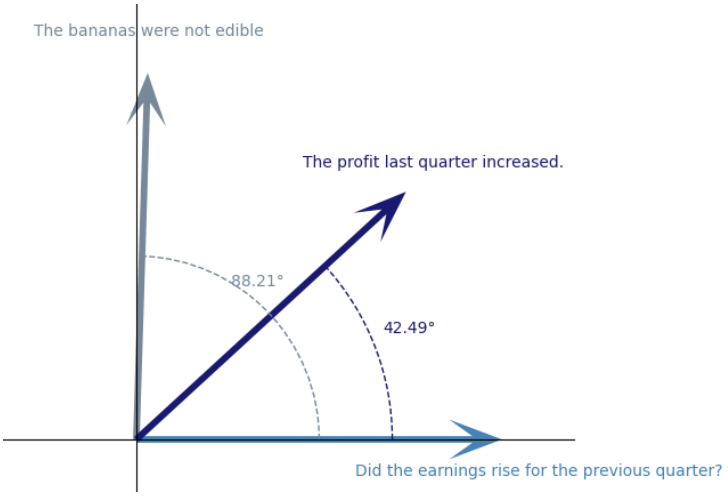


Figure 2.3 show embeddings for the three sentences. They were generated using the model 'all-MiniLM-L6-v2,' and cosine similarities were calculated: 0.7374 for the semantically related sentences and 0.0391 for the unrelated pair. These values were transformed into angles using the arccosine function. The visualization focuses solely on angles for illustrative purposes and does not account for vector magnitudes.

of various chunks from a financial report. In this example, chunks 125 and 126 appear to be the most similar to the user query. These chunks, along with others ranked by relevance, are prioritized and included with the query when passed to the LLM for answer generation.

Figure 2.4: Similarity scores between query and document chunks

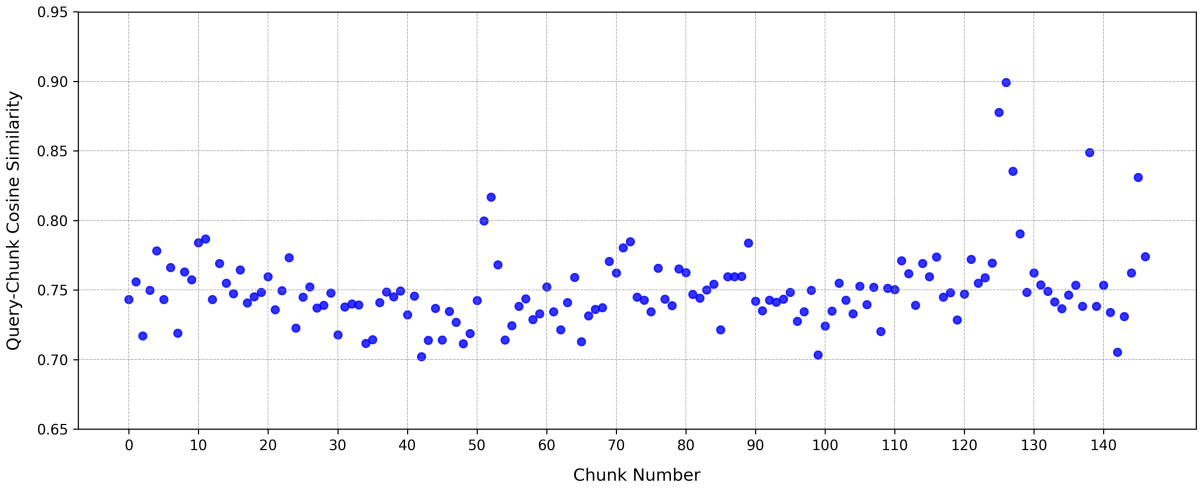


Figure 2.4 displays the cosine similarities between a query and document chunks for Salmar's Q2 2024 financial report. The phrase (query) used was: "Resource rent tax - implementation effect (deferred tax)"—a very specific query to clearly display differences. Furthermore, the document was chunked using a naive approach (this will be covered in Section 3)

Understanding embeddings and similarity search explains why chunking is so important. The way documents are chunked directly affects the quality of embeddings, which are crucial for similarity search. Poorly designed chunks that combine unrelated topics can produce embeddings that fail to represent the meaning of sections in the doc-

ument, making it harder for the system to identify relevant information during similarity calculations (Schwaber-Cohen, 2023). In contrast, carefully constructed chunks that preserve contextual integrity lead to clearer embeddings. This enhances the similarity search process, ensuring that user queries are matched with the most relevant document chunks, ultimately improving the quality of retrieval in RAG systems and consequently LLM outputs.

2.1.2 Chunk Retrieval

Although chunking's importance is evident, the existing literature on the subject remains relatively sparse. Nonetheless, studies such as Yepes et al. (2024) and Zhong et al. (2024) highlight its significant impact on RAG performance. These works empirically demonstrate that the chunking method used influences the quality of embeddings, which, in turn, determines how effectively RAG systems retrieve relevant information and generate accurate responses. By emphasizing the role of chunking, these findings underline the need for continued research and optimization in this area.

Studies by Antematter (2023), Theja (2023), and Setty et al. (2024) demonstrate that the size of the chunks is a critical factor in determining the performance of RAG systems. Larger chunks are better at capturing the overall context and relationships between sentences and paragraphs, while smaller chunks are more effective at understanding specific details within individual sentences (Schwaber-Cohen, 2023). Similarly, Günther et al. (2024) highlights that vector-based retrieval systems often perform better with shorter text segments due to their more precise semantic representations, but also warns that this approach can result in a loss of contextual information from surrounding content, leading to suboptimal results. Consequently, the literature identifies a trade-off between maintaining broader context and achieving greater accuracy for smaller details, which must be carefully balanced.

Furthermore, Setty et al. (2024) highlights that - in addition to chunk size - the structure of documents significantly influences RAG performance. Naive chunking strategies may fail to produce coherent chunks that preserve the intended meaning of the content. This issue is particularly pronounced in documents that include diverse elements such as tables, graphs and headers. To address these challenges, advanced chunking techniques have been developed that consider structural elements like paragraphs and tables. These methods show promise, especially when applied to knowledge bases composed of unstructured documents (Yepes et al., 2024). In summary, the ideal chunking strategy depends on the characteristics of the knowledge base and the balance required between context preservation and accuracy for answering user queries.

2.2 RAG in finance

While RAG has been extensively studied in fields such as medicine (J. Wu et al., 2024; Xiong et al., 2024), law (Pipitone & Alami, 2024; Wiratunga et al., 2024), and education (Han et al., 2024; Levonian et al., 2023) there is comparatively little research focused on its application in the finance sector.[5] Notable contributions include studies by Setty et al. (2024) and Yepes et al. (2024), which explore methods to improve text retrieval in RAG systems for financial documents. Yepes et al. (2024) focuses on various chunking techniques, while Setty et al. (2024) examines additional components of the RAG process, such as query expansion, re-ranking, and fine-tuning embedding models. Both studies demonstrate improvements, indicating significant potential for future research in this area.

The growing adoption of LLMs by large international investment banks underscores the practical value of these technologies in the finance industry. For instance, Morgan Stanley has integrated *AskResearchGPT*, an internal GPT-4-based chatbot that gives employees the ability to query the over 70,000 research reports the large bank produces annually (Son, 2024b). Similarly, JP Morgan Chase has deployed *LLM Suite*, an in-house virtual research assistant used by over 50,000 employees (Son, 2024a). Although the specifics of these systems remain proprietary, one can only speculate that they are designed with RAG capabilities to retrieve information accurately and efficiently from their internal knowledge bases.

The lack of publicly available research in this field creates an opportunity. By publishing research on RAG systems for finance, others in the industry can benefit and build on these findings. Additionally, examples from leading international banks demonstrate that the integration of LLMs are both practical and promising, suggesting similar implementations could be successful in Norway.

2.3 Hypothesis Development

Drawing from insights gained through interviews with financial analysts, technical knowledge of RAG systems, and existing literature on chunking strategies, this section outlines our hypotheses regarding the relationship between chunking methods and retrieval performance in financial contexts.

Financial analysts emphasized that overcoming the adoption barriers for LLMs requires

[5]This observation is supported by a Google Scholar search using the query "Retrieval Augmented Generation + domain," which yielded 175,000 results for medicine, 212,000 for law, 290,000 for education, and only 72,000 for finance. Additionally, this aligns with our own impressions based on thorough review of RAG literature.

highly accurate answers that can be easily validated. As discussed in Section 2.1, RAG systems address this challenge by selecting the k most relevant chunks of information from a knowledge base and passing them to the LLM for response generation.

Although analysts refer to high accuracy as the correctness of the LLM outputs, we address this indirectly by focusing on the accuracy of the retrieval system. Specifically, an accurate RAG system should ensure that at least one of the top k retrieved chunks contains all the relevant information needed to answer the query. Figure 2.5 illustrates an example where the system retrieves the 20 chunks it deemed the most relevant to the user query. The displayed chunk contains the information needed to answer the query, demonstrating that the retrieval system accurately identified the correct chunk within the top 20 results.

Figure 2.5: RAG-based system interface for financial documents

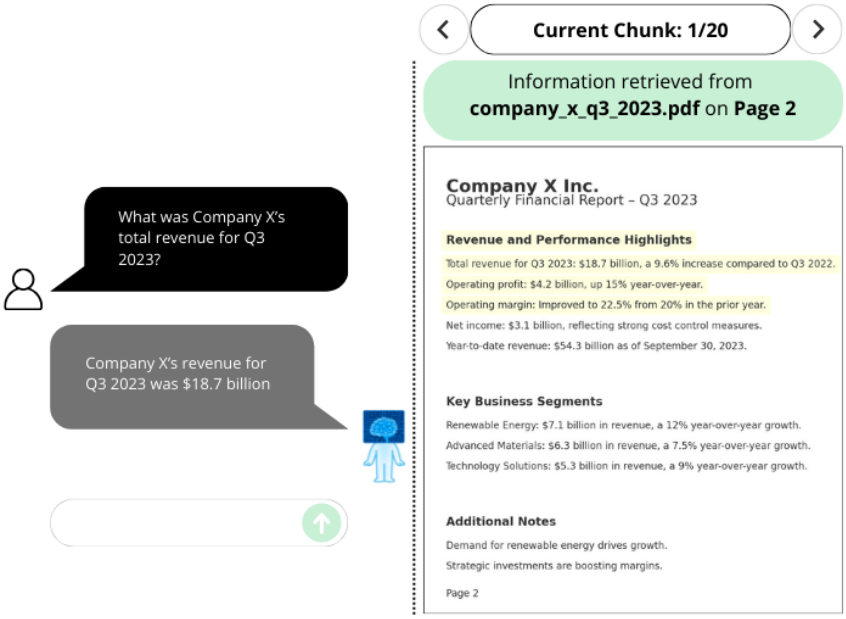


Figure 2.5 showcases a potential user interface for a RAG-based system applied to financial documents. On the left, the interface resembles a standard LLM front-end. However, the RAG system extends its functionality, as shown on the right. This extension displays the top 20 most relevant chunks retrieved from the database, with the currently viewed chunk highlighted in yellow. Additionally, the interface presents the corresponding page of the source document for enhanced context.

Building on existing literature that demonstrates RAG systems' ability to improve the accuracy of AI-generated outputs, we argue that these findings are directly applicable to retrieval accuracy. Section 2.1 highlighted that existing literature on RAG demonstrates improved AI-generated outputs, particularly for domain-specific tasks. While much of this research has focused on non-financial contexts, recent findings, as discussed in Section 2.2, indicate promising applications for financial documents. Additionally, Section 2.1.1 emphasized the critical role of effective chunking in enhancing LLM performance, supported by studies in Section 2.1.2. Consequently, we expect similar impacts in retrieval accuracy when testing various chunking strategies in our research.

To make it *easy* for analysts to validate answers generated by an LLM, the relevant information should appear early in the sequence of retrieved chunks (low k) and be presented in concise, manageable chunks. For instance, Figure 2.5 illustrates an ideal scenario where the correct chunk is identified as the most relevant ($k = 1$) by the RAG system. In addition, the chunk size is small, enabling an even easier validation.

In contrast, a worst-case scenario occurs when the RAG system either fails to identify the correct chunk or retrieves it too late in the sequence. Imagine an analyst reviewing 20 large chunks of information without finding the chunk the LLM answer was based on, ultimately giving up due to the inefficiency of the process. Thus, if the chunk with the necessary information is buried too far down the list (large k) or its hidden within large, unwieldy chunks, the validation process could become as labor-intensive and time-consuming as manually reviewing the documents. As a result, a well-designed RAG system for financial research must prioritize accuracy *and* ensure that relevant information is retrieved early in the process in a manageable way without chunks becoming overly large.

These insights form the basis for our main hypothesis:

H1: *Chunking strategies in RAG systems influence the accuracy and efficiency of information extracted from financial reports.*

The literature discussed in Section 2.1.2 indicated that the effectiveness of chunking strategies depends on the nature of the documents in the knowledge base. Financial documents often feature unique structures with numerous headers, paragraphs, and tabular data. Thus, we further hypothesize:

- **H1-a**: *Chunking strategies that account for tabular data and text structures (element-based methods) perform better than naive chunking strategies in ensuring accurate and efficient retrieval of information from financial documents.*

Additionally, the literature in Section 2.1.2 highlights a trade-off between accuracy and context preservation when determining chunk sizes. Financial analysts typically require a high degree of precision, as they often rely on specific numbers and information to inform their decisions. In contrast, broader questions, such as requests for summaries, may place greater emphasis on context rather than precision. Based on this understanding, we hypothesize:

- **H1-b**: *Shorter chunk sizes perform better than longer chunk sizes in ensuring accurate and effective retrieval of information from financial documents.*

3 Methodology

This section outlines the methods used to evaluate the accuracy and efficiency of various chunking strategies within RAG systems for financial documents. The chapter begins by describing the process of developing the *NorwegianFinanceQA* dataset, including data collection and generation. This is followed by a detailed explanation of three widely used chunking strategies: character-based, recursive, and semantic splitting. To address the unique challenges posed by multi-modal content, a specialized chunking strategy for processing tables separately is proposed. Finally, two key metrics are introduced to evaluate the accuracy and efficiency of retrieval for each chunking strategy: Full Keyword Coverage and Tokens To Answer.

