Evaluation of Embedding Models for Swedish Text

Authors:
Dilan Chroscik
Pierre Chateau
Paulo Vaccari

Abstract

The main goal of this project was to investigate and improve how embedding models work with Swedish data, focusing on using practical techniques and finding open source alternatives. To do this, a dataset of around 100 Swedish PDF documents was gathered and approximately 1000 questions were written about these documents. After embedding the documents and inserting them into a vector database (ChromaDB), the questions were also embedded, and a query with similarity search was performed for each question. The results were then compiled and analyzed. Using this workflow, different parsers, embedding models, and various settings were tested in a systematic way, in an attempt to find improvements.

August 12, 2025

Contents

1 Introduction

Traditional keyword-based approaches often fail to capture semantic relationships between queries and documents, leading to suboptimal retrieval performance. Vector embedding-based approaches have emerged as a powerful alternative, representing text as high-dimensional vectors that capture semantic meaning. This new way of searching using natural language (NLP) and enriching a LLM is called **Retrieval-Augmented Generation (RAG)**. It has revolutionized how large language models (LLMs) can access and leverage external knowledge. By combining the generative power of LLMs with a retrieval system that fetches relevant information from a knowledge base, RAG systems can produce more accurate, factual, and up-to-date responses. This approach directly addresses the limitations of static pre-trained models, such as their tendency to "hallucinate" or provide outdated information.

What is Retrieval-Augmented Generation (RAG)

RAG is a framework that enhances the output of large language models (LLMs) by giving them access to an external knowledge base. Instead of relying solely on the information they were trained on, RAG allows LLMs to "look up" and reference specific, relevant data from a separate source before generating a response.

This process involves a few steps:

1. **Retrieval:** When a user asks a question, the system first retrieves relevant documents or data from an external knowledge base (in our case a set of documents). This is done using a vector database where text is stored as numerical representations (embeddings) that can be quickly searched for semantic similarity.
2. **Augmentation:** The retrieved information is then added to the user's original prompt. This provides the LLM with the necessary context and facts to answer the question accurately.
3. **Generation:** The LLM then uses this augmented prompt, which now contains both the user's query and the relevant external information, to generate a more precise, factual, and up-to-date response.

Problem Statement

While RAG has seen widespread success in English, its application in other languages, especially those with smaller digital footprints like Swedish, presents unique challenges. The effectiveness of a RAG system hinges on its ability to accurately parse documents, segment text into meaningful chunks, and use robust embedding models to create a vector space where semantic similarity can be reliably measured. The availability of high-quality, open-source tools and models specifically trained for the Swedish language is a key factor in building effective and scalable solutions.

Goals and Limitation

The primary goal of this project was to systematically investigate and improve the performance of RAG systems on Swedish-language data. We focused on evaluating different open-source alternatives and practical techniques for each stage of the RAG pipeline. This included testing various **PDF parsers**, **chunking strategies**, and **embedding models** to identify the most effective combination.

This report documents our methodology, findings, and analysis. We begin by outlining the theoretical foundation of RAG and its core components. Following this, we detail the data creation process and the systematic experiments conducted on distinct datasets in Swedish. Our findings reveal insights into how different architectural choices, from chunk size to text normalization, impact retrieval accuracy. This work provides a practical guide for developing a RAG solutions for Swedish-language applications.

2 Method

2.1 Data Creation

The Full dataset with TOML Question files and PDFs used in this Research can be found here: Google Drive folder.

2.1.1 Document Sourcing

Since we were five in our group, we could manually find 20 documents each quite quickly and start generating questions. No scraping script was needed.

2.1.2 Question Structure

To Evaluate a RAG Implementation we needed to create a Dataset that contained Questions and Answers in Swedish with information on what file and page that Q/A data pointed do. We decided to use TOML structure to generate our Question/Answer Dataset because it is more Human friendly and easy to see the structure and not have to worry about missing curly brackets and parenthesis compared to JSON structure or CSV files etc.

The format was decided to look like this:

```
[[questions]]
id = "PMCCPPN011"
question = "Hur skapar och kör du ett projekt i Visual Studio?"
answer = "Skapa projektet genom att välja Build>Build Solution på huvudmenyn"
difficulty = "Hard"
category = "Programming"

[[questions.files]]
file = "cpp-get-started-msvc-170-001.pdf"
page_numbers = [15, 16]
```

The id is the id of the person who created the question. Difficulty refers to the difference between the phrasing of the question and answer. It's not about the difficulty of the subject.

