TECHNISCHE HOCHSCHULE LÜBECK

Master-Thesis

Vergleichende Analyse von Retrieval-Augmented Generation (RAG) Methoden: Embedding- vs. Volltext-basierte Ansätze

Submitted by:	Konrad Langenberg
Department:	Elektrotechnik und Informatik
Degree program:	Informatik / Softwaretechnik für verteilte Systeme
First examiner:	Prof. Dr.-Ing. Max Zimmermann

Issue Date:	14th July 2025
Submission Date:	14th January 2026

Task description

Retrieval-Augmented Generation (RAG) hat sich als vielversprechender Ansatz zur Verbesserung der Leistung von Large Language Models (LLMs) durch Integration externer Wissensquellen etabliert. Klassische RAG-Implementierungen nutzen dafür Vektor-Datenbanken und Embeddings für den semantischen Zusammenhang von Frage und Antwort, um Inhalte möglicher Antworten im zweiten Schritt an ein LLM zur Generierung der Antwort zu übergeben. Die Erstellung dieser Embeddings ist im Vergleich zu klassischen Datenbank-Indizes relativ aufwändig und ressourcenintensiv. Das Verfahren hat sich aber in der Praxis durchgesetzt, da es wesentlich effizienter als Finetuning ist.

Alternative Ansätze, wie die Nutzung von etablierten Methoden der Volltext-Suche, könnten diesen Prozess vereinfachen. Bislang fehlen jedoch eine systematische Untersuchung und ein Vergleich dieser unterschiedlichen Ansätze.

In der Masterarbeit soll die Hypothese untersucht werden, dass Volltext-such-basierte RAG-Ansätze in bestimmten Anwendungsfällen eine vergleichbare Leistung zu Embedding-basierten Methoden erzielen können, bei gleichzeitig geringerem Implementierungsaufwand.

Zur Beantwortung der Hypothese soll ein Benchmark, der verschiedene RAG-Varianten (e.g. mit und ohne Embeddings, verschiedene Suchstrategien etc.) vergleicht, durchgeführt werden. Die unterschiedlichen RAG-Implementierung werden gezielt evaluiert und anhand geeigneter Kriterien ausgewertet.

TECHNISCHE HOCHSCHULE LÜBECK

Eigenständigkeitserklärung

Declaration of Originality

Langenberg, Konrad | 367536

Name, Vorname
Last name, first name

Matrikelnummer
Matriculation number

Ich versichere hiermit, dass ich die vorliegende
I hereby declare that this

☐ **Hausarbeit** term paper ☐ **Bachelorarbeit** bachelor's thesis ☒ **Masterarbeit** master's thesis

mit dem Titel
with the title

Vergleichende Analyse von Retrieval-Augmented Generation (RAG) Methoden

eigenständig und ohne unerlaubte fremde Hilfe angefertigt habe. Ich habe keine anderen als die angegebenen Quellen und Hilfsmittel verwendet und Entlehnungen aus anderen Arbeiten kenntlich gemacht. Für den Fall, dass die Arbeit zusätzlich elektronisch und/ oder digital eingereicht wird, erkläre ich, dass die schriftliche und die elektronische und/ oder digitale Form identisch sind. Die Arbeit hat in gleicher oder ähnlicher Form noch keiner Prüfungsbehörde vorgelegen.

is my own original work and any assistance from third parties has been acknowledged. I have clearly indicated and acknowledged all sources and resources as well as any borrowings from other works. In case of an additional electronic and/or digital submission of this work, I declare that the written form and the electronic and/or digital form are identical. This work has not previously been submitted either in the same or in a similar form to another examination office.

Ich bin damit einverstanden, dass die vorliegende Hausarbeit/ Bachelorarbeit/ Masterarbeit für Veröffentlichungen, Ausstellungen und Wettbewerbe des Fachbereiches verwendet und Dritten zur Einsichtnahme vorgelegt werden kann.

I agree that this work can be used for publishing, exhibition or competition purposes and can be inspected by third parties.

☒ **ja** Yes ☐ **nein** no ☐ **es liegt ein Sperrvermerk bis _____vor** there is an embargo period until

Lübeck, 11.01.2026

Ort, Datum
Place, Date

K. Langenberg

Unterschrift
Signature

Erklärung zur KI-Nutzung

Bei der Anfertigung dieser Masterarbeit kamen generative KI-Werkzeuge zum Einsatz. Die verwendeten Tools umfassen ChatGPT (OpenAI), Claude (Anthropic) sowie Gemini (Google). Die Nutzung erfolgte ausnahmslos im Bereich der Textoptimierung: Verbesserung von Formulierungen, Prüfung von Grammatik und Rechtschreibung sowie stilistische Anpassungen, ebenso die Erstellung von Programmcode. Die wissenschaftlichen Inhalte, Fragestellungen, Analysen und Erkenntnisse dieser Arbeit sind vollständig meine eigene geistige Arbeit.