3.1 Test Corpus

We create *NorwegianFinanceQA*[6], a dataset that consists of 433 carefully generated questions from the financial reports of 9 different companies. *NorwegianFinanceQA* is, to the best of our knowledge, the first open-source RAG and LLM evaluation system based on Norwegian financial documents. It is also the first (open-source) dataset to divide questions into whether they need information from tables or the main text. This enables creators of RAG and LLM systems to assess performance across two different modalities.

3.1.1 Background

By reviewing the literature, we found that most studies on RAG in finance, including the two studies we identified as most relevant for our research discussed in Section 2.2, uses FinanceBench as their test data. FinanceBench is a Q&A dataset based on U.S. financial reports like 10-Ks and 10-Qs, which to a large extent follows a standardized structure. Norwegian financial documents, however, are not bound by strict structural regulations, resulting in significant variability in their formats and structures. Since we focus on RAG systems for Norwegian reports, we found that documents based on international datasets were too uniform. Thus, we saw the need to have a curated and well-defined dataset for Norwegian documents, thereby creating *NorwegianFinanceQA*.

[6] *NorwegianFinanceQA* is available at: https://github.com/Joakim-Sael/NorwegianFinanceQA. All results in this thesis were obtained using the data of the initial commit. Changes may occur if we notice errors

3.1.2 Dataset Collection

We collect data from the investor relations sections of companies listed on the Oslo Stock Exchange (Oslo Børs). We find this evaluation suitable since the documents represent the unstructured nature of material evaluated by analysts. One could argue that analysts also heavily rely on in-house calculations and reports, which might make them a better fit for creating a test corpus. However, as established in Section 2.2, public research holds significant importance, and since in-house analyst reports are often proprietary information, open-sourcing the data would not be possible. Therefore, publicly available annual and quarterly reports from Norwegian companies were collected.

We access and download financial reports from 9 companies across various industries. This was done as a measure to avoid bias in the event that certain industries follow similar structures. We included both annual and quarterly reports of different sizes and structures in the corpus. Table 3.1 provides a summary of words, tokens[7], tables and pages for each document in the dataset.

Table 3.1: *NorwegianFinanceQA* document statistics

File	Words	Tokens	Tables	Pages
Aker Q2 2024	14,317	20,958	32	28
Bouvet 2023 Annual Report	52,152	68,926	118	113
Equinor Q2 2024	22,833	35,362	61	52
Kongsberg Q2 2024	10,394	16,599	40	33
Mowi Q2 2024	19,120	29,365	50	36
Nortura 2023 Annual Report	1,744	3,021	5	8
Norwegian Q3 2024	10,727	15,751	15	26
Salmar Q2 2024	13,078	19,439	31	23
Statkraft Q3 2024	18,140	27,934	43	39
Totals	**162,505**	**237,347**	**395**	**358**

3.1.3 Dataset Generation

To create *NorwegianFinanceQA* we utilized LLMs ability to produce queries based on a given snippet of text. The LLM paired each query with correct answers, referred to as *ground truths*, a standard metric in RAG evaluation (Chatrath et al., 2024). While we do not use ground truths in our retrieval analysis, we included them in the test data to

[7]Tokens are a fundamental unit of measure in datasets used for LLMs, because they align with how LLMs process text. Thus, it gives a picture on the size of the dataset in terms of the computational workload it imposes on an LLM, potential costs, and whether it is within the context window (LLMs have a maximum token limit per input) to mention a few reasons. To count tokens we used TikToken for *cl100k_base* which is the standard used by OpenAI's GPT-4 (Smith & Troynikov, 2024).

enhance its usability for public purposes. Additionally, we introduce a variable called *necessary keywords*, which represents all the critical information required to answer a query, taken directly from the given text snippet. To showcase this structure we generate an example query, corresponding ground truth, and necessary keywords, as shown in Figure 3.1.

Figure 3.1: Query, ground truth, and necessary keywords.

Query: "What was the operational income of Fish Farming Central Norway for Salmar for Q2 2024?"
Ground Truth: "The operational income of Fish Farming Central Norway for Salmar in Q2 2024 is NOK 2,656 million."
Necessary Keywords: 'Fish Farming Central Norway', 'Q2 2024', 'Operating income', 'NOK million', '2,656'

The necessary keywords variable is used to evaluate both the accuracy and the efficiency of the retrieval process. Additionally, it accounts for the fact that financial documents often contain the same information in multiple locations. As a result, retrieving the information from a different section than where the question originated is still considered valid, thereby enhancing the system's robustness.

To generate each query, along with the corresponding ground truth and necessary keywords, we processed each document individually, dividing it into shorter segments suitable for question generation. Since a key focus of this thesis is on tables, we employed distinct strategies for the creation of text-based queries and table-based queries.

For text queries, we separated all text elements in each document into smaller text-snippets, all while avoiding any tables within the text snippet that was the basis for question generation. For tables we identified the tables within the document and used the table itself and surrounding text to create the question.

The snippets for both text queries and table queries were sent to an LLM which had a carefully curated prompt. Prompts were matched with a corresponding schema to ensure that the outputs were structured and reliable. This was achieved using the structured outputs feature of the OpenAI API.[8] The prompt and schema creation process for text and tables queries were different as they required different elements to focus on and a slightly different schema for each. The schema differed in the fact that for text-based questions, we included a category to distinguish between text reasoning and number reasoning. This allowed us to identify whether the answer was contained as a number or plain text. For text queries, we applied few-shot prompting, providing two examples - one

[8] A guide on structured outputs can be found at: https://platform.openai.com/docs/guides/structured-outputs

for text reasoning and one for number reasoning. In contrast, for table queries, we used one-shot prompting, including only a single example. This approach ensured the prompts were tailored to the specific requirements of each query type.

As mentioned, *NorwegianFinanceQA* is to our knowledge the first open-source test data to separate queries by modalities enabling developers of RAG applications to accurately pinpoint where their RAG systems struggle. A breakdown of query-types and their respective counts are presented in Table 3.3.

Table 3.3: Breakdown of Query-Types in *NorwegianFinanceQA*

Query-type	**Count**	**Reasoning**	
		Text	**Number**
Table	189		
Text	244	114	130

After we had created the queries we ensured all necessary keywords were present in the corpus. Moreover, since we were strict on adherence to the schema we had to punish creativity in the LLM by setting the hyperparameter *temperature* to 0.[9] Therefore, we had questions that felt too specific using exact word matching from the text snippet itself, which felt too unnatural in terms of how it would be phrased in a more natural Q&A setting. Consequently, we created a curation prompt that targeted the query to make it feel more natural and avoid using direct word matching.

All questions in the corpus are designed to be specific, ensuring each query can be assessed to have a definitive correct or incorrect answer. For instance, Table 3.4 illustrates examples of specific versus general questions, along with their status indicators, query scope (broad/specific), and complexity (multi-hop/single-hop). Broad questions like "Summarize Salmar's operational performance in Q2 2024" were excluded, as they require summarization rather than specific retrieval. Additionally, we excluded multi-hop queries requiring information from multiple documents, such as: "Which company, Salmar or Mowi, reported a higher EBIT in Q2 2024?". These multi-hop queries refer to compounded questions that require retrieving and synthesizing answers from two or more independent sub-queries. For our evaluation, we prioritized single-hop queries to maintain a straightforward testing approach.

[9]The temperature hyperparameter in the OpenAI API adjust creativity by making outputs more random (temperature = 1) or deterministic (temperature = 0). We refer to the API documentation for interested readers: https://platform.openai.com/docs/api-reference/making-requests

Table 3.4: Query Specifications

Query	Broad/Specific	Multi-hop/Single-hop	Status
What was Salmar's harvest volume in Q2 2024?	Specific	Single-hop	✓
What was Equinor's adjusted operating income in the first quarter of 2024?	Specific	Single-hop	✓
Summarize Salmar's operational performance in Q2 2024.	Broad	Single-hop	✗
Provide an overview of Mowi's earnings in the second quarter of 2024.	Broad	Single-hop	✗
Which company, Salmar or Mowi, had a higher EBIT in Q2 2024?	Specific	Multi-hop	✗
Which company, Salmar or Mowi, had a better Q2 2024?	Broad	Multi-hop	✗

Every query, corresponding ground truth, and necessary keywords were created using `GPT-4o` with the OpenAI API.[10]. The test data *NorwegianFinanceQA*, including all prompts and schemas, is available on GitHub:

`https://github.com/Joakim-Sael/NorwegianFinanceQA`

3.2 Chunking Strategies for Text

3.2.1 Fixed size strategy

The fixed-size chunking method, often referred to as naive chunking, is the simplest and most commonly used (Setty et al., 2024). It involves dividing the text into uniformly sized chunks based on a specified number of characters, with optional overlapping between chunks to preserve context (Qu et al., 2024).

More specifically, given a text T of character length n, a desired **chunk size** s, and an **overlap size** o between consecutive chunks, we aim to create chunks $\{C_i$, where $i = 1, 2 \ldots, k\}$.

The starting index I_i and ending index J_i of the i-th chunk is:

$$I_i = (i - 1) \times (s - o)$$
$$J_i = I_i + s$$

[10] At the time of writing, `GPT-4o` refers to the `gpt-4o-2024-08-06` model.

The chunks will be extracted accordingly:

$$C_i = T[I_i : J_i] \quad \text{for } i = 1, 2, \ldots, k$$

And to determine the maximum number of chunks:

$$k = \left\lceil \frac{n - o}{s - o} \right\rceil$$

While fixed-size chunking is straightforward to implement, it does not account for document structure, such as headers or tables. An example of how a document is broken into smaller pieces, *chunks*, using a fixed-size strategy is shown in Figure 3.2.

Figure 3.2: Fixed-size chunking with 800 chunk size

(a) 0 overlap **(b)** 100 in overlap

Figure 3.2 illustrates Salmar's Q2 2024 report page 3 processed using fixed-size chunking with a chunk size of 800 characters. The left subfigure shows chunking with no overlap, where each chunk is distinct. The right subfigure demonstrates chunking with an overlap of 100 characters, indicated by the darker or mixed highlighting. For detailed calculations on how chunking is performed, we refer to Appendix A.1.

3.2.2 Recursive strategy

The recursive splitting strategy differs from fixed-size splitting by considering the natural structure of the text. Instead of cutting at a fixed length, it uses a sequence of separators, such as paragraph breaks (`\n\n`), line breaks (`\n`), spaces (`" "`), and empty strings (`""`). The method works hierarchically: it first tries to split the text by the largest separator (paragraph break), and if the resulting chunk is still too long, it proceeds to the next separator in the list (Mishra, 2024). This process continues until all chunks meet the chunk size limit. Additionally, the method attempts to merge smaller chunks where possible to avoid unnecessary fragmentation, attempting to create chunk sizes as close to the limit as possible.

The **algorithm** can be divided into three steps:

Step 1: Recursive splitting

The process begins by attempting to split the text T with the first separator σ_1. For each resulting segment S_j, if the length $\ell(S_j)$ is less than or equal to the maximum chunk size s, the segment is directly added as a chunk C_k to a list of chunks C. However, if $\ell(S_j) > s$, the segment is split further using the next separator σ_2. This recursive process continues, moving through the list of separators $\Sigma = [\sigma_1, \sigma_2, \ldots, \sigma_k]$, until all resulting chunks C_k meet the size requirement $\ell(S_j) \leq s$. This step creates an initial list of chunks $C = \{C_1, C_2, \ldots, C_n\}$ that adhere to the size constraint but may include some chunks that are unnecessarily small due to consecutive separators or uneven splits.

Step 2: Merging Chunks

To reduce fragmentation and avoid very small chunks, the algorithm merges adjacent segments where possible. It initializes an empty chunk $P = \emptyset$ and iterates through the list of chunks C_k in C:

$$P = P + C_k \quad \text{if } \ell(P) + \ell(C_k) \leq s.$$

If the combined size $\ell(P) + \ell(C_k)$ exceeds s, the current chunk P is finalized and added to a new list of finalized chunks D_j. The chunk C_k then becomes the starting point of a new P. This step creates an initial list of finalized chunks $D = \{D_1, D_2, \ldots, D_n\}$, and ensures that the finalized chunks are as close as possible to the maximum size s.

Step 3: Adding Overlaps

To further enhance continuity between chunks, the algorithm introduces overlaps. For each chunk D_j (except the first), a portion of the previous chunk, of length o, is appended

to the start of D_j. Formally, the updated chunk becomes:

$$D'_j = D_{j-1}[l(D_{j-1}) - o :] + D_j, \quad \text{if } j > 1$$

where $l(D_{j-1})$ is the length of the previous chunk. This overlap ensures that important contextual information at the boundary of one chunk is preserved in the next, reducing the risk of losing coherence across chunks. The updated chunk D_j is then replaced with D'_j. The final result is a set of chunks $D = \{D_1, D_2, \ldots, D_n\}$ that respect the document's structure, meet the maximum chunk size s, and maintain continuity through overlaps of size o.

Figure 3.3 shows how chunks appear when the recursive chunking strategy is applied on a financial document. Compared to the fixed-size strategy, it avoids splitting text arbitrarily in the middle of a line (see Figure 3.2a) and takes greater care to preserve sections and line breaks, resulting in a more structured division of the text.