2.1.3 Question Writing

Some of us decide to use a LLM to assist in Question/Answer Creation. One of the Prompts that we have on our Github is the following System Prompt:

```
Du är en avancerad AI-assistent specialiserad på dokumentanalys och
frågeextraktion.

Ditt mål är att identifiera explicita frågor som kan besvaras direkt
från innehållet i ett dokument.

Skapa 12 stycken frågor.

För varje fråga: tillhandahåll ett svar i form av ett citat från texten.
Citatet från texten skall vara identisk med det från dokumentet.

**Utdataformat (TOML - liknande ChromaDB-struktur):**

Din utdata ska vara i TOML-format, strukturerad för att efterlikna en samling
frågor och deras associerade metadata (svar, källfiler, sidnummer).

**För varje identifierad fråga, tillhandahåll följande TOML-struktur:**
```

toml:

```toml
[[questions]]
# En unik identifierare för frågan (börja med siffran i exemplet.)
id = "unikt_fråge_id_N"
question = "Den fullständiga texten av den identifierade frågan på svenska."
answer = "Det citerade svaret som finns i dokumentet."
difficulty = "easy"
# Eller "medium" eller "hard"  baserat på hur lätt svaret är att hitta.
category = "physics"

# Kapslad tabell för fil- och sidinformation
[[questions.files]]
# Identifieraren för dokumentet där svaret finns. Anvend filnamnet i exemplet.
file = "dokument_namn_eller_id.pdf"
# En array med sidnummer där svaret finns. Första sidan är det som är angivet i exemplet.
page_numbers = [sidnummer_1, sidnummer_2]
```

2.2 Baseline RAG

This section provides a comprehensive overview of a RAG (Retrieval-Augmented Generation) system implemented in subdirectory: Baseline Code. The system first extracts text from PDF files, chunks it into token-based segments with optional overlap using tiktoken, generates embeddings using OpenAI's embedding models, stores these embeddings in a ChromaDB vector database, and then performs similarity searches against question embeddings to retrieve relevant document chunks for evaluation purposes. After that it compiles search results into CSV and Excel files with the results and also plots several graphs.

The architecture follows a modular design where each component serves a distinct purpose in processing documents through multiple stages:

- Document Parsing and Preprocessing
- Chunking with Overlap Management
- Tokenization
- Embedding Generation via API Integration
- Storage and Indexing into Vector Database
- Semantic Retrieval for Question Matching
- Evaluation and Result Compilation
- Generating Plots from Result Compilation

Each stage builds upon previous ones to ensure accuracy in identifying source documents and matching answers across query contexts.

2.2.1 Parsing

The system employs PyMuPDF (fitz) as its primary library for reading PDF files due to its ability to extract text while maintaining logical flow between pages and sections. It processes each page individually, applying necessary transformations such as removing line breaks at word boundaries or collapsing excessive whitespace.

Text Normalization Techniques

Text normalization ensures consistency across document content:

- Removal of hyphens followed by newlines (to prevent broken words)
- Replacement of multiple consecutive whitespaces with single space
- Correction of sentence endings to avoid erroneous spacing after punctuation marks

These steps ensure clean textual input suitable for downstream processing.

2.2.2 Chunking

Documents are segmented into fixed-length chunks using token counts rather than character or paragraph boundaries. This approach allows better handling of variable document structures while maintaining manageable sizes for embedding generation. The framework is using tiktoken from OpenAI.

Each PDF is parsed and chunked into tokens using the TikToken encoder associated with OpenAI's text-embedding-3-small model. These tokens are then organized into chunks defined by configurable parameters:

- Maximum token length ('MAX_TOKENS')
- Overlap between consecutive chunks ('OVERLAP')

Overlap Configuration Impact

When overlap is enabled, subsequent chunks share common tokens from the prior chunk's end portion. This mechanism increases coverage of important content that might straddle two boundary points and improves retrieval accuracy when matching against questions containing phrases spanning multiple segments.

2.2.3 Tokenization

TikToken encoding provides a standardized means to convert textual data into token representations compatible with various OpenAI models. For this implementation, text-embedding-3-small is used as an embedding model, ensuring alignment between segmentation logic and semantic representation.

Tokens are counted using precise encoding algorithms that respect word boundaries and punctuation marks appropriately. This enables accurate estimation of chunk sizes without causing truncation issues or loss of contextual meaning.

2.2.4 Embedding

The system utilizes the OpenAI Embeddings API to generate high-dimensional vector representations for both question queries and document chunks. These embeddings capture semantic similarities among texts.

API calls are batched where possible during embedding generation stages. Utilizing thread pooling reduces latency when generating multiple embeddings concurrently across different processes.

2.2.5 ChromaDB

ChromaDB is configured as a persistent client storing all generated embeddings in indexed collections. Each collection name includes version information derived from configuration variables like maximum token count and overlap values, ensuring reproducibility of experiments.

Every vector stored within the database contains associated metadata including:

- Unique chunk identifiers

- Source file names
- Page numbers corresponding to content origin
- Total number of chunks per document for internal reference

This metadata facilitates efficient post-retrieval filtering and ranking during query phases.