Abstract of the thesis

Department:	Elektrotechnik und Informatik
Degree program:	Informatik / Softwaretechnik für verteilte Systeme
Subject:	Vergleichende Analyse von Retrieval-Augmented Generation (RAG) Methoden: Embedding- vs. Volltext-basierte Ansätze
Abstract:	This thesis investigates whether full-text search can serve as a viable alternative to embedding-based approaches in Retrieval Augmented Generation (RAG) systems. The primary motivation stems from the computational and operational overhead of generating and maintaining embedding indices, whereas full-text search leverages established indexing technology. A comparative experiment was conducted using 6,284 questions from 11 datasets across four search engines (pgVector, BM25, Meilisearch, Typesense), each tested with different query preprocessing strategies including query rewriting and keyword generation. An automated Model-as-a-Judge approach rated responses against known correct answers. The results demonstrate that full-text search can outperform embedding-based retrieval when combined with query reformulation. The best full-text configuration achieved 40.44% correctness compared to 31.19% for the best embedding-based approach, though performance varied considerably across datasets. These findings suggest that full-text search with appropriate query preprocessing constitutes a practical alternative to embedding-based RAG, offering reduced complexity and improved retrieval accuracy. Results may differ with other model configurations.
Author:	Konrad Langenberg
Supervising professor:	Prof. Dr.-Ing. Max Zimmermann
WS / SS:	Wintersemester 2025

Contents

1 Introduction

Large Language Models (LLMs) are trained on text corpora collected up to a specific cutoff date, which means they lack knowledge about events and information that occurred after this cutoff date. To extend the knowledge of LLMs with new information or domain-specific knowledge outside of the trained parametric memory, Retrieval Augmented Generation (RAG) has become the established approach to incorporate this knowledge into LLM responses.

RAG works by performing a search in a database based on the user's query to find relevant documents that might contain the answer to the question. These documents are then provided to the LLM, which generates an answer using the retrieved information. Research has shown that this approach performs significantly better than alternatives like fine-tuning, where additional knowledge is added to the model by retraining parts of it. RAG is also more flexible since only the knowledge corpus in the database needs to be updated, rather than fine-tuning an entire model each time. This is particularly important for data that changes regularly and needs to be updated frequently in the LLM.

However, the retrieval pipeline[1] introduces considerable complexity to make a RAG system work reliably. Today, vector embeddings are most commonly used in practice for retrieval, because of their ability to search for semantically similar documents based on a user query. Using an embeddings-based approach requires maintaining a pipeline to keep these embeddings up-to-date. This means developers face mostly classical engineering challenges around large-scale data and database management, rather than AI-specific problems like training models.

This raises the question: can RAG systems work effectively without using embeddings?

After all, the core task is just finding the right documents, which should be possible without embeddings. Full-text search has existed for a long time and is supported by many databases. It naturally comes to mind when thinking about retrieving documents based on a user query.

While databases need to perform indexing to enable full-text search, which is similar to having to maintain embeddings, the overall complexity is much lower. A database optimized for full-text search handles indexing and the creation of search indexes all by itself. This makes using these databases much less complex compared to building an embeddings pipeline where embeddings need to be kept up to date.

An embeddings pipeline usually consists of one data store where the actual content is located in human-readable text and another one where the embeddings are stored. The text content (not the embedding vectors) needs to be stored to pass it to the LLM to generate the response. To make the content searchable via text embeddings, the content has to be processed. First, it needs to be divided into useful chunks[2], then the embeddings need to be generated for them.

[1] The component responsible for finding the right documents from the database.

[2] Because embedding vectors can only capture a limited amount of tokens, text needs to be split into multiple smaller chunks to not overwhelm the embedding model.

Creation of embeddings uses specialized embedding models. Most of the time these are hosted externally and accessed using APIs, with the providers charging for usage.

Finally, embeddings have to be stored in a format that makes it possible to search through them.

When searching in an embeddings database, embeddings need to be generated first for the input search query. These embeddings are then used to search in the database. Combined, all of these steps increase resource usage and response times of the overall application.

With a full-text database, the system only needs to add content to the database – and this requires the same effort as with embeddings since the content needs to be stored in a human readable format even when using embeddings. To search in the stored document corpus, only a search query is required. This can be the input query directly from the user or a transformed query. The database completely handles the index, developers only access the search functionality. As a result, the overall system is less complex with a full-text database compared to an embeddings-based approach.

Both full-text search and embedding-based approaches require the same preprocessing pipeline: document retrieval, parsing, and database storage. The operational complexity for the preprocessing pipeline remains equivalent across both methods. However, full-text search offers a more straightforward implementation path for indexing content compared to the vector embedding approach.