Figure 3.3: Recursive chunking with 800 chunk size

(a) 0 overlap **(b)** 100 in overlap

Figure 3.3 illustrates Salmar's Q2 2024 report page 3 processed using recursive chunking with a chunk size of 800 characters. The left subfigure shows chunking with no overlap, where each chunk is distinct. The right subfigure demonstrates chunking with an overlap of 100 characters, indicated by the darker or mixed highlighting. For detailed calculations on how chunking is performed, refer to Appendix A.2.

However, while the recursive strategy considers the structural format of the document, it

often fails to preserve contextual integrity. It may split sections inappropriately, as seen in Figure 3.3, where recursive chunking frequently divides sections mid-sentence due to line breaks. Furthermore, this approach does not account for the semantic meaning of the text, treating all sentence divisions equally without regard to their relationships.

3.2.3 Semantic Strategy

To overcome the limitations of the recursive strategy, a *semantic strategy* has gained significant attention (Qu et al., 2024). Unlike structural approaches, this method focuses on the inherent meaning of sentences, leveraging sentence embeddings to compare semantic relationships and group sentences into coherent chunks. This approach was introduced by Greg Kamradt, and while there exist modifications to his methodology, we will follow his techniques in this chapter.[11]

The semantic strategy builds upon the concept of *retrieving the most similar chunks* (discussed in Section 2.1.1), where the embeddings of groups of sentences are used to assess semantic similarity. For a given sentence d_i, the method considers a window of w sentences, including both prior and subsequent sentences, to capture context:

$$\text{Window}(d_i) = \{d_{i-w}, \ldots, d_i, \ldots, d_{i+w}\}.$$

The most common approach is using $w = 1$, thus looking at the previous and post sentence for a sentence i. The combined embedding for the window is computed as:

$$d_i^{\text{combined}} = \text{Encoder}_d(\text{Concatenate}(d_{i-w}, \ldots, d_i, \ldots, d_{i+w})),$$

where Encoder_d maps the concatenated text into a high-dimensional vector space.

The semantic similarity between two consecutive windows, represented by their combined embeddings d_i^{combined} and $d_{i+1}^{\text{combined}}$, is measured using cosine similarity:

$$s(d_i, d_{i+1}) = \frac{d_i^{\text{combined}} \cdot d_{i+1}^{\text{combined}}}{\|d_i^{\text{combined}}\| \|d_{i+1}^{\text{combined}}\|}.$$

The *cosine distance*, which quantifies dissimilarity, is then given by:

$$\text{Distance}(d_i, d_{i+1}) = 1 - s(d_i, d_{i+1}).$$

[11] For an introduction to this approach, see Greg Kamradt's GitHub repository: https://github.com/FullStackRetrieval-com/RetrievalTutorials/blob/main/tutorials/LevelsOfTextSplitting/5_Levels_Of_Text_Splitting.ipynb

A *breakpoint* is introduced between d_i and d_{i+1} if their semantic distance exceeds a predefined threshold[12] τ, signaling a significant topic change:

$$\text{If Distance}(d_i, d_{i+1}) > \tau, \text{ insert a breakpoint.}$$

An example of how semantic chunking is applied to page 3 of Salmar's Q2 2024 report is illustrated in Figure 3.4. Whenever the cosine distance - representing the difference between two groups of sentences - exceeds the threshold τ, a breakpoint is introduced, and a new chunk begins at that index. The figure demonstrates how this strategy enables variable chunk sizes, determined by the semantic structure of the text, rather than a fixed size. Figure 3.5 further illustrates this point. One key advantage of the semantic chunking approach is its ability to maintain the natural structure of the text. Unlike the recursive strategy (Figure 3.3) and the fixed-size strategy (Figure 3.2), semantic chunking avoids splitting text mid-sentence, resulting in better coherence and preserving the context within each chunk.

Figure 3.4: Cosine distance plot

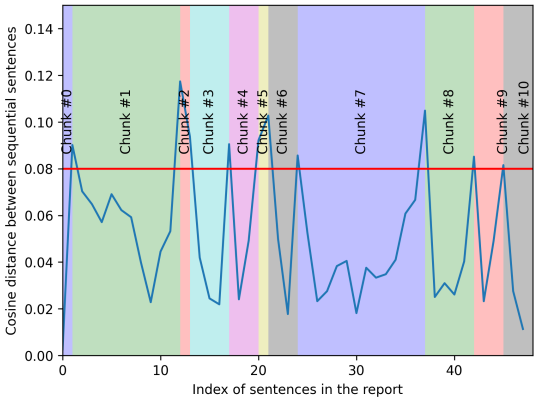


Figure 3.4 illustrates the cosine distance plot for sentences in Salmar's Q2 2024 report, page 3. A window size of $w = 1$ is used, incorporating one sentence before and after each sentence i. The cosine distance threshold is set at 0.08 to identify breakpoints. While this value is chosen for demonstration purposes due to the small sample size, it effectively showcases the concept, as more standard approaches like percentile-based thresholds are impractical in this context.

However, while semantic chunking seems like a promising algorithm to catch the change of topics in the text, it has its disadvantages of being computationally expensive (Qu et al., 2024).

[12]Common thresholds are percentile, standard deviation, and interquartile criteria. In this thesis we use default parameters: percentile and breakpoint threshold = 0.95

Figure 3.5: Semantic Chunking

Figure 3.5 illustrates Salmar's Q2 2024 report page 3 processed using semantic chunking.

3.3 Chunking Strategy for Tables

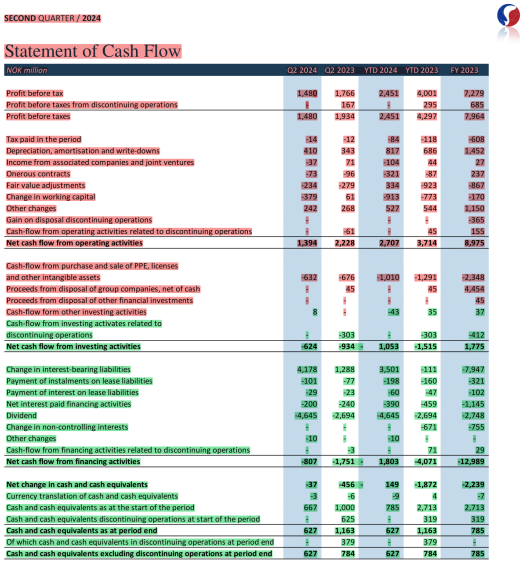
3.3.1 Background

The three chunking techniques presented does not account for the multi-modal nature of financial documents. As discussed in Section 3.1.1, Norwegian financial documents often include a mix of text, tables, graphs, and images. This variety makes it important to develop a strategy that accommodates their multi-modal structure, particularly given the importance of these elements in Norwegian financial documents.

Through a manual review of Norwegian financial documents, we found tables to be the most important multi-modal component to address. Images often act as filler content, and graphs are typically supported by explanatory text referring to the metrics within the graph, which allows the retrieval system to extract relevant information directly from the text. In addition, graphs and images require vision models which are beyond the scope of this thesis. We found that tables frequently contain critical data in isolation without references to the exact numbers and metrics in the text, making them a key source of information. Additionally, Table 3.1 shows that, on average, there is more than one table per page in our corpus, which demonstrates how common and essential tables are in these documents.

However, tables present unique challenges during chunking, particularly when they are large. Unlike text, where sentences usually include enough context on their own, tables rely on the connection between rows and columns to convey meaning. For example, Table 3.6 contains 37 rows. If a query asks for data in the bottom rows, it requires the column headers for proper interpretation. For instance, even when using a relatively large chunk size of 2000 characters, different segments within the table will be split up. This disconnects rows from their headers, making it harder for retrieval systems to extract the correct information in a coherent format.

Figure 3.6: Table chunking with fixed-size strategy



SECOND QUARTER / 2024

Statement of Cash Flow

NOK million	Q2 2024	Q2 2023	YTD 2024	YTD 2023	FY 2023
Profit before tax	1,480	1,766	2,451	4,001	7,279
Profit before taxes from discontinuing operations		167		295	685
Profit before taxes	1,480	1,934	2,451	4,297	7,964
Tax paid in the period	-14	-12	-84	-118	-608
Depreciation, amortisation and write-downs	410	343	817	686	1,452
Income from associated companies and joint ventures	-37	71	-104	44	27
Onerous contracts	-73	-96	-321	-87	237
Fair value adjustments	-234	-279	334	-923	-867
Change in working capital	-379	61	-913	-773	-170
Other changes	242	268	527	544	1,150
Gain on disposal discontinuing operations					-365
Cash-flow from operating activities related to discontinuing operations		-61		45	155
Net cash flow from operating activities	1,394	2,228	2,707	3,714	8,975
Cash-flow from purchase and sale of PPE, licenses and other intangible assets	-632	-676	-1,010	-1,291	-2,348
Proceeds from disposal of group companies, net of cash		45		45	4,454
Proceeds from disposal of other financial investments					45
Cash-flow form other investing activities	8		-43	35	37
Cash-flow from investing activates related to discontinuing operations		-303		-303	-412
Net cash flow from investing activities	-624	-934	1,053	-1,515	1,775
Change in interest-bearing liabilities	4,178	1,288	3,501	-111	-7,947
Payment of instalments on lease liabilities	-101	-77	-198	-160	-321
Payment of interest on lease liabilities	-29	-23	-60	-47	-102
Net interest paid financing activities	-200	-240	-390	-459	-1,145
Dividend	-4,645	-2,694	-4,645	-2,694	-2,748
Change in non-controlling interests				-671	-755
Other changes	-10		-10		
Cash-flow from financing activities related to discontinuing operations		-3		71	29
Net cash flow from financing activities	-807	-1,751	1,803	-4,071	-12,989
Net change in cash and cash equivalents	-37	-456	149	-1,872	-2,239
Currency translation of cash and cash equivalents	-3	-6	-9	4	-7
Cash and cash equivalents as at the start of the period	667	1,000	785	2,713	2,713
Cash and cash equivalents discontinuing operations at start of the period		625		319	319
Cash and cash equivalents as at period end	627	1,163	627	1,163	785
Of which cash and cash equivalents in discontinuing operations at period end		379		379	
Cash and cash equivalents excluding discontinuing operations at period end	627	784	627	784	785

Figure 3.6 displays page 13 of Salmar's Q2 2024 report. It is chunked using a fixed-size strategy with 2000 chunk size and 0 overlap.

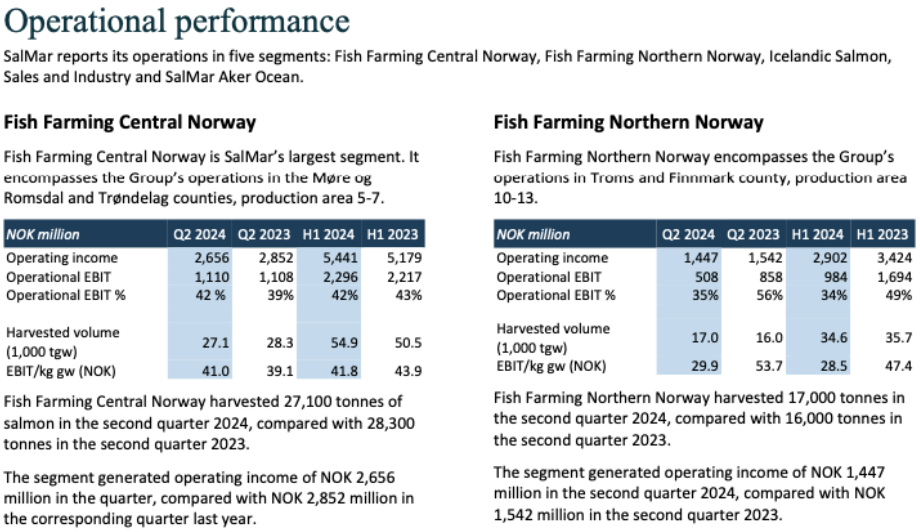
While chunking can also split text in awkward places, such as in the middle of sentences (see Figures 3.2a and 3.3a), the impact is usually less severe. Text tends to be concise and self-contained, with individual sentences or paragraphs often providing enough information for a query. Tables, however, rely on both semantic and spatial relationships across rows and columns, making them more vulnerable to losing context when split.

To address this issue, we propose a strategy that processes Norwegian financial documents differently based on their content; tables should be extracted and chunked separately to preserve their structure, while text can be handled using conventional chunking methods.

3.3.2 The context problem

An issue with tables is that their contents are often linked to certain segments or explanatory text which means that even if we manage to chunk coherently adhering to the tabular structure of the table, all the necessary context might not be captured within the

table itself. To illustrate this, consider the two tables in Figure 3.7. A financial analyst might pose the query: *"What is the operational income of Fish Farming Central Norway for Salmar for Q2 2024?"*.



Operational performance

SalMar reports its operations in five segments: Fish Farming Central Norway, Fish Farming Northern Norway, Icelandic Salmon, Sales and Industry and SalMar Aker Ocean.

Fish Farming Central Norway

Fish Farming Central Norway is SalMar's largest segment. It encompasses the Group's operations in the Møre og Romsdal and Trøndelag counties, production area 5-7.

NOK million	Q2 2024	Q2 2023	H1 2024	H1 2023
Operating income	2,656	2,852	5,441	5,179
Operational EBIT	1,110	1,108	2,296	2,217
Operational EBIT %	42 %	39%	42%	43%
Harvested volume (1,000 tgw)	27.1	28.3	54.9	50.5
EBIT/kg gw (NOK)	41.0	39.1	41.8	43.9

Fish Farming Central Norway harvested 27,100 tonnes of salmon in the second quarter 2024, compared with 28,300 tonnes in the second quarter 2023.

The segment generated operating income of NOK 2,656 million in the quarter, compared with NOK 2,852 million in the corresponding quarter last year.

Fish Farming Northern Norway

Fish Farming Northern Norway encompasses the Group's operations in Troms and Finnmark county, production area 10-13.