2.2.6 Result Evaluation

Results are evaluated using several criteria:

- Filename Match Accuracy
- Page Number Correspondence
- Textual Similarity Thresholds (based on match percentages)
- No-Match Detection Rates
- Distance Score Comparisons (Cosine Distances)

These metrics collectively provide insights into system robustness and effectiveness in real-world applications.

Final results are exported to CSV and Excel formats for detailed inspection:

- Tabular representation of matched documents and answer candidates
- Statistical summaries enabling analysis across question categories or difficulties
- Support for direct integration with downstream analytics pipelines

2.3 Measurements

The system calculates and visualizes standard information retrieval metrics:

- Accuracy: The proportion of queries with at least one correct result
- Precision: The proportion of retrieved results that are relevant
- Recall: The proportion of relevant results that are retrieved

These metrics are calculated for three levels of granularity:

1. File-level matching
2. Page-level matching
3. Chunk-level matching

2.4 Creating Second Dataset

After gathering some results with the original dataset, some intuition was developed. It was decided to test the intuition on a smaller and much more narrow dataset. The second dataset consisted of documents from Bekämpningsmedelsregistret - Kemikalieinspektionen.

These documents contain decisions and conditions regarding the usage of chemicals by farmers. Each product has one or many decisions (older ones are kept), each decision has around 3 - 4 short documents associated with them. The products are divided in two main categories, biocidal and plant protectors. Biocidal products often only have one file per decision, with a different format compared to the plant protector products. The filenames are well composed as they contain registration number and date of decision for all files, which can be used in the code.

Due to time constraints, limitations had to be set for the second dataset. It was chosen to only include two types of documents and biocidal products were excluded completely. The two document types, in each decision, that were chosen were deemed to have all the information a farmer would care about for that specific product. The remaining files in the decision had information aimed at the supplier of the product, instead of the user.

The filenames had the following structure:

`(Reg_Nr)_Bilaga_Villkor_(YYYY-MM-DD).pdf`
`(Reg_Nr)_Bilaga_Villkor_för_användning_(YYYY-MM-DD).pdf`

In early testing it became apparent that markdown conversion was beneficial for this dataset. Both visually and results-wise.

2.4.1 Writing Questions

The next step was to write questions, and the first challenge was to choose which string to choose as an answer for the text match. Since all answers were found in tables, and all the tables had the same layout across files, it was hard to grab a string just by copying directly from pdf. The parsed string of that section looks different depending on the parser, which also means that different parsers might need different normalizing strategies when matching text.

Additionally, since a parser is not perfect, it can interpret the same type of table differently across many files. The decision was made to write answers in markdown format, since we would be using markdown conversion for this dataset.

Due to the very similar content of the documents, it was hard to formulate questions that did not include a product name or chemical substance. No general questions, such as "Which products can be used potatoes", were written since a complete answer to that question cannot be simply extracted without adding more functionality to our code.

The metadata for the second dataset could include more fields than the first dataset. Fields were added for registration number, decision dates, decision type, and tags for tables for potential future use (if adding tags around tables in the markdown file). This metadata can easily be used for filtering. Particularly, filtering by registration number or decision date, both can be found in the filename as well.

This is the structure of the questions for the second dataset:

```
[[questions]]
id = "DC_5510_002"
question = "Vilka grödor får Pico 750 WG användas på?"
difficulty = "Easy"
subject = "Pico 750 WG"
reg_nr = "5510"
date_decision = "2025-06-12"
date_end = "2032-06-30"
decision nr = "5.1.1-B24-00648"
decision type = "Bifall" # or Utfasning or Återkall

[[questions.files]]
file_answer = "|Höstvete, höstråg, höstrågvete och höstkorn|"
file = "5510_Bilaga_Villkor_för_användning_2025-06-12.pdf"
page_numbers = [1]
tables = ["<TABELL = Användning och syfte, Sida = 1>"]
```

A csv file with registration numbers and product names were saved, for use in filtering and context addition. When generating chunks, this context can be prefixed to the chunk.

3 Results

In this section we present the results for the Original dataset, as well as the second dataset. First we discuss finding in the early exploration phase, after that we show tables with some further results which include variations of embedding models.

3.1 Original Dataset - Early Findings

Since there were many variables to vary in each attempted version to build the RAG system, early tests were used to develop intuition into which parameters had the most value to vary. Early findings were as follows:

Parsers: After trying a couple parsers on a smaller collection of the PDF files, PyMuPDF seemed to have a lot of functionality and good documentation. It was easier to use and gave the best parsed results. We also used the PyMuPDF4llm package to convert files to markdown.

Chunking: Early on, it was clear that a lower token size of 128 and 256 gave better results. The same with recursive versus regular character splitting. Recursive character splitting performed better.