Best Matching 25 (BM25) [1] has been the standard search algorithm for years, but newer databases like Typesense and Meilisearch now offer full-text search with different ranking algorithms. Since BM25 is the established approach, it's worth investigating whether it should still be used or if alternatives might perform better in a RAG setting.

The core contribution of this thesis is examining whether RAG can be done effectively without embeddings and if it produces good results.

To answer this question, an experiment was designed which performs RAG tasks with different search methods on multiple datasets. To establish baseline results, a None retriever condition was tested in which the LLM has to generate an answer without any retrieved text input, relying only on its internal parametric memory. These datasets are pre-filtered and were constructed in [2] by filtering with the `GPT-4o` LLM from OpenAI to obtain questions that an LLM cannot answer from its parametric memory. Most of these are Question Answering (QA) tasks that aim to mimic typical chat applications. Other use cases like agents are out of scope and are topics for future research.

The remainder of this thesis is structured as follows: First, Section 2 reviews current research approaches on RAG, followed by Section 3 detailing the experimental design and implementation. Section 4 reports the obtained results, followed by Section 5, which interprets the findings and addresses the research question. Finally, Section 6 concludes the thesis with key takeaways.

2 Related Work

Current research in the area of RAG tends to focus on two topics: The improvement of how LLMs understand and process knowledge and methods to improve the retrieval of information for usage in RAG. Since the public release of ChatGPT in late 2022, many papers have been published in this field - it is very much an active area of research.

This chapter describes related work and current research areas to improve RAG. In Section 2.1, the origins of the RAG paradigm are described, followed by a general architecture overview in Section 2.2, leading up to a brief explanation of embeddings in Section 2.3, Section 2.4 explores knowledge injection through fine-tuning and related approaches. Section 2.5 examines research focused on improving the retrieval process. Section 2.6 presents fundamental search algorithms, and finally Section 2.7 discusses approaches for evaluating RAG systems.

2.1 RAG origins

The term RAG was coined in the seminal paper by Lewis et al. in 2021 [3]. They introduced the method as a hybrid approach that combines pre-trained parametric memory embedded in a BART language model at the time, with non-parametric memory (dense vector retrieval of content, in the case of the paper text content from Wikipedia) to improve performance on knowledge-intensive Natural Language Processing (NLP) tasks. The language model, which marginalizes over retrieved documents during generation, achieved state-of-the-art results on open-domain question answering, reduced hallucination compared to purely parametric models, and demonstrated the ability to update knowledge by simply replacing the retrieval index without retraining. Two variants were proposed: RAG-Sequence, which uses the same retrieved document for the entire output sequence, and RAG-Token, which can utilize different documents for each generated token. They showed that this approach hallucinates less while being more factually correct than other approaches. Factual correctness was verified using FEVER, a fact-check benchmark [3].

They also found that the quality of the given response ultimately depends on the quality of the retrieved documents. This is still true today.

Their research introduced a new way to update a model's knowledge without having to re-train it fully or fine-tune parts of it - a process which is expensive and needs a lot of resources.

While many improvements have been made to the way a RAG-System works since the paper was published, modern RAG-Systems look a lot like the one originally proposed.

The different architectures have been presented by Gao et al. in a comprehensive survey of RAG techniques for Large Language Models [4]. They categorize the evolution of the pattern into three paradigms: Naive RAG (basic retrieve-read framework, as outlined in Section 2.2), Advanced RAG (incorporating pre- and post-retrieval optimizations), and Modular RAG (flexible architectures with specialized components).

The analyzed studies demonstrate that RAG effectively mitigates LLM limitations such as hallucination and outdated knowledge while generally outperforming fine-tuning approaches for knowledge-intensive tasks.

RAG presents a way to add new or external knowledge to an LLM without having to retrain or fine-tune the model, which makes these systems very interesting for a variety of use-cases. The most obvious is question-answering systems which can reply to a wide variety of user questions, but also chatbots, creating documents or generating software code.

2.2 Architecture of a RAG system

On a high level, a RAG system consists of two parts:

1. A **retriever** which fetches text content based on a user query
2. A **generator** which uses the retrieved content to generate a response to the query.

Formally, a RAG system can be expressed as follows:

$$D_r = R(D, q)$$

$$a = G(q, D_r)$$

Where R is a retriever function which, given a user query q and a set of documents D, retrieves relevant documents D_r. The relevant documents D_r are then passed to the generator function G along with the original query q to produce the final answer a. The generator is in almost all cases an LLM being called with a special prompt to produce an answer.

Figure 1 shows an overview of what such a system looks like on a high level.