NOK million	Q2 2024	Q2 2023	H1 2024	H1 2023
Operating income	1,447	1,542	2,902	3,424
Operational EBIT	508	858	984	1,694
Operational EBIT %	35%	56%	34%	49%
Harvested volume (1,000 tgw)	17.0	16.0	34.6	35.7
EBIT/kg gw (NOK)	29.9	53.7	28.5	47.4

Fish Farming Northern Norway harvested 17,000 tonnes in the second quarter 2024, compared with 16,000 tonnes in the second quarter 2023.

The segment generated operating income of NOK 1,447 million in the second quarter 2024, compared with NOK 1,542 million in the second quarter 2023.

Figure 3.7: Salmar Q2 report page 5

For a human, it is clear that the left-hand table contains the relevant information. However, when using a retrieval system that relies solely on the parsed table, the tables are (except for numbers within) exact matches. This underscores the significance of the surrounding context, as these tables are structurally similar but differ in content, making it challenging for the RAG system to differentiate between them. Therefore, simply chunking tables independently introduces a new challenge: missing surrounding context.

3.3.3 Table summarization

To address the context problem, we propose a strategy that enriches table chunks by adding relevant contextual information through summarization. Similar approaches have been explored in works by Lance Martin (Martin, 2024) and Greg Kamradt (Kamradt, 2024), suggesting the generation of summaries of multi-modal elements using LLMs to enhance retrieval. However, these methods do not explicitly account for the surrounding text, which, as illustrated in Figure 3.7, often provides essential context. This surrounding text can offer valuable details about the table's content, such as the segment. To address this limitation, we propose a three-step approach:

1. **Parsing Documents**: The financial documents are parsed into JSON format using the Unstructured API.[13] This API extracts document elements, enabling us to

[13]Documentation for the Unstructured API is available at: https://docs.unstructured.io/welcome

categorize *"Title"*, *"Table"*, and regular text, referred to in the JSON as *"NarrativeText"*.

2. **Contextual Summarization**: To create summaries, we iterate over the JSON elements. When a *"Table"* element is encountered, we retrieve the text from the preceding five *"NarrativeText"* elements or until a *"Title"* element is reached. The title, if present, is also included to enrich the context. The step of processing JSON elements to create content for summarization prompts is described in detail in *Algorithm 1: Process JSON Elements*.

Algorithm 1 Process JSON Elements

```
Initialize currentTitle ← NULL
narrativeBuffer ← Empty List
tableContexts ← Empty List
for each element in jsonData.elements do
    if element.type = "Title" then
        currentTitle ← element.content
        narrativeBuffer ← Empty List                    ▷ Reset buffer
    else if element.type = "NarrativeText" then
        narrativeBuffer.append(element.content)
        if length(narrativeBuffer) > 5 then
            narrativeBuffer.removeFirst()
        end if
    else if element.type = "Table" then
        context ← copy(narrativeBuffer)
        if currentTitle ≠ NULL then
            context.prepend(currentTitle)
        end if
        tableContexts.append(context)          ▷ Add context to tableContexts
    end if
end for
```

3. **Summary Generation**: The extracted text, along with the parsed table, is incorporated into a prompt to generate the table summary.

For each parsed table element, the next step in the process was to create a prompt that accurately summarizes the content. In developing this prompt, we explored a variety of common prompting techniques and refined the elements that the LLM should prioritize. Recognizing that the prompt is a pivotal component of the entire pipeline, we invested considerable time into its curation. However, we acknowledge that prompt engineering is not an exact science and that there remains potential for further enhancement. The prompt used is displayed in Figure 3.8.

Referring back to the example with the two similar tables in Figure 3.7, we tested our new

Figure 3.8: Prompt for summarizing tables

Write a coherent accurate summary focusing on the column and row names based on a table. Mention the surrounding (above) text as it can give information about what the table is about. For financial metrics, this is important because the financial data can be linked to segments — we need to include the segment. Don't use words like "this table", "the columns", or "the surrounding text"; just focus on the contents. Also, mention which document the table is a part of.

Example:
This table is a part of the Q2 report for Mike's Juices

Above text:

Financials Wholesale

For the second quarter, the wholesale numbers for both juice and apples went up in North Dakota for Mike's Juices

Example table:

```
Product Name    Price    Quantity    Category    Supplier
Apple           $1.00    50          Fruits      Farm Fresh
Orange Juice    $2.50    30          Beverages   Juice Co.
```

Example Response:
In the second quarter, Mike's Juices in North Dakota increased wholesale numbers, selling 50 apples at $1.00 each in the Fruits category supplied by Farm Fresh, and 30 units of orange juice at $2.50 each in the Beverages category from Juice Co.

Based on the information above create a summary for this:

This table is a part of the {file_description}
Here is the surrounding (above) text and the table:

{table_text}

methodology. By including the text preceding the table up to the title ("Fish Farming Central Norway") and using the table summarization prompt, we generated a summary to represent the chunk. This approach achieved a cosine similarity score of 0.84 for the table, while the raw text representation received a score of 0.67. These results show that our method improves the semantic quality of chunks in multi-modal financial documents by enriching the context.

The final step in this approach is to map and replace the table summary with the original

raw table and its related preceding text. While the LLM-generated summary improves chunking and retrieval by providing higher similarity scores, it can sometimes change or leave out important details from the table. For this reason, this strategy is only used during the **r**etrieval process in the RAG pipeline.

For the **a**ugmentation and **g**eneration steps, we use the raw table and its context. This ensures that the user query is handled with the complete, unchanged information. This step is also important for our evaluation in Section 3.4, where accuracy is tested against specific keywords from the report. If the summary changes certain words, it might reduce the measured accuracy, even if the summary is retrieved correctly.

A detailed overview of the table processing pipeline is shown in Figure 3.9. While standard chunking techniques (covered in 3.2) are applied to text, the steps for creating summaries and mapping tables are specific to table processing.

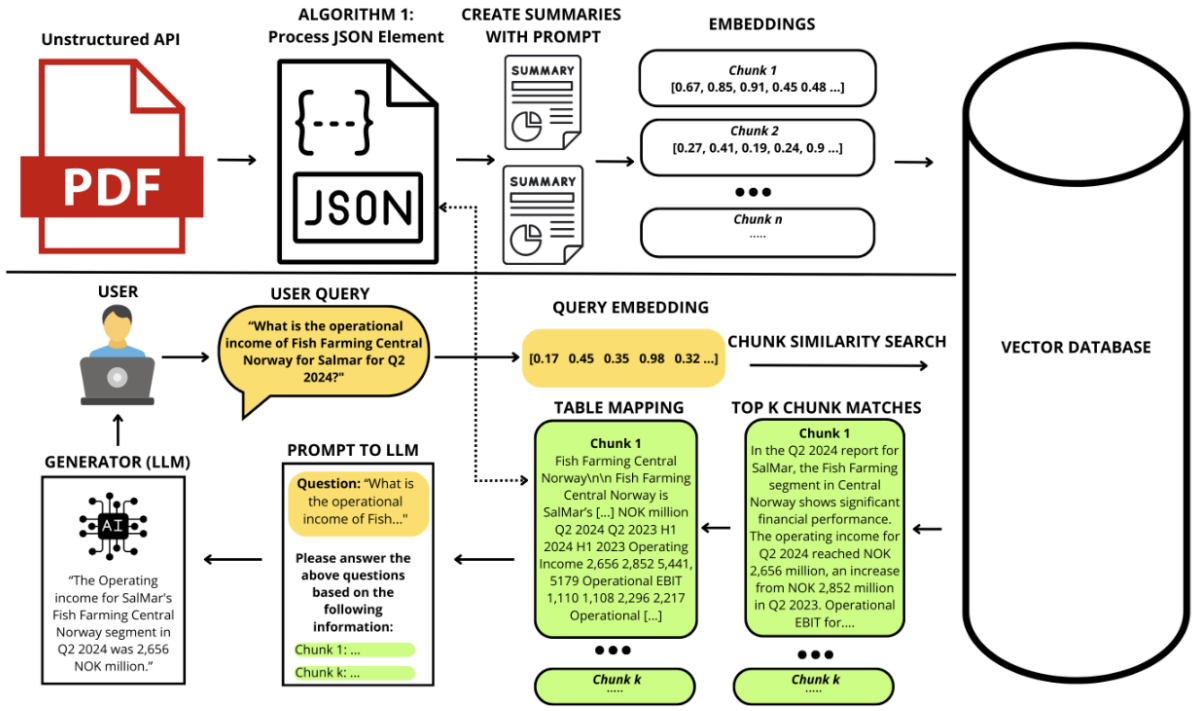


Figure 3.9: Table Summary Retrieval Pipeline

3.4 Retrieval Evaluation

Evaluating the retrieval component is crucial for assessing how well different chunking strategies deliver relevant information in a question-answering system. We develop two metrics to measure the retrieval system's performance: Full Keyword Coverage (Section 3.4.1), which assesses retrieval accuracy, and Tokens to Answer (Section 3.4.2), which measures retrieval efficiency.

To explain these metrics, we revisit the example in Figure 3.1, introduced in Section 3.1.3. Here, we modify the example by removing the ground truth, as the evaluation does not focus on whether the LLM provides the correct answer to the query. Instead, it checks if the *RAG system* can retrieve the most relevant chunks of the documents to address the query, focusing only on the query and the necessary keywords, as shown in Figure 3.10.

Query: "What is the operational income of Fish Farming Central Norway for Salmar for Q2 2024?"
Necessary Keywords: 'Fish Farming Central Norway', 'Q2 2024', 'Operating income', 'NOK million', '2,656'

Figure 3.10: Query and Necessary Keywords from Salmar Q2 2024 document

3.4.1 Accuracy - Full Keyword Coverage

The *Full Keyword Coverage* metric evaluates whether the retrieved content contains all the necessary keywords required to answer a query. Specifically, it verifies whether at least one retrieved chunk fully contains all the essential information within the *top-k* chunks.

We argue that all necessary keywords should be in a single chunk for two reasons. First, analysts often ask very specific, single-hop questions, as reflected in the creation of *NorwgianFinanceQA* dataset (section 3.1.3). These questions typically reference a specific section of a document, meaning the relevant information should be concentrated and captured within one chunk. Additionally, splitting sentences or tables during chunking can lead to incomplete or incoherent information. While some may argue that having keywords scattered across multiple chunks is sufficient for text, this approach can result in a loss of coherence, which is especially critical in tables as discussed in Section 3.3.1. For example, if a table is split and the necessary keywords are divided between chunks, the LLM may struggle to understand the context and fail to answer the query correctly.

To illustrate this metric, consider Figure 3.11, which shows a RAG system retrieving the top 2 most relevant chunks for the example query in Figure 3.10. Chunk 1 partially covers the necessary keywords, mentioning "Salmar," "Q2 2024," and "operating income," but it does not provide the exact metric or explicitly link it to "Fish Farming Central Norway." In contrast, Chunk 2 includes all the required keywords: the region ("Fish Farming Central Norway"), the time period ("Q2 2024"), the financial metric ("operating income"), the currency/unit ("NOK million"), and the exact amount ("2,656"). While Chunk 1 contains some correct information, it is classified as not accurate. Chunk 2, however, achieves Full Keyword Coverage. Based on this query, the evaluation would mark the retrieval as accurate since at least one of the chunks retrieved fully contains all

the relevant keywords.

Figure 3.11: The two most relevant chunks to a query

Chunk 1

"Salmar reported results for Q2 2024. The operating income across various regions accumulated to a large increase from the previous two quarters."

Chunk 2

"In Q2 2024, Salmar reported an operating income of 2,656 NOK million for its Fish Farming Central Norway operations."

In our retrieval system, for each query in *NorwegianFinanceQA*, the specified chunking strategy retrieves the top **20** chunks deemed most relevant to answer the query. The choice of 20 was based on analyst discussions, balancing between having enough chunks to ensure the correct one is included, and not overwhelming analysts with too many results to validate. Retrieving more than 20 chunks would also increase costs while retrieving fewer might exclude the correct chunk. Moreover, synthesizing specific information from large retrievals during the generation phase in RAG can negatively impact performance, as key details may become "lost in the middle" (Liu et al., 2024). This trade-off led us to settle on 20 chunks as the optimal threshold for this metric.

We can express Full Keyword Coverage by letting $\mathcal{D}_i = \{d_{i,1}, d_{i,2}, \ldots, d_{i,M_i}\}$ represent the set of necessary keywords for query i, where M_i is the total number of keywords. Let $\mathcal{C}_i = \{c_{i,1}, c_{i,2}, \ldots, c_{i,K}\}$ denote the set of top $K = 20$ retrieved chunks for query i. For each chunk $c_{i,k} \in \mathcal{C}_i$, where $k \in \{1, 2, \ldots, K\}$, we compute the number of matched keywords as:

$$\text{matches}_{i,k} = \sum_{m=1}^{M_i} \mathbf{1}\left(d_{i,m} \in c_{i,k}\right),$$

where $\mathbf{1}(d_{i,m} \in c_{i,k})$ is the indicator function, which equals 1 if the keyword $d_{i,m}$ is present in the chunk $c_{i,k}$, and 0 otherwise.

Next, the coverage ratio for each chunk $c_{i,k}$ is defined as:

$$\text{coverage}_{i,k} = \frac{\text{matches}_{i,k}}{M_i} = \frac{\sum_{m=1}^{M_i} \mathbf{1}\left(d_{i,m} \in c_{i,k}\right)}{M_i}.$$

The maximum coverage across all retrieved chunks for query i is then computed as:

$$\text{max\_coverage}_i = \max_{1 \le k \le K} \text{coverage}_{i,k}.$$

If $\text{max\_coverage}_i = 1$, it implies that at least one chunk $c_{i,k}$ contains all the necessary

keywords. This result is expressed as a binary indicator:

$$\text{max\_coverage\_chunk\_bin}_i = \begin{cases} 1 & \text{if max\_coverage}_i = 1, \\ 0 & \text{otherwise.} \end{cases}$$

In our example, $\text{max\_coverage\_chunk\_bin}_i = 1$, signifying that the retrieval step succeeded in providing a single chunk containing all critical information required to answer the query.