Database: The space configuration of the vector database also showed consistently better results when cosine space was used instead of linear space (L2), which was standard in ChromaDB.

Normalizing: Normalizing before embedding and inserting into the database seemed to produce worse results, when parsing regularly and when converting to markdown format. The normalizing was done after the retrieval instead, for better text matching.

Markdown: Converting to markdown format did not seem to improve the results for this first dataset, so it was mostly tried on the best results, to look for eventual improvements.

These settings were set for most versions, built for the original dataset.

In the table below, the following settings were used for all rows:

- Parser - PyMuPDF
- Chunking - No Overlap
- Normalizing - Only after query
- Markdown - Not used here

3.2 Original Dataset - Result Tables

Chunk size	Chunk strategy	Embedding model	DB Space	Accuracy Chunks	Accuracy Files	Accuracy Pages	Precision Chunks	Precision Files	Precision Pages	Recall Chunks	Recall Files	Recall Pages
128	Recursive character	text-embedding-large	Cos	79%	97%	83%	78%	96%	82%	95%	96%	95%
128	Recursive character	multilingual-e5-large	Cos	79%	95%	80%	79%	95%	80%	100%	100%	100%
256	Recursive character	multilingual-e5-large	Cos	79%	94%	77%	79%	94%	77%	100%	100%	100%
512	Recursive character	multilingual-e5-large	Cos	77%	92%	74%	77%	92%	74%	100%	100%	100%
256	Recursive character	text-embedding-small	Cos	74%	93%	76%	74%	93%	76%	100%	100%	100%
256	Recursive character	text-embedding-small	L2	73%	93%	75%	73%	93%	75%	89%	92%	90%
256	Token	text-embedding-small	Cos	72%	93%	75%	70%	93%	75%	93%	94%	93%
256	Token	text-embedding-small	L2	71%	93%	75%	69%	93%	75%	84%	88%	85%
512	Token	text-embedding-small	L2	70%	90%	73%	69%	93%	73%	75%	80%	76%
128	Recursive character	jina-embeddings-v3	Cos	70%	93%	75%	70%	92%	75%	100%	100%	100%
128	Recursive character	text-embedding-small	Cos	70%	95%	79%	70%	95%	79%	97%	98%	97%
128	Recursive character	text-embedding-small	L2	69%	94%	78%	69%	94%	78%	94%	96%	95%
128	Recursive character	mistral-embed	Cos	66%	93%	74%	66%	93%	74%	100%	100%	100%
128	Token	text-embedding-small	L2	65%	93%	75%	63%	93%	75%	90%	93%	91%
128	Token	text-embedding-small	Cos	65%	93%	75%	65%	93%	75%	99%	100%	99%
128	Recursive character	sentence-bert-swedish-cased	Cos	59%	86%	65%	60%	87%	65%	93%	95%	94%
384	Token	all-MiniLM-L12-v2	L2	34%	68%	37%	37%	73%	40%	54%	70%	56%
128	Recursive character	all-MiniLM-L6-v2	Cos	29%	67%	35%	29%	67%	35%	95%	98%	96%
N/A	Semantic chunk	text-embedding-small	L2	73%	90%	68%	74%	90%	69%	99%	99%	99%

Table 1: Main results from original dataset

3.3 Original Dataset - Result Tables, for other parsers

Here are some results for parsers, other than PyMuPDF. All rows implement cosine space and recursive character split.

Chunk size	Parser	Embedding	Accuracy Chunks	Accuracy Files	Accuracy Pages	Precision Chunks	Precision Files	Precision Pages	Recall Chunks	Recall Files	Recall Pages
128	PyMuPDF4llm - Markdown	text-embedding-small	66%	93%	75%	66%	93%	75%	100%	100%	100%
128	Docling - Markdown	text-embedding-small	64%	89%	73%	65%	89%	73%	99%	99%	99%
128	Mistral OCR - Markdown	text-embedding-small	36%	91%	68%	35%	92%	68%	92%	97%	95%

Table 2: Main results from original dataset, with non-baseline parsers

3.4 Second Dataset - Early Findings

The second dataset was a narrow dataset, consisting of short documents (mostly tables) with very few variations. Making it necessary to yet again explore which variables to vary. Parsing

Again, PyMuPDF and PyMyPDF4llm performed the best when reading tables, even when comparing to table specific parses such as Camelot. It was also possible to parse the tables separate from the text, to help the parser out in complicated tables. This was used to insert tags around tables, however not enough time was left to optimize this to get better results.

Chunking It became clear quite quickly that larger token size limits gave better results. However, after inserting context (registration number and product name), lower token size limits performed better. Using recursive character split makes more sense here, since it might help to keep tables intact in the chunks.