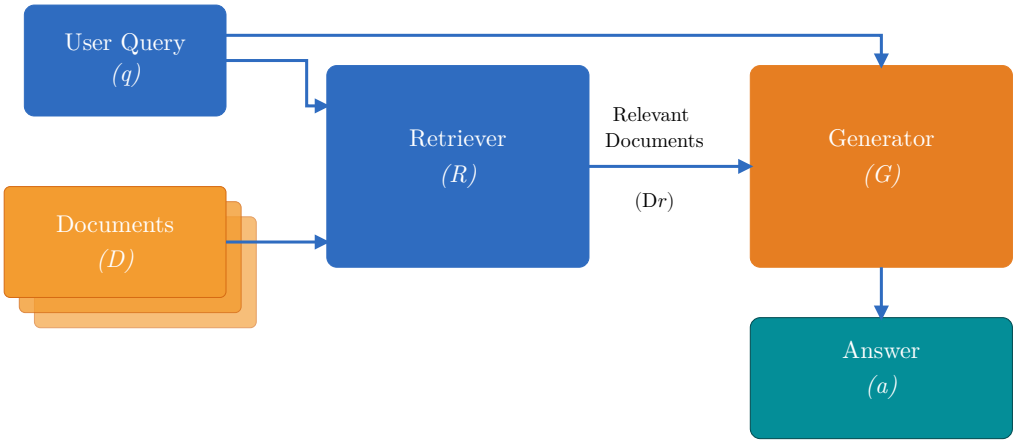

Figure 1: A general definition of a RAG System, combining documents D and a user query q with a retriever R and a generator G to produce an answer to a query a.

The system is opaque to the way the retriever works under the hood. The retriever can be a simple keyword-based search engine, a more complex embedding-based search engine, or a combination of both.

The quality and relevance of the retrieved documents are highly important for the quality of the response to the user query.

Practitioners are expressing this as “If you want to make a good RAG tool [...], you should start by making a search engine over those documents that would be good enough for a human to use themselves.” [5]

Plenty of research is currently being conducted to improve the search process so that the LLM used to generate the response has the most relevant content it needs to give an accurate response to the user query.

2.3 Embeddings

Retrieving content happens mostly by using text embeddings in industry practice today. With embeddings, all text content is first transformed into a high-dimensional vector representation which captures its semantic meaning. These vectors can then be searched for similarity to the input query using cosine distance or similar methods, returning only those pieces of content semantically similar to the input query.

Neelakantan et al. demonstrated that embeddings produced by pre-training on unsupervised data produce high-quality text and code embeddings [6] which can then be used to retrieve text based on semantic similarity.

Nussbaum et al. present nomic-embed-text-v1 [7], the first fully reproducible long-context text embedding model that achieves competitive performance with only 137 million parameters and 8192 token context length. The model employs a three-stage training pipeline using architectural modifications to Bidirectional encoder representations from transformers (BERT). It demonstrates superior performance to OpenAI’s `text-embedding-ada-002` and `text-embedding-3-small` on both short-context (MTEB) and long-context (LoCo) benchmarks. Notably, the authors release all training artifacts including curated datasets, training code, and model weights, addressing the lack of transparency in existing high-performing embedding models.

The same authors also introduce Nomic Embed v2 [8], the first general-purpose Mixture of Experts (MoE) text embedding model, addressing the efficiency challenges of scaling multilingual embedding models. Traditionally, these embedding models require 3-5x more parameters than monolingual counterparts to achieve comparable performance. The model uses an adapted XLM-RoBERTa architecture with 8 experts and top-2 routing, training on 1.6 billion high-quality pairs, resulting in 475M total parameters with 305M active during inference. Experimental results demonstrate that the MoE approach outperforms similarly-sized dense models on both monolingual (BEIR) and multilingual (MIRACL) benchmarks. This work represents a fundamental shift from previous scaling approaches that relied solely on increasing dense model capacity.

While embeddings are very useful for RAG systems, their main downside is the resource intensive process of creating and keeping them up to date since in all cases this needs expensive AI-infrastructure to either host a model or use a provider to access embedding models via an API. When building these systems outside of lab tests, a considerable amount of software

engineering has to be done to build the overall system in a way that ensures information is kept up to date and embeddings are created for all text in the system.

2.4 Injecting knowledge into an LLM through fine-tuning and related approaches

Before the original RAG paper was published, Lauscher et al. [9] investigated adapter-based knowledge injection into BERT using bottleneck adapters rather than full model fine-tuning to avoid catastrophic forgetting[3] of distributional knowledge.

While overall GLUE benchmark results showed limited improvements, the trained models demonstrated substantial performance gains on inference tasks requiring factual world knowledge and named entity information, but performed worse on tasks which required common sense reasoning. The authors concluded that explicit knowledge injection is effective for factual information but insufficient for complex reasoning tasks.