3.4.2 Efficiency - Tokens To Answer

While Full Keyword Coverage ensures that at least one retrieved chunk fully meets the information requirements, the *Tokens To Answer* metric evaluates how efficiently this coverage is achieved. This includes the position of the first fully comprehensive chunk, referred to as the *Retrieved Chunk Index*, and the cumulative number of tokens required to reach it.

Let $\mathcal{D}_i = \{d_{i,1}, \ldots, d_{i,M_i}\}$ denote the set of necessary keywords for query i, where M_i is the total number of keywords. Let $\mathcal{C}_i = \{c_{i,1}, \ldots, c_{i,K}\}$ represent the top K retrieved chunks, where $K = 100$ for this metric. For each chunk $c_{i,k}$, the coverage is defined as:

$$\text{coverage}_{i,k} = \frac{\sum_{m=1}^{M_i} \mathbf{1}(d_{i,m} \in c_{i,k})}{M_i},$$

where $\mathbf{1}(d_{i,m} \in c_{i,k})$ is 1 if the keyword $d_{i,m}$ is present in $c_{i,k}$, and 0 otherwise.

The first fully comprehensive chunk is identified as:

$$k^* = \min\{k \mid \text{coverage}_{i,k} = 1\}.$$

If such a chunk exists, k^* is the Retrieved Chunk Index. The Tokens To Answer metric then sums the token counts of all chunks up to k^*:

$$\text{TokensToAnswer}_i = \sum_{k=1}^{k^*} T_{i,k},$$

where $T_{i,k}$ is the token count for chunk $c_{i,k}$.

If no chunk achieves full coverage (k^* does not exist), we calculate the total tokens across all K retrieved chunks:

$$\text{TokensToAnswer}_i = \sum_{k=1}^{K} T_{i,k}.$$

For the example in Figure 3.11, Chunk 2 achieves full coverage, with $\text{coverage}_{i,2} = 1.0$, making $k^* = 2$. Using the token counts $T_{i,1} = 28$ and $T_{i,2} = 28$, the Tokens To Answer metric is calculated as $\text{TokensToAnswer}_i = T_{i,1} + T_{i,2} = 28 + 28 = 56$. This result indicates that 56 tokens are needed to reach a chunk that fully covers the query's necessary keywords.

It is important to note that for the Tokens To Answer metric, we set $K = 100$. While the accuracy metric uses $K = 20$, which is more representative of real-world scenarios, $K = 100$ was chosen for this metric to better assess how far down the retrieved chunks the relevant information might be located. This ensures that enough chunks are included to capture all potentially relevant information. In cases where no chunk achieves full coverage, we *impute* the cost of missing information by treating non-retrieved chunks as being 100 times the size of the chunk size used. This imputation accounts for the extra effort required to locate missing information while avoiding excessive penalization when full coverage is not achieved. Additionally, we note that all queries and chunks for obtaining these metrics, using *NorwegianFinanceQA*, were embedded using OpenAI's `text-embedding-3-large` embedding model.

4 Results and Analysis

In this section, we present and analyze the results of our experiments, focusing on the performance of different chunking strategies: character-based, recursive, semantic, and table-specific. We evaluate their effectiveness in retrieving relevant information from financial documents, discussing the key metrics of accuracy (Full Keyword Coverage), and efficiency (Tokens To Answer), as well as reviewing query type performance (text vs. tables). Finally, we address limitations and suggest directions for future research to refine retrieval strategies further.

4.1 Main Findings

We present and analyze the results of various chunking strategies tested on *NorwegianFinanceQA*. Chunk sizes tested ranged from 400 to 4,000 characters (specifically, 400, 800, 1,600, 2,400, and 4,000). To maintain consistency, the chunk overlap was set to 25% of the chunk size, allowing us to focus primarily on the impact of chunk size on performance.

Table 4.1 displays the best-performing configuration for each chunk strategy, evaluated by Full Keyword Coverage. The default recursive splitter with a chunk size of 4,000 characters and a 1,000-character overlap achieved the highest accuracy, retrieving the correct chunk in 94% of the top 20 retrieved chunks across all queries. For table summarization (TS) approaches, the recursive splitter with a chunk size of 800 characters and a 200-character overlap delivered the best performance, achieving 90% accuracy. Default strategies and table-summary strategies demonstrated notable differences in their ability to handle text and tables. Table-specific strategies excelled in table query accuracy, retrieving the correct chunk for all table questions (100%). However, they struggled with text queries. On the other hand, default strategies provided more balanced performance, handling both text and table queries effectively.

Strategy	Chunk Size	Overlap	Accuracy	Accuracy Text	Accuracy Table
Character	4000	1000	0.91	0.91	0.92
Recursive	4000	1000	**0.94**	**0.93**	0.94
Semantic	N/A	N/A	0.87	0.83	0.92
TS Character	2400	600	0.88	0.77	**1.00**
TS Recursive	800	200	0.90	0.82	**1.00**
TS Semantic	N/A	N/A	0.76	0.57	**1.00**

Table 4.1: Accuracy comparison between selected strategies

While retrieving the correct chunk is crucial, it must be considered alongside a validation process. Section 2.3 outlined that more effective retrieval appeals to analysts as it reduces

the need for extensive validation. Table 4.2 highlights the number of tokens required to retrieve a chunk containing all necessary keywords, revealing significant differences in efficiency across strategies. The table-specific recursive strategy (chunk size: 800, overlap: 200) demonstrated the best overall performance, requiring the fewest tokens on average to find the correct chunk. Additionally, there are differences in retrieval efficiency between text-based and table-based queries. Default approaches perform better for text queries, whereas table summarization enables highly efficient retrieval for table queries, although being less efficient for text queries.

Strategy	Chunk Size	Overlap	Efficiency	Efficiency Text	Efficiency Table
Character	1600	400	7,250	3,289	12,365
Recursive	800	200	5,911	**1,878**	11,118
Semantic	N/A	N/A	21,817	24,282	18,634
TS Character	800	200	4,106	6,861	**550**
TS Recursive	800	200	**2,423**	3,873	**550**
TS Semantic	N/A	N/A	12,683	22,089	**540**

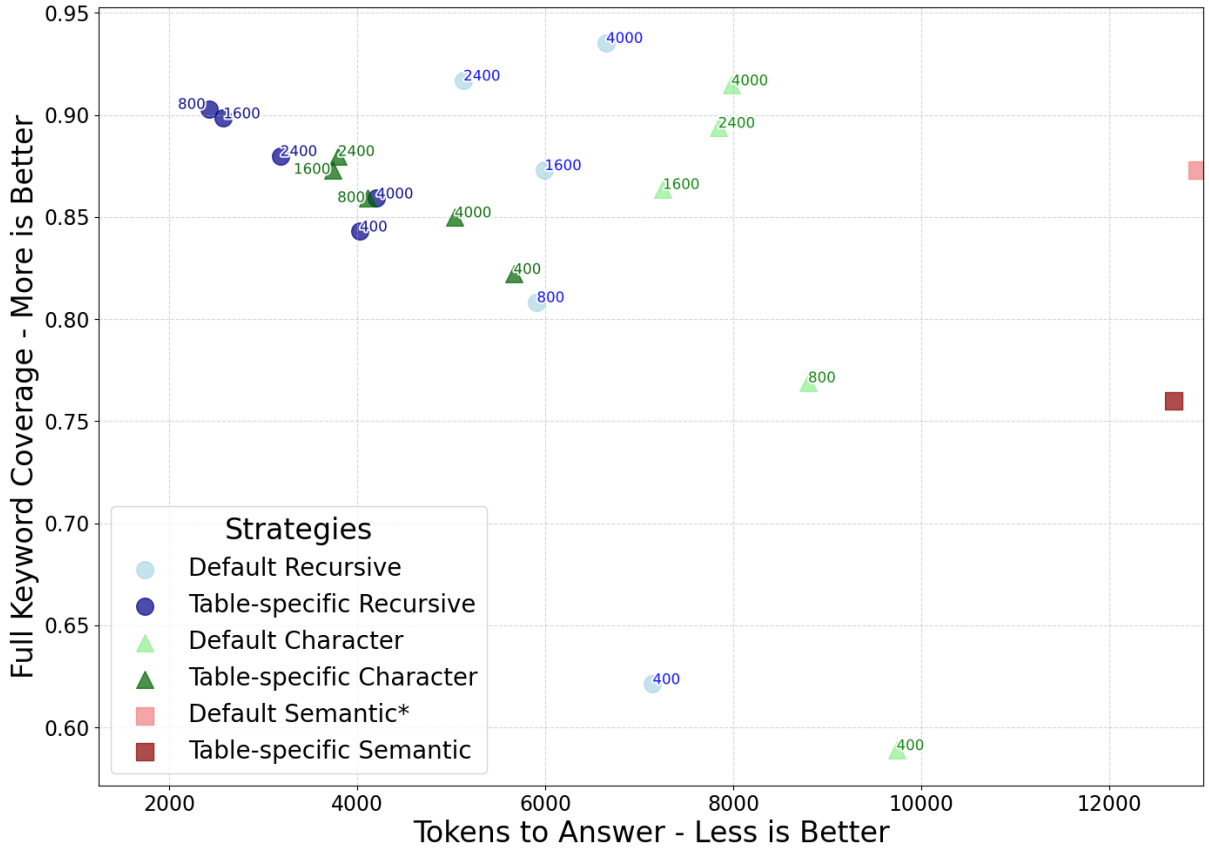
Table 4.2: Efficiency comparison between selected strategies

To evaluate the most promising overall strategy we display Full Keyword Coverage (accuracy) and Tokens To Answer (efficiency) for all tested strategies in Figure 4.1. Detailed results for all strategies can be found in Appendix B. The most balanced approaches, located at the top left of Figure 4.1, are table-specific recursive strategies utilizing small to medium-sized chunks (chunk size 800 and 1600).[14]

Recursive chunking strategies outperform their character chunking counterparts for equal chunk sizes in terms of both accuracy and efficiency. Semantic chunking seems like the least efficient strategy, requiring large amounts of tokens to retrieve an answer. The best efficiency is observed in table-specific chunking, which, as shown in Table 4.2, is due to their optimized retrieval for table-related queries.

For default strategies, there is a clear positive relationship between chunk size and accuracy - larger chunks tend to improve accuracy. However, this comes at the expense of efficiency, which initially improves with larger chunk sizes but eventually declines beyond a certain point.

[14]We define "small" and "large" chunk sizes based on both practical considerations of what may be too large for analyst validation and insights from existing literature. For example, LlamaIndex considers 128 tokens (approximately 512 characters) as a small chunk size (Theja, 2023).



* The default semantic splitter Tokens To Answer is artificially set lower to not skew the figure.

Figure 4.1: Scatter plot summarizing Accuracy and Efficiency for all tested strategies.

4.2 Element based chunking

In Section 2.3, we introduced hypothesis **H1-a**, which suggests that element-based chunking methods, like chunking tables separately, are better at retrieving accurate and relevant information compared to simpler, naive methods. This is because financial documents often have an unstructured layout that requires more specialized approaches. In this section, we explore whether element-based methods perform better. First, we compare how the default chunking strategies perform against each other (Section 4.2.1). Then, we look at how table-specific strategies compare to those that do not use table-specific methods (Section 4.2.2).

4.2.1 Default Strategies

Table 4.3 shows the average and maximum accuracy, as well as the average and minimum efficiency, for each strategy. Character chunking is less accurate and efficient compared to the recursive approach, while the semantic chunking strategy has the lowest maximum

accuracy. The mean accuracy for semantic chunking is higher because the character and recursive strategies include some poorly performing chunks. Semantic chunking has the lowest efficiency among the strategies, with both the mean and minimum being notably larger.

Table 4.3: Accuracy and Efficiency for Default Strategies

Strategy	Accuracy mean	Accuracy Max	Efficiency Mean	Efficiency Min
Character	0.80	0.91	8,320	7,250
Recursive	0.83	0.94	6,165	5,131
Semantic	0.87	0.87	21,817	21,817

We earlier suggested that recursive chunking might be better than character chunking because it considers the structure of documents. This is especially true for financial documents, which often have frequent paragraph shifts. Since the overall trend shows that recursive chunking performs better than character chunking, this supports hypothesis H1-a, which states that splitting methods based on document elements improve performance.

Similarly, based on our hypothesis, we anticipated that semantic chunking would outperform naive methods like character chunking. By splitting text based on semantic meaning, semantic chunking indirectly preserves the structure of the text, maintaining coherence and logical flow within chunks. However, table 4.3 shows that it performs worse than character chunking, which goes against our hypothesis. We argue that this underperformance may stem from the domain-specific nature of financial documents, which poses challenges for semantic chunking in detecting nuanced shifts in meaning. Semantic chunking may perform better in documents with clearer and more distinct topic transitions, but struggles to adapt to the complex language and structure of financial texts.

We can further extend our analysis by evaluating the chunking strategies performance on table and text queries, as seen in Figure 4.2. The recursive strategy generally performs better than the character strategy across various chunk sizes and query types. The figure also shows that the accuracy averages for recursive and character chunking in Table 4.3 are pulled down by certain chunk sizes with very low accuracy. While semantic chunking has the highest overall average, it doesn't consistently outperform these methods in this context. The figure also highlights a notable trend: text-based queries generally achieve higher accuracy and efficiency compared to table-based queries for all methods, except semantic chunking. This shows the added complexity of handling tabular data, which, as discussed in 3.3.1, requires preserving structural and contextual relationships - something that is less critical for text-based data.