Markdown Since the data consisted of a lot of tables, there was more to be gained from converting to markdown when parsing the documents, unlike the original dataset.

Context It is necessary to inject context in this dataset, especially for smaller token size limits. Unless metadata filtering is used, it could be hard to detect which document the chunks belong to. All documents have similar content, only a few keywords and headers change. Thus, it is helpful to inject the registration number and product name to each chunk, since the chunk might not include any distinguishing information.

Generating LLM Response Once the relevant chunks are retrieved from a query, they can be passed to an LLM as context to answer the original query. Due to time limitations, this was only done a few times with local models.

Reranking When relaying context to the LLM, one needs to think about tokens. It is more efficient to only pass a couple relevant chunks, rather than all retrieved chunks. For this purpose, a reranker was used, to only feed the top two results to the LLM as context.

In the table below, the following settings were used for all rows:

- Parser - PyMuPDF4llm
- Chunking - Recursive character splitting. No Overlap
- Database - Cosine space
- Normalizing - No normalizing, even after query. Due to markdown conversion
- Markdown - Used for all rows

3.5 Second Dataset - Result Tables

Chunk size	Embedding model	Accuracy Chunks	Accuracy Files	Accuracy Pages	Precision Chunks	Precision Files	Precision Pages	Recall Chunks	Recall Files	Recall Pages	Queries with results	Notes
1024	text-embedding-small	96%	100%	99%	96%	100%	99,00%	100%	100%	100%	100%	Context
2048	text-embedding-small	96%	100%	100%	96%	100%	100,00%	100%	100%	100%	100%	Context
512	text-embedding-small	96%	100%	100%	96%	100%	100,00%	100%	100%	100%	100%	Context
1024	jina-embeddings-v3	96%	100%	98%	96%	100%	98,00%	100%	100%	100%	100%	Context
2048	jina-embeddings-v3	96%	100%	100%	96%	100%	100,00%	100%	100%	100%	100%	Context
1024	text-embedding-large	96%	100%	100%	95%	100%	98,00%	98%	99%	99%	99%	Context
512	text-embedding-large	96%	100%	100%	95%	100%	100,00%	100%	100%	100%	100%	Context
2048	text-embedding-small	95%	98%	98%	94%	97%	97,00%	99%	100%	100%	100%	
512	multilingual-e5-large	94%	99%	97%	94%	99%	97,00%	100%	100%	100%	100%	Context
1024	text-embedding-small	94%	99%	97%	94%	98%	96,00%	99%	100%	100%	100%	
512	jina-embeddings-v3	89%	98%	94%	89%	98%	94,00%	100%	100%	100%	100%	Context
256	multilingual-e5-large	88%	100%	99%	88%	100%	99,00%	100%	100%	100%	100%	Context
512	text-embedding-small	88%	99%	96%	87%	98%	95,00%	99%	100%	99%	100%	
128	multilingual-e5-large	87%	100%	99%	87%	100%	99,00%	100%	100%	100%	100%	Context
512	sentence-bert-swedish-cased	87%	92%	90%	83%	90%	86,00%	91%	92%	92%	93%	Context
256	text-embedding-large	84%	99%	98%	84%	99%	98,00%	100%	100%	100%	100%	Context
128	text-embedding-large	81%	99%	97%	81%	99%	97,00%	100%	100%	100%	100%	Context
512	multilingual-e5-large	78%	99%	94%	78%	99%	94,00%	100%	100%	100%	100%	
256	text-embedding-small	77%	100%	97%	77%	100%	97,00%	100%	100%	100%	100%	Context
256	sentence-bert-swedish-cased	68%	94%	89%	66%	93%	88,00%	97%	98%	98%	98%	Context
128	text-embedding-small	67%	100%	98%	67%	100%	98,00%	100%	100%	100%	100%	Context
2048	text-embedding-large	65%	100%	100%	65%	98%	98,00%	93%	95%	95%	95%	Context
512	all-MiniLM-L6-v2	25%	60%	53%	24%	60%	52,00%	98%	99%	99%	100%	Context

Table 3: Main results from second dataset

3.6 Second Dataset - Result Tables, for other parsers

Here are some results for parsers, other than PyMuPDF. All rows implement cosine space and recursive character split.