Similarly, Wang et al. proposed K-ADAPTER [11], a framework for injecting knowledge into pre-trained language models by keeping the original model parameters frozen and training compact knowledge-specific adapters independently. This improves upon previous methods where the resulting model would lose previously learned knowledge during fine-tuning.

Newer research by Ovadia et al. systematically compared unsupervised fine-tuning and RAG for knowledge injection in LLMs [12]. They evaluated three 7B-parameter models across MMLU benchmark tasks and a custom current events dataset containing information beyond the models' training cutoff[4]. Their findings demonstrate that RAG consistently outperforms fine-tuning for both previously encountered and entirely new knowledge. The authors attribute this superiority to RAG's ability to provide relevant context alongside factual information while avoiding the catastrophic forgetting that can occur during fine-tuning. The study also reveals that LLMs struggle to internalize new factual information through unsupervised fine-tuning alone, though exposing models to multiple paraphrased variations of the same facts during training shows promise for improving knowledge retention.

Combining both approaches, Zhang et al. [13] propose RAFT (Retrieval-Augmented Fine Tuning), a new training strategy that fine-tunes LLMs for domain-specific RAG. The method trains models on question-answer pairs where some training instances include "golden" documents which contain the answer along with distractor documents, while others contain only distractors, teaching the model which documents from a retrieval set are relevant for answering a question. RAFT then generates chain-of-thought style answers with verbatim citations from relevant documents. Experiments across different benchmarks demonstrate that RAFT consistently outperforms standard supervised fine-tuning approaches.

[3] Catastrophic forgetting is a phenomenon where an LLM forgets factual information it was originally trained on when it is fine-tuned for other tasks. [10]

[4] The training cutoff is the date when no more training data was collected and model training was started. Without relying on external sources, LLMs don't have any knowledge about events that happened beyond that training cutoff date.

Borgeaud et al. introduced RETRO (Retrieval-Enhanced Transformer) [14], a semi-parametric language model that conditions on document chunks retrieved from a 2 trillion token database using frozen BERT embeddings and a chunked cross-attention mechanism. This approach is similar to RAG in that it uses a retrieval mechanism to augment a language model, but it integrates retrieval directly into the transformer architecture rather than as a separate step. In comparison with RAG, RETRO slightly outperforms it (45.5 accuracy of RETRO vs 44.5 accuracy of RAG).

Despite using 25x fewer parameters than GPT-3, RETRO achieves comparable performance on benchmarks like the Pile and Wikitext103, demonstrating that retrieval from massive-scale databases can effectively decouple computational scaling from model memorization. The approach shows consistent improvements across model sizes (150M-7B parameters) and can be retrofitted to existing models, while also reducing hallucinations and improving factual accuracy compared to purely parametric models.

In summary, the research results presented in this section indicate that it does not make much sense to pursue fine-tuning as a viable alternative to RAG, since it is less flexible and yields worse results than RAG. Even though Lauscher et al. and K-ADAPTER demonstrated better performance of their fine-tuning approach, they only compared a fine-tuned model with a vanilla BERT model, whereas current state-of-the-art LLMs are more complex and have significantly more parameters.

2.5 Improving the Retrieval Process

As established in earlier sections, to achieve good results with a RAG system, the quality and relevance of the retrieved documents are most important for the quality of the overall response. Therefore, it makes sense to focus on improving the retrieval part of a RAG system. This section and its subsections explore current research in this area.

Anthropic have proposed a way to enrich RAG-content by prepending chunk-specific explanatory content to document chunks before creating embeddings. They call this pattern Contextual Retrieval [15]. This addresses the problem of chunks lacking sufficient context for accurate retrieval.

The method automatically generates contextual information based on the full document using an LLM and reduces top-20 chunk retrieval failure rates by 49% when combining the resulting embeddings with Contextual BM25 (from 5.7% to 2.9%), and achieves a 67% reduction (to 1.9%) when further combined with reranking.

The study demonstrates that combining semantic embeddings with lexical matching (BM25), adding contextual information to chunks, and implementing reranking all contribute additively to retrieval performance improvements across multiple knowledge domains.

2.5.1 Tool use

Tool use refers to the capability of LLMs to invoke external functions that extend their base functionality. State-of-the-art LLMs are trained to identify appropriate tools for a given task and generate structured function calls accordingly. If the LLM wants to use a tool, it outputs a tool invocation request, which the host application executes externally. The tool results are then provided back to the LLM as context, enabling it to generate a final response that incorporates the result of the function execution.

Toolformer [16] introduces this pattern in a self-supervised approach for training language models to use external tools through API calls. The method uses in-context learning before fine-tuning the model on a dataset filtered by those API calls. Experimental results show that the resulting model, based on a 6.7B parameter GPT-J model, significantly outperforms baseline models and even much larger models like GPT-3 on mathematical reasoning, factual knowledge retrieval, and multilingual tasks while preserving core language modeling capabilities. The approach demonstrates that language models can learn to autonomously decide when and how to leverage external tools in zero-shot settings, though it is limited to single API calls per input.