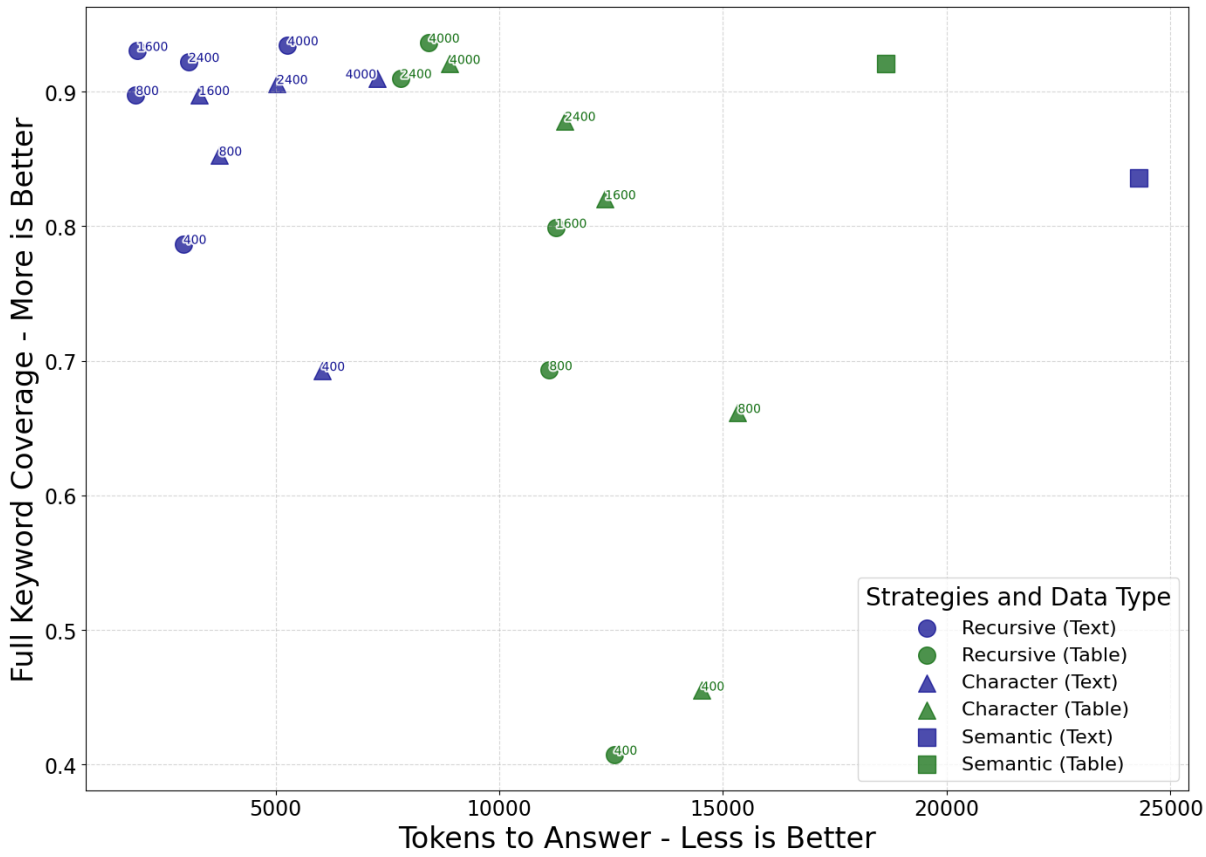


Figure 4.2: Scatter plot Accuracy and Efficiency (text vs. table)

4.2.2 Table Summarization Approach

The table summarization approach was evaluated by comparing its performance to the default strategies across both table and text queries. Table 4.4 presents the average performance across all chunking strategies. The results demonstrate that the table-specific and default approaches do not have any notable differences in average accuracy across the whole corpus. However, when we break down the results by query type, clear distinctions emerge. The table summarization approach correctly finds the relevant chunk for all table queries, achieving perfect accuracy. However, this comes at the expense of a lower average score for text queries, suggesting a limitation in performance when handling diverse query types.

Strategy	Avg. Accuracy	Avg. Accuracy Text	Avg Accuracy Table
Default	0.84	0.86	0.80
TS	0.83	0.70	1.00

Table 4.4: Accuracy default vs. table-specific

Table 4.5 provides a closer look at the average retrieval efficiency for text and table queries across default and table summarization strategies. The results show that the table-specific approach uses substantially fewer tokens to retrieve the necessary keywords compared to

the default method. With the default approach, users need to process 1.8 times more tokens to cover all relevant keywords compared to the table summarization method. This efficiency is due to the table-specific strategy's strong performance in quickly retrieving the correct information for table queries. Again, we see the improvement for table queries comes at the expense of lower retrieval efficiency for text queries, where it performs worse than the default strategies.

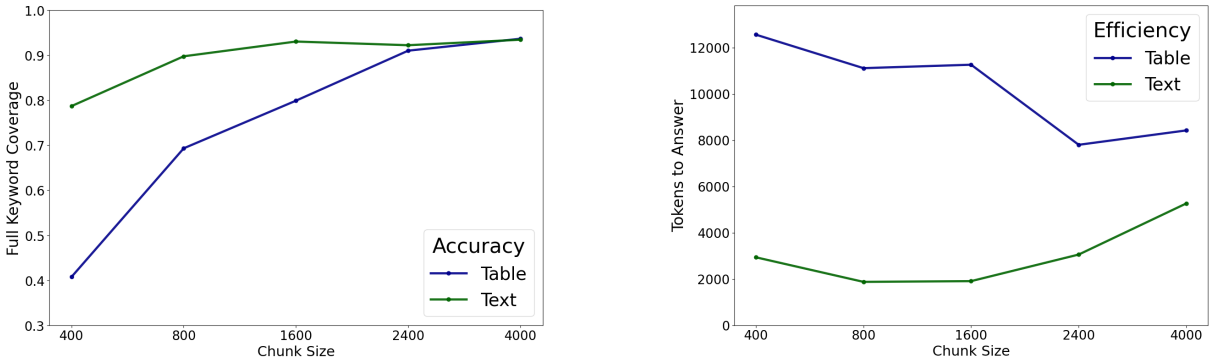
Strategy	Avg. Efficiency	Avg. Efficiency Text	Avg. Efficiency Table
Default	12,101	10,789	13,795
TS	6,811	11,666	546

Table 4.5: Efficiency default vs. table-specific

Overall, the table-specific strategy exhibits exceptional performance for table queries, as evidenced by the perfect table query accuracy in Table 4.4 and its reduced tokens used for retrieval shown in Table 4.5. These results support hypothesis H1-a, as the table-specific strategy demonstrates how specialized handling of tabular data preserves critical contextual relationships and outperforms naive methods in retrieving accurate and relevant information. However, this comes with a notable trade-off, as text-query accuracy and efficiency declines due to the system's prioritization of table representations.

4.3 Chunk size analysis

In section 2.3, we present hypothesis **H1-b** where we hypothesize that shorter contexts perform better than longer due to more specific semantic representations. To analyze this we test retrieval across various chunk sizes for the recursive strategy to see how the chunk size affects performance on tables and text queries. We base our analysis on the recursive strategy approach, as we observed similar results for character chunking, but avoid redundancy by focusing on one strategy. All results can be found in Appendix B.



(a) Accuracy - Full Keyword Coverage **(b)** Efficiency - Tokens To Answer

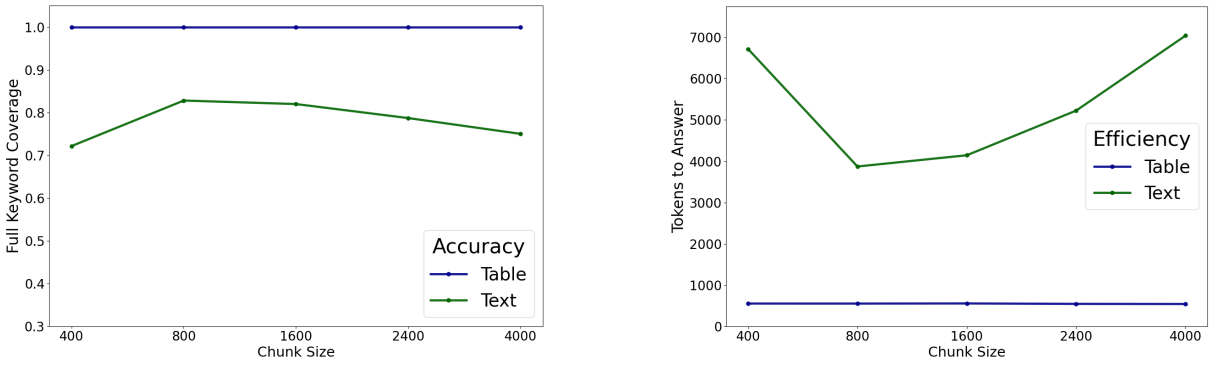
Figure 4.3: Performance of the Recursive Strategy on different chunk sizes

Figure 4.3a shows an interesting pattern across different chunk sizes: smaller chunk sizes

reveal clear differences between table and text queries. For text queries, accuracy starts relatively high at around 80% for the smallest chunk size, improves slightly as chunk sizes increase, and stabilizes around 800 characters, with minimal gains beyond that. In contrast, table query accuracy starts very low at around 40% for the smallest chunk size (400 characters) but improves significantly as chunk sizes increase, appearing to stabilize around 2400 characters. However, this stabilization is less consistent, as larger chunk sizes generally seem to further improve retrieval accuracy.

Figure 4.3b further demonstrates that smaller chunk sizes (around 800 characters) are optimal for text queries, as retrieval efficiency declines with larger chunks. For table-related queries, efficiency decreases steadily as chunk sizes grow, likely because larger chunks are necessary to capture the full context of extensive tables.

Figure 4.4 presents the same experiment using the table-specific approach, helping us evaluate how table summaries work alongside the three conventional chunking strategies. From this figure, we observe that for text queries, efficiency declines after a chunk size of 800 characters, confirming the trend shown in Table 4.3b. Accuracy is also quite high at 800 characters but decreases beyond that, indicating poorer accuracy for larger chunk sizes in text queries. In summary, Figures 4.3 and 4.4 suggest that 800 characters may be the optimal chunk size for text queries. Larger chunk sizes tend to reduce efficiency and either maintain the same accuracy (Figure 4.3) or lead to worse accuracy (Figure 4.4). Since 800 is a relatively small chunk size, this supports our hypothesis that smaller chunk sizes are better for text queries. However, the same does not hold true for table queries, where larger chunk sizes are often necessary.



(a) Accuracy - Full Keyword Coverage **(b)** Efficiency - Tokens To Answer

Figure 4.4: Performance of the TS Recursive Strategy on different chunk sizes

Furthermore, a comparison between Figures 4.4 and 4.3 reveals that while the retrieval of table queries becomes more reliable with table-specific strategies, the performance on text queries declines. This performance drop is most pronounced at the extremes: small chunk sizes, where the text fragments are too narrowly defined, and large chunk sizes, where the context becomes overly broad and diluted. Under these conditions, the retrieval system

tends to rely more on the summarized table chunks, as their representations often remain more semantically aligned with the query than the text-only chunks.

We further explore potential reasons for the reduced performance observed at both small and large chunk sizes. While no significant anomalies were found for the smallest chunk size of 400 characters, an interesting pattern emerged for chunk size 4000. Specifically, the drop in performance for text-based queries was most pronounced for numerical reasoning tasks compared to text reasoning tasks, as shown in Table 4.6. The table-specific recursive splitter at a chunk size of 4000 achieved only 67% accuracy on numerical reasoning tasks, compared to 81% for text reasoning. In contrast, the default recursive approach performed significantly better at the same chunk size, achieving 93% accuracy for numerical reasoning and 89% for text reasoning. These findings, as shown in Table 4.6, can be explained by the nature of table summaries, which are derived directly from financial tables that are heavily populated with numerical data. Queries involving numerical reasoning often reference numbers embedded in textual explanations. However, because table summaries dominate the semantic space for numerical content, they can overshadow text chunks that contain the correct numerical answers. As a result, the system may fail to retrieve the appropriate text-based chunks, leading to lower accuracy for numerical text reasoning queries.

Chunking Strategy	**Number Reasoning**	**Text Reasoning**
TS Recursive (chunk size, 4000)	67%	81%
Default Recursive (chunk size, 4000)	93%	89%

Table 4.6: Accuracy for Number and Text Reasoning Tasks

From Figure 4.4 we observe that the best accuracy as well as efficiency for text retrieval occurs at chunk sizes around 800. This closely aligns with the mean character count of 785 observed in the table summaries we created. This suggests that embedding lengths of similar sizes work together more effectively. This may also explain the relatively poor performance of semantic chunking, where the default semantic splitter has a mean chunk size of 3,834 and a standard deviation of 5,283, showing a large variation in chunk sizes.

In the context of **H1-b**, we observe that for small to medium-sized chunks in text-related queries, retrieval is both accurate and efficient. Chunk sizes around 800 demonstrate the best balance of accuracy and efficiency for these queries. This supports our hypothesis that shorter contexts provide more specific semantic representations, enabling efficient and precise retrieval. For table-related queries, on the other hand, larger chunk sizes are necessary to preserve the broader context required for accurate retrieval. The largest chunk size (4,000) yields the highest accuracy, as it allows the retrieval system to capture more of the tabular data. This highlights a key trade-off: while larger chunks benefit table queries, they can negatively impact the efficiency of text queries due to overly broad

semantic representations.

4.4 Limitations and Future Work

RAG is a new, and rapidly evolving technology, where best practices are continuously shifting. In this work, we propose techniques to evaluate retrieval and contribute to the ongoing discussion on how to effectively assess RAG systems. However, we acknowledge that our methodology is not without limitations.

The Accuracy (Full Necessary Keywords) metric was designed to account for the redundancy often present in financial information, where the same details may appear in multiple locations. This approach relies on identifying specific words rather than the exact location of the information. In our testing, however, Necessary Keywords finds a match when there is an exact word match, meaning synonyms and minor variations are not captured. Future work could involve developing a word-matching system that is more robust to subtle differences.