Chunk size	Parser	Embedding model	Accuracy Chunks	Accuracy Files	Accuracy Pages	Precision Chunks	Precision Files	Precision Pages	Recall Chunks	Recall Files	Recall Pages
1024	Docling	text-embedding-small	91%	100%	99%	91%	100%	99%	100%	100%	100%
2048	Docling	text-embedding-small	91%	98%	98%	91%	97%	97%	99%	99%	99%
512	Docling	text-embedding-small	88%	97%	96%	88%	97%	95%	100%	100%	100%
2048	Mistral OCR	text-embedding-small	80%	96%	96%	75%	93%	93%	97%	97%	97%
1024	Mistral OCR	text-embedding-small	79%	99%	97%	77%	98%	96%	99%	100%	99%
512	Mistral OCR	text-embedding-small	73%	100%	97%	73%	100%	97%	100%	100%	100%
256	Docling	text-embedding-small	67%	99%	96%	67%	99%	96%	100%	100%	100%
128	Docling	text-embedding-small	62%	99%	97%	62%	99%	97%	100%	100%	100%
128	Mistral OCR	text-embedding-small	26%	97%	95%	26%	97%	94%	100%	100%	100%
128	Docling	text-embedding-small	2%	81%	75%	1%	80%	72%	100%	100%	100%

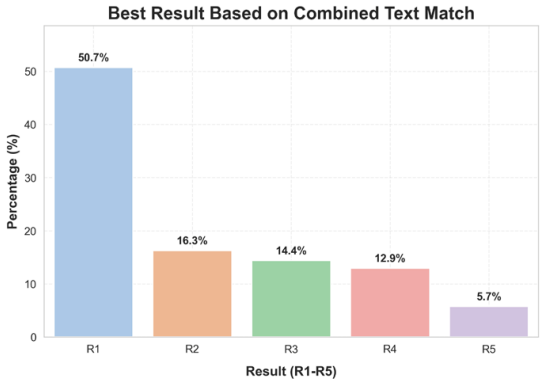
Table 4: Main results from second dataset, with non-baseline parsers.

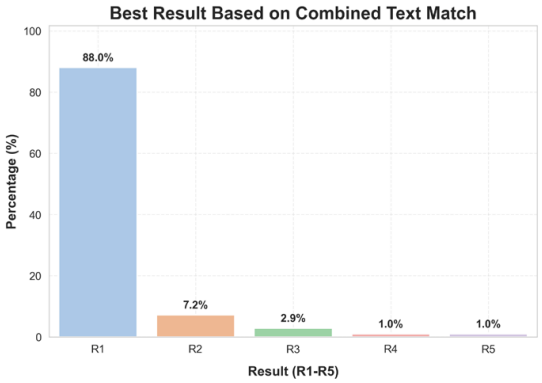
3.7 Second Dataset - Reranking

Below are figures showing the effect of using reranking. Figures to the left are from the original, while the figures on the right side are the reranking results. Results are from the following row:

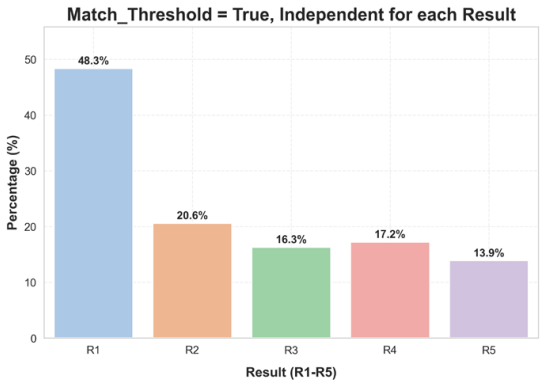
- Chunk size: 512
- Embedding Model: multilingual-e5-large
- Notes: With added context

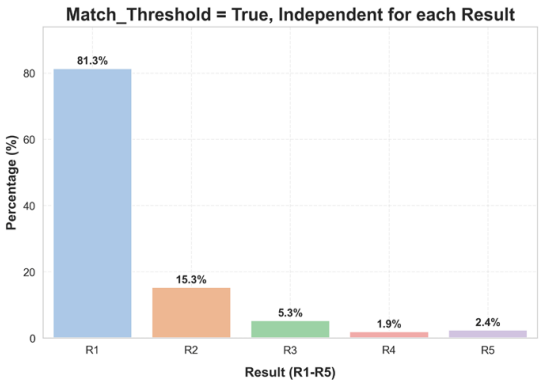
The first figures describe which retrieved result (out of five total results) has the highest text match, per query. We see that there is less spread after reranking.





The figures below show independent statistics for each retrieved result, this means that the columns can add up to more than 100%. The value being measured is "Match_Threshold", which is true or false depending on if the text match met the set threshold. So the bars indicate how often each retrieved result has a match that meets the threshold. In both cases, the threshold was set to 50% combined text match.





3.8 Second Dataset - Generating LLM Response

Since the dataset was small, it was possible to generate LLM responses for each query. However, inspecting the results was done manually, since there was no method to programmatically check the validity of the answers. Manual inspection did show that the LLM managed to extract the correct information, on most (but not all) occasions that it had a text match above the match threshold.