Combining reasoning with tool use, Yao et al. introduce ReAct [17], a prompting paradigm that enables large language models to interleave verbal reasoning traces with task-specific actions. The approach combines the benefits of chain-of-thought reasoning with external environment interaction, using a simple Wikipedia API for knowledge-intensive tasks. ReAct consistently outperforms both reasoning-only and action-only baselines while providing enhanced interpretability and reduced hallucination compared to standard chain-of-thought approaches. The method's effectiveness extends to fine-tuning scenarios, where ReAct demonstrates superior performance even with smaller models and limited training data, suggesting its potential as a foundation for integrating reasoning capabilities with external knowledge retrieval.

This shows an interesting direction for infusing new knowledge into existing language models at runtime by providing a search function which can be used like a search engine, making this a worthwhile area to explore for RAG systems.

2.5.2 Using graph data structures

LightRAG [18] addresses limitations of traditional retrieval-augmented generation systems by incorporating graph structures into text indexing and retrieval processes. They use large language models to extract entities and relationships from documents to construct knowledge graphs that capture the dependencies between information sources.

The system employs a dual-level retrieval paradigm which combines low- and high-level retrieval for narrower and broader topics, to better answer specific and abstract queries. Experimental evaluation across multiple datasets from the UltraDomain benchmark demonstrates that LightRAG consistently outperforms baseline RAG methods, particularly on datasets where baseline methods struggle to synthesize information across multiple document sources.

Going in a similar direction, Fatehkia et al. present Tree-RAG (T-RAG) [19], a system that combines Retrieval-Augmented Generation with a fine-tuned Llama-2 7B model and a novel tree-based context component. The tree structure specifically addresses hierarchical entity relationships within organizations, which traditional RAG and knowledge graph approaches handle poorly. In human evaluations on 37 questions from organizational documents, T-RAG achieved 73% correct responses compared to 56.8% for standard RAG and 54.1% for fine-tuning alone, though the limited evaluation scale raises questions about the broader generalizability of these improvements.

Sepasdar et al. introduce Structured-GraphRAG [20], a framework that enhances RAG systems by automatically constructing knowledge graphs from structured datasets, specifically demonstrated using soccer data from the SoccerNet dataset. Unlike traditional GraphRAG approaches that require domain experts to design knowledge graphs, the researchers developed an automated method for transforming structured tabular data into graph representations. The system operates through a four-step process: knowledge graph construction, query translation (converting natural language to Cypher queries), information retrieval from the graph database, and answer generation using GPT-3/GPT-4 models. Evaluation on soccer data showed significant performance improvements over traditional RAG methods, achieving 64% accuracy compared to 36% for baseline approaches, while also demonstrating substantial execution time reductions.

While demonstrated on soccer data, the methodology is designed to be domain-agnostic and applicable to any structured dataset organized in tabular format, though the evaluation was conducted with a limited scope of 10 questions tested across 5 iterations each.

Their study shows that RAG systems dealing with structured data rather than text-only content can benefit from graph-based representations to better capture relationships and dependencies within the data, ultimately improving RAG results.

Extending graph-based RAG from structured tables to unstructured corpora, HippoRAG [21] introduces a neurobiologically inspired retrieval layer that treats a schemaless OpenIE knowledge graph as a hippocampal-style index for long-term memory in LLMs.

In contrast to other graph-based RAG methods, HippoRAG does not need corpus data to be in a graph format, but constructs a knowledge graph from unstructured text using OpenIE. During retrieval, it runs a Personalized PageRank algorithm over the graph to find relevant passages. Empirically, HippoRAG achieves sizable gains on multi-hop QA while being 6-13x faster and 10-30x cheaper than strong retrievers. The approach is unsupervised, incrementally updatable, and particularly effective for “path-finding” queries that require linking dispersed facts.

The presented studies indicate that using graph data structures rather than text-only approaches can yield better results when the underlying data benefits from structured representation and hierarchical relationships are important.

2.5.3 Reducing the need for retrieval

Jiang et al. introduce Forward-Looking Active REtrieval augmented generation, FLARE [22], a method that enhances retrieval-augmented language models by actively deciding when and what to retrieve during generation, addressing the limitations of single-time retrieval approaches in long-form text generation tasks. FLARE iteratively generates a temporary next sentence and uses it as a query to retrieve relevant documents when the model exhibits low confidence, then regenerates the sentence conditioned on the retrieved information. The approach achieves superior performance compared to different retrieval baselines across four diverse long-form generation tasks, demonstrating that forward-looking retrieval queries that anticipate future content significantly outperform past-context-based approaches. The method is applicable to any large language model at inference time without requiring additional training, making it a practical solution for improving RAG systems.