The Efficiency (Tokens To Answer) metric also has limitations, as it is dependent on Full Keyword Coverage. If a retrieval strategy fails to meet Full Keyword Coverage, Tokens To Answer cannot be calculated, requiring us to impute a value. Not imputing would disproportionately favor strategies with poor Accuracy, while excessive imputation risks unfairly penalizing strategies, artificially inflating the Efficiency for certain chunking strategies. To address this, we opted to impute at one hundred times the chunk size, as we found it offered a balanced solution.

Overall, our findings indicate that table summaries can be highly effective, although they do not integrate seamlessly with conventional text-splitting techniques. This opens up opportunities for future research to explore combining text summarization with table summarization for improved retrieval. Research, such as Anthropic's study on contextual embeddings (Anthropic, 2024), as well as our own work, has highlighted the great potential of summarization for improving RAG performance. This opens up opportunities for future research to explore the performance differences between general text summarization techniques and approaches that incorporate specific structural elements. For instance, investigating whether summarization strategies tailored to the unique characteristics of tables or specific document sections outperform more generic methods. These findings could further refine retrieval systems to achieve a better balance between Accuracy and Efficiency across diverse content types in unstructured financial documents.

5 Conclusion

In this thesis, we set out to address the limitations of LLMs in financial research by enhancing RAG systems. Specifically, we examined how different chunk strategies influence the retrieval accuracy and efficiency of financial information, a crucial requirement for high-precision domains like finance.

Our study presented several important findings. Among the evaluated chunking methods, recursive chunking with a large character limit emerged as the most accurate strategy for retrieval, achieving 94% accuracy. However, this strategy came at the cost of efficiency, requiring analysts to process a higher number of tokens. Table-specific chunking strategies delivered exceptional performance for table-based queries, achieving 100% accuracy and high efficiency. Despite their success, table-specific strategies struggled with text-based queries, highlighting a trade-off between handling multi-modal elements and preserving overall accuracy.

One of the key contributions of this thesis is the creation of *NorwegianFinanceQA*, the first open-source dataset specifically tailored for evaluating RAG systems in Norwegian financial contexts. By incorporating both text and table-based queries, the dataset provides a robust foundation for benchmarking retrieval methods in unstructured and multi-modal financial documents. Additionally, our approach to table summarization demonstrates the importance of adapting retrieval systems to the unique structure of financial documents.

This thesis highlights the critical role of chunking in optimizing RAG systems for financial research. By addressing the challenges of accuracy, efficiency, and multimodality, we lay the groundwork for more reliable and practical LLM tools in the finance sector, empowering analysts with enhanced capabilities for information retrieval and decision-making

References

Antematter. (2023). Optimizing rag: Advanced chunking techniques study. https://antematter.io/blogs/optimizing-rag-advanced-chunking-techniques-study

Anthropic. (2024). Contextual retrieval. https://www.anthropic.com/news/contextual-retrieval

Balažević, I., Allen, C., & Hospedales, T. (2019). Multi-relational poincaré graph embeddings. *arXiv preprint arXiv:1905.09791*.

Bansal, A., & Suddala, S. (2024). Enhancing generative ai capabilities through retrieval-augmented generation systems and llms. *Library Progress International*, *44*(3), 17765–17775.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., ... Amodei, D. (2020). Language models are few-shot learners. *arXiv preprint arXiv:2005.14165*.

Caspari, L., Dastidar, K. G., Zerhoudi, S., Mitrovic, J., & Granitzer, M. (2024). Beyond benchmarks: Evaluating embedding model similarity for retrieval-augmented generation systems. *arXiv preprint arXiv:2407.08275*.

Chatrath, V., Lotif, M., & Raza, S. (2024). Fact or fiction? can llms be reliable annotators for political truths? *arXiv preprint arXiv:2411.05775*.

CMU School of Computer Science. (n.d.). Word embedding demo tutorial. https://www.cs.cmu.edu/~dst/WordEmbeddingDemo/tutorial.html

Databricks. (2024). Long-context rag performance: How llms handle large inputs. https://www.databricks.com/blog/long-context-rag-performance-llms?utm_source=chatgpt.com

Devlin, J., Chang, M.-W., Lee, K., & Toutanova, K. (2019). Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Donavan, R. (2024). Breaking up is hard to do: Chunking in rag applications. https://stackoverflow.blog/2024/06/06/breaking-up-is-hard-to-do-chunking-in-rag-applications/

Fleischer, D., Berchansky, M., Wasserblat, M., & Izsak, P. (2024). RAG Foundry: A framework for enhancing LLMs for retrieval augmented generation. *arXiv preprint arXiv:2408.02545*.

Gao, Y., Xiong, Y., Gao, X., Jia, K., Pan, J., Bi, Y., Dai, Y., Sun, J., Wang, M., & Wang, H. (2024). Retrieval-augmented generation for large language models: A survey. *arXiv preprint arXiv:2312.10997*.

Gekhman, Z., Yona, G., Aharoni, R., Eyal, M., Feder, A., Reichart, R., & Herzig, J. (2024). Does fine-tuning llms on new knowledge encourage hallucinations? *arXiv preprint arXiv:2405.05904*.

Günther, M., Mohr, I., Williams, D. J., Wang, B., & Xiao, H. (2024). Late chunking: Contextual chunk embeddings using long-context embedding models. *arXiv preprint arXiv:2409.04701*.

Guu, K., Lee, K., Tung, Z., Pasupat, P., & Chang, M.-W. (2020). Realm: Retrieval-augmented language model pre-training. *arXiv preprint arXiv:2002.08909*.

Han, Z. F., Lin, J., Gurung, A., Thomas, D. R., Chen, E., Borchers, C., Gupta, S., & Koedinger, K. R. (2024). Improving assessment of tutoring practices using retrieval-augmented generation. *arXiv preprint arXiv:2402.14594*.

Islam, P., Kannappan, A., Kiela, D., Qian, R., Scherrer, N., & Vidgen, B. (2023). Financebench: A new benchmark for financial question answering. *arXiv preprint arXiv:2311.11944*.

Kamradt, G. (2024). Levels of text splitting: Recursive character splitting. https://github.com/FullStackRetrieval-com/RetrievalTutorials/tree/a4570f3c4883eb9b835b0ee18990e62298f518tutorials/LevelsOfTextSplitting#RecursiveCharacterSplitting

Kasneci, E., Sessler, K., Küchemann, S., Bannert, M., Dementieva, D., Fischer, F., Gasser, U., Groh, G., Günnemann, S., Hüllermeier, E., Krusche, S., Kutyniok, G., Michaeli, T., Nerdel, C., Pfeffer, J., Poquet, O., Sailer, M., Schmidt, A., Seidel, T., ... Kasneci, G. (2023). Chatgpt for good? on opportunities and challenges of large language models for education. *Learning and Individual Differences*, *103*, 102274.

Levonian, Z., Li, C., Zhu, W., Gade, A., Henkel, O., Postle, M.-E., & Xing, W. (2023). Retrieval-augmented generation to improve math question-answering: Trade-offs between groundedness and human preference. *arXiv preprint arXiv:2310.03184*.

Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., Riedel, S., & Kiela, D. (2021). Retrieval-augmented generation for knowledge-intensive nlp tasks. *arXiv preprint arXiv:2005.11401*.

Li, H., Su, Y., Cai, D., Wang, Y., & Liu, L. (2022). A survey on retrieval-augmented text generation. *arXiv preprint arXiv:2202.01110*.

Li, X., Chan, S., Zhu, X., Pei, Y., Ma, Z., Liu, X., & Shah, S. (2023). Are chatgpt and gpt-4 general-purpose solvers for financial text analytics? a study on several typical tasks. *arXiv preprint arXiv:2305.05862*.

Liu, N. F., Lin, K., Hewitt, J., Paranjape, A., Bevilacqua, M., Petroni, F., & Liang, P. (2024). Lost in the middle: How language models use long contexts. *Transactions of the Association for Computational Linguistics*, *12*, 157–173.

Martin, L. (2024). Multi-modal retrieval-augmented generation. https://github.com/langchain-ai/langchain/blob/master/cookbook/Multi_modal_RAG.ipynb

Mikolov, T., Chen, K., Corrado, G., & Dean, J. (2013). Efficient estimation of word representations in vector space. *arXiv preprint arXiv:1301.3781*.

Mishra, A. (2024). Five levels of chunking strategies in rag: Notes from greg's video. https://medium.com/@anuragmishra_27746/five-levels-of-chunking-strategies-in-rag-notes-from-gregs-video-7b735895694d

Pipitone, N., & Alami, G. H. (2024). Legalbench-rag: A benchmark for retrieval-augmented generation in the legal domain. *arXiv preprint arXiv:2408.10343*.

Qu, R., Tu, R., & Bao, F. (2024). Is semantic chunking worth the computational cost? *arXiv preprint arXiv:2410.13070*.

Schwaber-Cohen, R. (2023). Chunking strategies for vector search in pinecone. https://www.pinecone.io/learn/chunking-strategies/

Setty, S., Thakkar, H., Lee, A., Chung, E., & Vidra, N. (2024). Improving retrieval for rag-based question answering models on financial documents. *arXiv preprint arXiv:2404.07221*.

Smith, B., & Troynikov, A. (2024, July). *Evaluating chunking strategies for retrieval* (tech. rep.). Chroma. https://research.trychroma.com/evaluating-chunking

Son, H. (2024a). Jpmorgan chase launches artificial intelligence assistant inspired by chatgpt. https://www.cnbc.com/2024/08/09/jpmorgan-chase-ai-artificial-intelligence-assistant-chatgpt-openai.html

Son, H. (2024b). Morgan stanley rolls out openai-powered chatbot for wall street division. https://www.cnbc.com/2024/10/23/morgan-stanley-rolls-out-openai-powered-chatbot-for-wall-street-division.html

Theja, R. (2023, October). Evaluating the ideal chunk size for a rag system using llamaindex. https://www.llamaindex.ai/blog/evaluating-the-ideal-chunk-size-for-a-rag-system-using-llamaindex-6207e5d3fec5

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., & Polosukhin, I. (2023). Attention is all you need. *arXiv preprint arXiv:1706.03762*.

Wiratunga, N., Abeyratne, R., Jayawardena, L., Martin, K., Massie, S., Nkisi-Orji, I., Weerasinghe, R., Liret, A., & Fleisch, B. (2024). Cbr-rag: Case-based reasoning for retrieval-augmented generation in llms for legal question answering. In J. A. Recio-Garcia, M. G. Orozco-del-Castillo, & D. Bridge (Eds.), *Case-based reasoning research and development* (pp. 445–460). Springer Nature Switzerland.

Woo, J. J., Yang, A. J., Olsen, R. J., Hasan, S. S., Nawabi, D. H., Nwachukwu, B. U., Williams, R. J., & Ramkumar, P. N. (2024). Custom large language models improve accuracy: Comparing retrieval augmented generation and artificial intelligence agents to non-custom models for evidence-based medicine. *Arthroscopy: The Journal of Arthroscopic Related Surgery*.

Wu, J., Zhu, J., Qi, Y., Chen, J., Xu, M., Menolascina, F., & Grau, V. (2024). Medical graph rag: Towards safe medical large language model via graph retrieval-augmented generation. *arXiv preprint arXiv:2408.04187*.

Wu, S., Xiong, Y., Cui, Y., Wu, H., Chen, C., Yuan, Y., Huang, L., Liu, X., Kuo, T.-W., & Xue, C. J. (2024). Retrieval-augmented generation for natural language processing: A survey. *arXiv preprint arXiv:2407.13193*.

Xiong, G., Jin, Q., Wang, X., Zhang, M., Lu, Z., & Zhang, A. (2024). Improving retrieval-augmented generation in medicine with iterative follow-up questions. In *Biocomputing 2025* (pp. 199–214).

Xu, J. (2024, July). Genai and llm for financial institutions: A corporate strategic survey [Standard Chartered Bank Singapore].

Yepes, A. J., You, Y., Milczek, J., Laverde, S., & Li, R. (2024). Financial report chunking for effective retrieval-augmented generation. *arXiv preprint arXiv:2402.05131*.

Yu, H., Gan, A., Zhang, K., Tong, S., Liu, Q., & Liu, Z. (2024). Evaluation of retrieval-augmented generation: A survey. *arXiv preprint arXiv:2405.07437*.

Zhang, Y., Li, Y., Cui, L., Cai, D., Liu, L., Fu, T., Huang, X., Zhao, E., Zhang, Y., Chen, Y., Wang, L., Luu, A. T., Bi, W., Shi, F., & Shi, S. (2023). Siren's Song in the AI Ocean: A survey on hallucination in large language models. *arXiv preprint arXiv:2309.01219*.

Zhao, R., Chen, H., Wang, W., Jiao, F., Do, X. L., Qin, C., Ding, B., Guo, X., Li, M., Li, X., & Joty, S. (2023). Retrieving multimodal information for augmented generation: A survey. *arXiv preprint arXiv:2303.10868*.

Zhong, Z., Liu, H., Cui, X., Zhang, X., & Qin, Z. (2024). Mix-of-granularity: Optimize the chunking granularity for retrieval-augmented generation. *arXiv preprint arXiv:2406.00456*.

Declaration on the use of AI tools in the work on this master's thesis

Name (and version) of the AI tools: ChatGPT-4o, ChatGPT-o1, ChatGPT-o1-mini, Claude 3.5 Sonnet.

Purpose of using the tools: We utilized AI to generate the test corpus, employed AI-powered embedding models, and leveraged AI for language refinement.

We are aware that we are responsible for all content of this master's thesis, including the parts where AI tools are used. We are responsible for ensuring that the thesis complies with ethical rules for privacy and publication.

Appendices

A Chunking Examples

A.1 Fixed-Size Chunking Example

This example demonstrates how the first two chunks are extracted from the SalMar Q2 2024 report, as shown in Figure 3.2b. With a chunk size of $s = 800$ characters and an overlap of $o = 100$ characters, the text T is split into manageable segments. The selected portion of 1500 characters is chosen for simplicity, as it creates exactly two chunks.