Example of LLM responses:

```
-----------------------------------------------
DC_4491_003, File_Match: True, Page_Match: True, Match_Threshold: True.

Question: Hur ofta får jag bespruta med Mavrik på sockerbetor?

LLM_ANS: Max 2 behandlingar per år, med minst 10 dagars mellanrum mellan
varje behandling och minst 14 dagar mellan sista behandlingen och skörden.

Expected_Answer_Source: |Socker-, foder- och rödbetor||Mot betfluga (Pegomya hyoscyami)
och betbladlus (Aphis fabae)|Bomspruta|BBCH 09-39|2 per år|10|14|0,2 L/ha|
|48 g tau- fluvalinat/ha|||Antalet behand- lingar per år med denna produkt får
aldrig vara fler än två, oavsett om du följer produktens etikett och
bruksanvisning eller användnings- villkoren i det utvidgade||

-----------------------------------------------

DC_4491_004, File_Match: True, Page_Match: True, Match_Threshold: True.

Question: Vad är syftet med att skaka Mavrik före användningen?

LLM_ANS: Syftet med att skaka Mavrik före användning är att produkten
ska vara homogen vid tillredning av brukslösning.

Expected_Answer_Source: Villkoret är till för att produkten
ska vara homogen vid tillredning av brukslösning

-----------------------------------------------
```

4 Analysis

The analysis will be broken down into three segments. First we go through the results for the original dataset, followed by the second dataset. The last segment will bring up recommended future research, based on what we did not have time to investigate or optimize within the project timeframe.

4.1 Original dataset

Analysis of the results for the original dataset. Some of the findings will differ between the datasets, due to the differences in document variation, structure, and content.

Parsing
When inspecting results from different parsers, PyMuPDF performed best when it comes to parsing straight to text. However, for complicated files (multicolumn or tables) it seems that converting to markdown could be beneficial. The original dataset contained mostly "kind" PDF files, with just a few challenging ones to see some initial results for them as well. This means that the complicated documents do not affect the results impactfully.

Visually, Mistral OCR produced the best looking markdown files and was the only parser to manage latex formula encoding gracefully. This was, however, not reflected in the results. They all gave similar results, with PyMuPDF4llm performing best. Not much time was given to optimize the conversion to markdown and information extraction. Parsers were used with standard attributes set. No attempt to annotate or describe images was done when parsing, which was possible with most libraries and could help performance as well. For this dataset, markdown conversion with our settings did not improve overall results.

Chunking
When it comes to chunk sizing, a size of 128 tokens outperforms the larger sizes in the original dataset, we can also see that the recursive character splitting does give better results than regular character splitting. This fits the findings of the technical report from ChromaDB [1].

The most gain from using recursive character splitting is given to chunk accuracy. This could be because more paragraphs are kept intact, improving the chances of a larger text match. The fact that page accuracy is not affected as much could support this. If a paragraph is broken in two, both resulting chunks could give a short distance when calculating similarities with the query, increasing chances of finding the correct page compared to the chances of having a text match above the match threshold.

A reason why smaller chunk sizes work better, could be a simple matter of size. The queries are short questions, while the chunks are parts of, or full, paragraphs. Additionally, smaller chunk sizes allows for files and pages to be broken down into several pieces, giving them multiple representations.

One common issue with RAG is the removal of context, a chunk by itself might not be enough to understand its context or information. Therefore, adding context could probably increase results, especially for shorter chunk sizes since a lot of information is lost. After some initial attempts with local models, it was decided that we did not have enough compute power or time to test it at that point in the project. Another helpful tool is metadata filtering, which was not utilized either.

Overlap
Overlapping does not seem to reliably enhance the results. However, this might be expected in the case of recursive character splitting, since the chunks are already segmented in a particular way. The overlapping breaks the form created that is thought to be beneficial in recursive character splitting.

Normalizing
Normalizing the parsed text before embedding and inserting into the database seems to give worse results. However, normalizing the returned chunks at query does help with text matching. The raw parsed text includes many extra spaces, line breaks, and invisible control characters specific to PDF files. Intuitively, one wants to send clean and normalized text down the pipeline. However, this might remove some non-visual information that we as humans might deem unnecessary. This could be a reason for the worse results when normalizing at parse. Normalizing the returned chunks was however necessary, since the chunk was compared to a clean string without extra characters. This does not affect the retrieval, since it is done after.

Embedding models
The embedding Model, multilingual-e5-large, performed on par with OpenAI's large embedding model, text-embedding-large. Both were top performers, together with the smaller model, text-embedding-small. It seems that it is worth it to use the open-source alternative, multilingual-e5-large, when compared to OpenAI's models. Jina's embedding model, jina-embeddings-v3, was only behind on chunk accuracy, but was equal on file and page accuracy. The other embedding models had lower performance. All good performing embedding models had an embedding vector size of at least 1024 tokens. Multilingual-e5-large has an embedding vector size of 1024 tokens, but it performed a bit better than OpenAI's text-embedding-small which has an embedding vector size of 1536 tokens.