Déjean [23] presents a method for training large language models to determine when RAG is necessary by developing an "I Know" (IK) classifier which predicts whether an LLM can answer questions using only its parametric memory. The method enables a reduction of over 50% in retrieval operations across various question-answering datasets while maintaining or improving answer quality. The work provides empirical evidence that LLMs can be trained to assess their own knowledge limitations, with performance varying significantly across different dataset types and retrieval requirements.

Due to the need to train a model on the proposed behaviour, this approach is suitable only for use cases where the model already has extensive knowledge of the topic the RAG system is being built for. This makes it not suitable for the task explored in this thesis.

Chan et al. propose Cache-Augmented Generation (CAG) [24] as an alternative to RAG for knowledge-intensive tasks, leveraging the extended long context capabilities of modern LLMs. Their approach involves preloading all relevant documents into the LLM's context window and then storing the resulting key-value (KV) cache offline, eliminating retrieval latency and potential retrieval errors inherent in traditional RAG systems. Experiments on SQuAD and HotPotQA benchmarks using Llama-3.1 8B demonstrate that CAG consistently achieves higher BERT Scores than both sparse (BM25) and dense (OpenAI embeddings) RAG baselines.

The authors conclude that, as long as the entire knowledge base fits within the model's context window, CAG outperforms RAG, with the performance gap narrowing as the document collection size increases. This also means the approach is limited to scenarios where the entire knowledge base can fit within the model's context window. For applications with small, constrained knowledge bases such as internal documentation or FAQs, CAG can provide a more effective alternative to RAG, though hybrid approaches combining preloading with selective retrieval may offer optimal solutions for larger-scale applications.

Li et al. conduct an evaluation comparing Long Context (LC) and RAG approaches for LLMs [2]. To assess conflicting findings in prior literature, the study employs a methodology that

filters out questions answerable from parametric knowledge, to make sure that their evaluation focuses on the retrieval and does not benchmark the LLM itself. It evaluates multiple retrieval methods (chunk-based, index-based, and summarization-based), and expands existing datasets to approximately 20,000 questions across 12 QA benchmarks.

Their experiments reveal that LC generally outperforms RAG (56.3% vs 49% accuracy), particularly with well-structured, dense contexts such as Wikipedia articles and narrative texts.

However, RAG demonstrates advantages when handling fragmented information, especially in dialogue-based scenarios and general questions requiring synthesis from multiple sources. Among retrieval methods tested, RAPTOR (a summarization-based approach using hierarchical clustering) achieved the best performance at 38.5% accuracy, outperforming chunk-based and index-based retrievers.

In summary, approaches that reduce retrieval needs have distinct limitations: FLARE enables adaptive retrieval at inference time, the "I Know" classifier requires domain-specific training data, and CAG is constrained by context window size. These methods work best when knowledge is stable and well-represented in the model's training data, limiting applicability to use cases requiring frequent content updates or specialized information.

2.5.4 Building better retrieval methods

Leto et al. investigate optimization strategies for retrieval components in RAG pipelines [25], evaluating systems with two instruction-tuned LLMs and two dense retrieval models across three datasets. Key findings show QA performance plateaus at 5-10 retrieved documents, with gold document[5] recall being more critical than search recall, with more gold documents yielding better results. Notably, the study demonstrates that approximate nearest neighbor search with reduced accuracy provides substantial speed and memory benefits with minimal performance loss.

Contrary to prior work, injecting noisy documents consistently degrades both correctness and citation quality, indicating that retrieval systems should prioritize retrieving relevant gold documents over maximizing retrieval quantity.

Soman and Roychowdhury conducted experimental studies on RAG systems for technical documents [26] using IEEE specifications and battery terminology, finding that sentence embeddings become unreliable with increasing chunk sizes, particularly when queries or documents exceed 200 words. Their key finding was that similarity score thresholding for retrieval augmentation can be unreliable and potentially result in sub-optimal generator performance, while better contextual retrieval (sentence-based similarity with paragraph-level retrieval) and splitting definitions from terms in glossaries improved overall system performance. The authors demonstrated that chunk length significantly affects retriever embeddings and that keyword positioning within sentences influences retrieval accuracy, though they acknowledge the domain-specific nature of their telecom-focused findings may limit generalizability.

[5] A document which contains the ground truth to a given question.

Weller et al. introduce Promptriever [27], the first retrieval model capable of being prompted like language models to dynamically adjust relevance criteria on a per-query basis. Using LLaMA as a backbone, the authors train a Bi-encoder on a curated dataset of ~500k MS MARCO instances augmented with instance-level natural language instructions and "instruction negatives" - cases where query-passage pairs become less relevant when specific instructions are added.