Example Text: T

SECOND QUARTER / 2024
Financial performance
Summary
The Farming segments in Norway had a satisfactory quarter with a positive development in biological status and in cost levels. Harsh winter conditions and extreme weather earlier in the year continued to negatively impact the superior share in the quarter. This had an adverse effect on price achievement, particularly for Farming Northern Norway. Sales and Industry continued to perform well, driven by efficient management of harvesting and processing facilities. The combination of high contract share and high spot prices however, resulted in a negative contribution from contracts. Icelandic Salmon faced high costs on harvested volume due to biological challenges and low harvest volume. SalMar Aker Ocean successfully transferred smolt to its Ocean Farm 1. The next harvest is set for 2025.
In the second quarter 2024, the SalMar Group harvested 44,800 tonnes of salmon in total, up from 44,300 tonnes in the second quarter 2023. The Norskott Havbruk joint venture (Scottish Sea Farms) delivered another solid quarter, with improved biological status in all regions. In the second quarter 2024, the price of salmon (NASDAQ Salmon Index) averaged NOK 110.9 per kg, up from NOK 107.1 per kg in the second quarter in 2023.
Income statement for the second quarter 2024
Operating revenues amounted to NOK 5,838 million in the second quarter 2024, compared with NOK 5,895 million in the second quarter 2023. SalMar's most important key performance

Example Calculations

The number of chunks k is calculated using the formula:

$$k = \left\lceil \frac{n - o}{s - o} \right\rceil$$

Substituting the values:

$$n = 1500,\ s = 800,\ o = 100$$

$$k = \left\lceil \frac{1500 - 100}{800 - 100} \right\rceil = \left\lceil \frac{1400}{700} \right\rceil = \lceil 2 \rceil = 2$$

Further, using the formulas for the indices of each chunk:

$$I_i = (i - 1) \times (s - o), \quad J_i = I_i + s$$

we calculate the indices for the two chunks ($k = 2$):

$$\text{Chunk 1: } I_1 = (1 - 1) \times (800 - 100) = 0, \quad J_1 = 0 + 800 = 800$$

$$\text{Chunk 2: } I_2 = (2 - 1) \times (800 - 100) = 700, \quad J_2 = 700 + 800 = 1500$$

The chunks will be extracted accordingly:

$$C_i = T[I_i : J_i] \quad \text{for } i = 1, 2, \ldots, k$$

Since $k = 2$, we extract:

$$C_1 = T[0 : 800], \quad C_2 = T[700 : 1500]$$

Resulting Chunks:

Chunk 1: Text[0:800]

Chunk 1

SECOND QUARTER / 2024
Financial performance
Summary
The Farming segments in Norway had a satisfactory quarter with a positive development in biological status and in cost levels. Harsh winter conditions and extreme weather earlier in the year continued to negatively impact the superior share in the quarter. This had an adverse effect on price achievement, particularly for Farming Northern Norway. Sales and Industry continued to perform well, driven by efficient management of harvesting and processing facilities. The combination of high contract share and high spot prices however, resulted in a negative contribution from contracts. Icelandic Salmon faced high costs on harvested volume due to biological challenges and low harvest volume. SalMar Aker Ocean successfully transferred smolt to it

Chunk 2: Text[700–1500]:

Chunk 2

biological challenges and low harvest volume.
SalMar Aker Ocean successfully transferred smolt to its Ocean Farm 1. The next harvest is set for 2025.
In the second quarter 2024, the SalMar Group harvested 44,800 tonnes of salmon in total, up from 44,300 tonnes in the second quarter 2023. The Norskott Havbruk joint venture (Scottish Sea Farms) delivered another solid quarter, with improved biological status in all regions. In the second quarter 2024, the price of salmon (NASDAQ Salmon Index) averaged NOK 110.9 per kg, up from NOK 107.1 per kg in the second quarter in 2023.
Income statement for the second quarter 2024
Operating revenues amounted to NOK 5,838 million in the second quarter 2024, compared with NOK 5,895 million in the second quarter 2023. SalMar's most important key performance

The overlapping content "biological challenges and low harvest volume. SalMar Aker Ocean successfully transferred smo", appears at the end of Chunk 1 and the start of Chunk 2. This duplication maintains context between chunks, ensuring coherence and continuity.

A.2 Recursive Chunking Example

This example demonstrates how recursive splitting works using the formulas described in Section 3.2.2. We use the same text from Section A.1 to illustrate how the recursive strategy differs. The text is represented as a string:

Example Text: T

'SECOND QUARTER / 2024\nFinancial performance\nSummary\nThe Farming segments in Norway had a satisfactory quarter\nwith a positive development in biological status and in cost\nlevels. Harsh winter conditions and extreme weather earlier\n in the year continued to negatively impact the superior\nshare in the quarter. This had an adverse effect on price\nachievement, particularly for Farming Northern Norway.\nSales and Industry continued to perform well, driven by\nefficient management of harvesting and processing\nfacilities. The combination of high contract share and high\nspot prices however, resulted in a negative contribution\n from contracts.\nIcelandic Salmon faced high costs on harvested volume due\nto biological challenges and low harvest volume.\nSalMar Aker Ocean successfully transferred smolt to its\nOcean Farm 1. The next harvest is set for 2025.\n In the second quarter 2024, the SalMar Group harvested\n44,800 tonnes of salmon in total, up from 44,300 tonnes in\nthe second quarter 2023.\nThe Norskott Havbruk joint venture (Scottish Sea Farms)\ndelivered another solid quarter, with improved biological\n status in all regions.\nIn the second quarter 2024, the price of salmon (NASDAQ\n Salmon Index) averaged NOK 110.9 per kg, up from NOK\n107.1 per kg in the second quarter in 2023.\nIncome statement for the second quarter 2024\nOperating revenues amounted to NOK 5,838 million in the\nsecond quarter 2024, compared with NOK 5,895 million in\nthe second quarter 2023.\nSalMar's most important key performance'

Step 1: Recursive Splitting

Using the separator hierarchy $\Sigma = [\texttt{\textbackslash n\textbackslash n}, \texttt{\textbackslash n}, \texttt{" "}, \texttt{""}]$, the text is split iteratively. Starting with the first separator $\sigma_1 = \texttt{\textbackslash n\textbackslash n}$, we observe that no double line breaks exist in T. Thus, the entire text remains unsplit:

$$S = \{T\}, \quad \ell(S_1) = 1500.$$

Proceeding to the next separator $\sigma_2 = \texttt{\textbackslash n}$, the text is split into individual lines. Each resulting segment S_j is checked against $s = 800$. Since all lines have $\ell(S_j) < s$, no further splitting is required at this stage.

Step 2: Merging Chunks

Using the segments $C = \{C_1, C_2, \dots\}$ from Step 1, we merge smaller chunks to reduce fragmentation. Starting with $P = \emptyset$, we add each C_k iteratively:

$$P = P + C_k \quad \text{if } \ell(P) + \ell(C_k) \leq s.$$

- **Adding C_1:**

  $$\ell(C_1) = 21 \quad \text{(SECOND QUARTER / 2024)}.$$

  Since $\ell(P) + \ell(C_1) = 21 \leq 800$, C_1 becomes the initial chunk:

  $$P = C_1, \quad \ell(P) = 21.$$

- **Adding C_2:**

  $$\ell(C_2) = 24 \quad \text{("Financial performance")}.$$

  Adding C_2 results in:

  $$\ell(P) + \ell(C_2) = 21 + 24 = 45 \quad \text{(still under 800)}.$$

  Merge C_2 into P.

- **Continue to C_{16}:** Each C_k from C_3 to C_{16} is added, maintaining $\ell(P) \leq 800$.

- **At C_{17}:**

  $\ell(C_{17}) = 59$, ("SalMar Aker Ocean successfully transferred smolt to its"). $\ell(P) = 758$.

  Adding C_{17} results in $\ell(P) + \ell(C_{17}) = 817 > 800$. Finalize P as:

  $$D_1 = P, \quad \ell(D_1) = 758.$$

  Start a new chunk with C_{17}:

  $$P = C_{17}, \quad \ell(P) = 59.$$

Repeat this process for the remaining segments, ensuring each chunk D_j adheres to the size constraint $\ell(D_j) \leq s$. Since the remaining chunks fit within the size limit $s = 800$, the final result consists of two chunks:

$$D = \{D_1, D_2\}.$$

Step 3: Adding Overlaps

To preserve context, overlaps of size $o = 100$ characters are added between consecutive chunks. For each finalized chunk D_j (except the first), append the last o characters of the preceding chunk D_{j-1} to the start of D_j:

$$D'_j = D_{j-1}[\ell(D_{j-1}) - o :] + D_j, \quad \text{if } j > 1.$$

- **Chunk D'_1:** As D_1 is the first chunk, no overlap is added:

$$D'_1 = D_1.$$

- **Chunk D'_2:** Add the last 100 characters of D_1 to the start of D_2:

$$D'_2 = D_1[\ell(D_1) - 100 :] + D_2.$$

Final Output:

After applying these steps, the text is divided into two overlapping chunks:

$$D = \{D'_1, D'_2\}$$

Chunk 1

SECOND QUARTER / 2024
Financial performance
Summary
The Farming segments in Norway had a satisfactory quarter with a positive development in biological status and in cost levels. Harsh winter conditions and extreme weather earlier in the year continued to negatively impact the superior share in the quarter. This had an adverse effect on price achievement, particularly for Farming Northern Norway.
Sales and Industry continued to perform well, driven by efficient management of harvesting and processing facilities. The combination of high contract share and high spot prices however, resulted in a negative contribution from contracts.
Icelandic Salmon faced high costs on harvested volume due to biological challenges and low harvest volume.

Chunk 2

Icelandic Salmon faced high costs on harvested volume due to biological challenges and low harvest volume. SalMar Aker Ocean successfully transferred smolt to its Ocean Farm 1. The next harvest is set for 2025.

In the second quarter 2024, the SalMar Group harvested 44,800 tonnes of salmon in total, up from 44,300 tonnes in the second quarter 2023.

The Norskott Havbruk joint venture (Scottish Sea Farms) delivered another solid quarter, with improved biological status in all regions.

In the second quarter 2024, the price of salmon (NASDAQ Salmon Index) averaged NOK 110.9 per kg, up from NOK 107.1 per kg in the second quarter in 2023.

Income statement for the second quarter 2024

Operating revenues amounted to NOK 5,838 million in the second quarter 2024, compared with NOK 5,895 million in the second quarter 2023.

SalMar's most important key performance

B Results

B.1 Full Character Splitter Results

Table B.1: Character Default Splitter Results at k=20

Chunk Size	Overlap	FoundChunk	FoundChunkText	FoundChunkTable
400	100	0.5889	0.6926	0.4550
800	200	0.7691	0.8525	0.6614
1600	400	0.8637	0.8975	0.8201
2400	600	0.8938	0.9057	0.8783
4000	1000	0.9146	0.9098	0.9206

Table B.2: Character TS Splitter Results at k=20

Chunk Size	Overlap	FoundChunk	FoundChunkText	FoundChunkTable
400	100	0.8222	0.6844	1.0000
800	200	0.8591	0.7500	1.0000
1600	400	0.8730	0.7746	1.0000
2400	600	0.8799	0.7869	1.0000
4000	1000	0.8499	0.7336	1.0000

Table B.3: Character Default Token To Answer Splitter Results at k=100

Chunk Size	Overlap	TokenToAnswer Mean	TokenToAnswer Text	TokenToAnswer Table
400	100	9,736.58	14,515.73	6,034.70
800	200	8,796.13	15,329.58	37,35.39
1600	400	7,250.36	12,365.01	3,288.59
2400	600	7,839.88	11,465.34	5,031.64
4000	1000	7,979.03	8,891.05	7,272.58

Table B.4: Character TS Token To Answer Splitter Results at k=100

Chunk Size	Overlap	TokenToAnswer Mean	TokenToAnswer Text	TokenToAnswer Table
400	100	5,662.36	539.07	9,630.82
800	200	4,106.40	550.52	6,860.75
1600	400	3,739.44	551.85	6,208.51
2400	600	3,795.08	551.01	6,307.91
4000	1000	5,036.61	551.10	8,511.05

B.2 Full Recursive Splitter Results

Table B.5: Recursive Default Splitter Results at k=20

Chunk Size	Overlap	FoundChunk	FoundChunkText	FoundChunkTable
400	100	0.6212	0.7869	0.4074
800	200	0.8083	0.8975	0.6931
1600	400	0.8730	0.9303	0.7989
2400	600	0.9169	0.9221	0.9101
4000	1000	0.9353	0.9344	0.9365

Table B.6: Recursive TS Splitter Results at k=20

Chunk Size	Overlap	FoundChunk	FoundChunkText	FoundChunkTable
400	100	0.8430	0.7213	1.0000
800	200	0.9030	0.8279	1.0000
1600	400	0.8984	0.8197	1.0000
2400	600	0.8799	0.7869	1.0000
4000	1000	0.8591	0.7500	1.0000

Table B.7: Recursive Default Token To Answer Splitter Results at k=100

Chunk Size	Overlap	TokenToAnswer Mean	TokenToAnswer Text	TokenToAnswer Table
400	100	7,142.32	2,940.37	12,567.05
800	200	5,911.24	1,877.76	11,118.48
1600	400	5,993.95	1,909.56	11,266.92
2400	600	5,130.57	3,059.63	7,804.16
4000	1000	6,648.17	5,270.50	8,426.75

Table B.8: Recursive TS Token To Answer Splitter Results at k=100

Chunk Size	Overlap	TokenToAnswer Mean	TokenToAnswer Text	TokenToAnswer Table
400	100	4,027.17	6,719.69	551.11
800	200	2,422.66	3,872.79	550.54
1600	400	2,577.46	4,145.17	553.55
2400	600	3,182.97	5,227.59	543.36
4000	1000	4,205.13	7,043.80	540.40