Possible errors
We can see in Table 1, rows 3-4, that the accuracy for chunks is higher than page accuracy. This means that the match threshold may be a bit too low, allowing for false positives. However, the correlation between page match and chunk match is to be expected if an embedding model is performing well. Additionally, some toml files include questions with short very answers, which could produce false positives in the shrinking text matching process. Some files also include questions with a context which is not specific enough, making it almost impossible to find the correct answer. The questions should be reviewed before further usage.

Chunking script needs to be fixed, when counting tokens with smaller models.

4.2 Secondary Dataset

Due to the fact that the second dataset only consists of documents for 20 products and 200 questions, the margin of error is larger and certainty is lower.

Parsing
For this dataset, it is hard to confidently measure chunk accuracy. It becomes very dependent on the quality of the normalization and answer formatting, due to the parser not reliably parsing a table the same in different files. So the actual string matching with the saved answer is the challenge to overcome. An improved way to save answers or generate text matches should be employed in future research.

Converting to markdown gave better results, especially for chunk accuracy. Some of this could be due to changing the answer format to markdown as well. However, some was gained in file and page accuracy as well.

Chunking
Opposite to the first dataset, larger chunk sizes worked better. The larger chunk sizes could contain several pages in one chunk, sometimes all pages. This meant that the full page context was given to each chunk. Each page has a header with meta information about the document, which includes registration number, product name, and relevant dates. Since all this gets included, the context loss in the chunks are minimized. This is supported by the fact that lower chunk sizes perform much better when adding the context of registration number and product name. An attempt to add tags around the tables was made, for testing and to try out other ways of

chunk matching. However, this did not improve any results and was abandoned after a few trials.

Normalizing Normalizing at text match is important if chunk accuracy is to be reliable, otherwise it might increase the number of false negatives. More time needs to be spent to test various strategies here. The answer length should also be at least a sentence long, to help with matching.

Embedding models
The OpenAI embedding models performed equally well in this task. This time, both multilingual-e5-large and jina-embeddings-v3 performed equally as well.

Reranking
Reranking works well, especially on results which are worse from the start. Reranking results which are already good (matches mostly in result 1 and result 2) will not improve the results, and sometimes even yield worse results.

Possible errors
Too much depends on the quality of the normalization, to use chunk accuracy reliably. Page accuracy should be the most granular accuracy check for this.

4.3 Future considerations

1. Hybrid Retrieval
Combining semantic search with traditional keyword-based approaches

2. Query Expansion
Automatically expanding queries with related terms to improve recall

3. Cross-lingual Support
Extending the system to handle multilingual documents and queries

4. Real-time Processing
Optimizing the system for real-time document processing and querying

5. Advanced Evaluation Metrics
Incorporating more sophisticated metrics such as nDCG and MRR

6. SQL Database Integration with MCP Tool

6.1 Data Ingestion and Parsing : This process begins with the ingestion of documents into the system. A parsing component would be responsible for extracting key information from these documents. This could include metadata like Datum, Aktnr, Regnr, Diarienr, Produktnamn, Växtskyddsmedel, Funktion, Sökt användningsområde, Verksamma ämnen, Referensprodukt, Referensprodukt regnr, Beslut, Giltighet start, Giltighet slut, Godkännande, Klassificering, Märkning as well as specific entities and relationships identified through named entity recognition (NER) or other information extraction techniques. This parsed data would then be structured and stored in a relational SQL database.

6.2 MCP Tool for Querying
The MCP tool would serve as a powerful interface for interacting with this structured data. Instead of relying solely on the textual content of the documents, the system could translate a user's query into a SQL query. This allows for highly specific, structured searches that are difficult to achieve with pure semantic or keyword-based methods.
For example, a user might ask: "Vilka produkter blev godkända den här månaden för potatis" The MCP tool would convert this natural language query into a SQL statement.

6.3 Advantages of This Approach

- **Precision:** SQL queries allow for highly precise searches based on specific attributes and conditions, reducing irrelevant results.
- **Structured Information Retrieval:** It enables the retrieval of specific data points (e.g., all approved products for this date) rather than just entire documents.
- **Filtering and Aggregation:** The system could support complex filtering, sorting, and aggregation operations directly on the structured data, offering powerful analytical capabilities.

By integrating an MCP tool with a SQL database, the research system moves beyond simple document retrieval to become a sophisticated information management and analytical platform. It provides a powerful complement to the other future considerations, such as hybrid retrieval and query expansion, by offering a structured and precise method for accessing the information within the documents.

References

[1] ChromaDB:
https://research.trychroma.com/evaluating-chunking

[2] Anthropic:
https://www.anthropic.com/news/contextual-retrieval