This approach would be used in the retrieval step in a RAG pipeline where a user query is transformed to a vector embedding for search in a vector database. Instead of a generic embedding model, Promptriever produces embeddings conditioned on the input instruction, enabling users to specify detailed relevance criteria (e.g., "movies before 2022 that are not co-directed") without requiring traditional filters or reranking approaches.

Promptriever achieves state-of-the-art performance on instruction-following retrieval benchmarks while maintaining competitive standard retrieval performance, and demonstrates the ability to reliably improve retrieval through zero-shot prompting.

These studies demonstrate that QA performance plateaus at 5 to 10 retrieved documents, with gold document recall being more critical than overall search recall, and that chunk sizes should not exceed 200 words to maintain embedding reliability. Since retrieving relevant documents proves more important than maximizing retrieval quantity, reranking techniques (Section 2.5.5) seem promising to ensure the limited number of documents provided to the generator are of highest relevance.

2.5.5 Reranking

Reranking is the process of ranking documents for relevance compared to a user query. In RAG, this step is performed after retrieving documents from the retriever, refining the document search results before passing them to the generator. Formally, this can be expressed as an additional step or as part of the retrieval component.

Yu et al. [28] present RankRAG, an instruction-tuned LLM which can rank relevant documents and provide the answer to a user query based on the top-k reranked documents. The approach uses a two-stage training process that unifies ranking and generation tasks into a standardized question-context-answer format, enabling effective knowledge transfer across tasks. During inference, RankRAG adds an additional ranking step to traditional RAG pipelines, where the model first reranks retrieved contexts and then generates answers using the top-ranked passages.

Experimental results using Llama3 8B and 70B models demonstrate that RankRAG significantly outperforms existing RAG methods. Additionally, the method shows strong generalization capabilities, achieving comparable performance to GPT-4 on biomedical benchmarks without domain-specific training, suggesting that the dual ranking and generation capabilities mutually enhance each other in RAG systems.

Yan et al. propose Corrective Retrieval Augmented Generation (CRAG) [29], a plug-and-play framework that addresses the robustness issues in RAG systems when retrieval quality is poor. The approach employs a lightweight T5-based retrieval evaluator to assess document relevance and triggers three corrective actions: knowledge refinement for relevant documents, web search fallback for irrelevant retrievals, and a hybrid approach for ambiguous cases. Experimental results demonstrate significant performance improvements over standard RAG and Self-RAG, while maintaining minimal computational overhead.

Going in a similar direction, Zhang et al. present mGTE [30], a framework for building long-context multilingual text representation and reranking models. The system combines a hybrid text representation model capable of generating both dense and sparse vectors with a cross-encoder reranker, both trained on a large-scale multilingual dataset. Evaluation results demonstrate that their base-sized encoder outperforms the previous state-of-the-art XLM-R on natural language understanding benchmarks, while their retrieval models match the performance of larger BGE-M3 models and achieve superior results on long-context retrieval tasks.

Blagojevic introduces two novel ranking components for enhancing RAG pipelines in the Haystack framework [31]: DiversityRanker, which uses sentence transformers and a greedy algorithm to select semantically diverse documents from a relevance-filtered pool, and LostInTheMiddleRanker, which mitigates the lost in the middle problem[6] by positioning the most relevant documents at the beginning and end of the LLM's context window. Both components were evaluated on long-form question answering tasks and found to achieve a 20-30% increase in average pairwise cosine distance between context documents compared to baseline pipelines. However, the evaluation methodology primarily relied on diversity metrics rather than comprehensive answer quality assessment, limiting the conclusions about overall RAG performance improvements.

Proposing a more integrated approach, Asai et al. introduce SELF-RAG [32], a framework which reviews and critiques retrieved documents before using them to generate a response. They train an LLM to output retrieval tokens to trigger a retrieval model and critique tokens to evaluate the output and choose the best sources for answer generation.

For training, they distill GPT-4 feedback into a critic that labels training data with Retrieve/ ISREL/ISSUP/ISUSE tokens; the generator is then trained to predict both outputs and these tokens. At test time, only the generator and an external retriever are needed.

Across six tasks, SELF-RAG outperforms instruction-tuned and RAG baselines. Limitations include occasional unsupported generations despite citations, dependence on off-the-shelf retrieval and corpus choices, and sensitivity to training data scale. Human evaluations report good alignment of reflection tokens with annotator judgments.

[6]The lost in the middle problem is a phenomenon where LLMs when given a list of documents, seem to prioritize those documents at the start and end of the prompt, losing the information in the middle.

Similarly, Xia et al. propose Self-Reasoning [33], an end-to-end framework that enhances Retrieval-Augmented Language Models by incorporating self-generated reasoning trajectories through three processes: relevance assessment, evidence selection with citation, and trajectory synthesis. The framework trains LLMs to internally evaluate and filter retrieved documents without external tools, requiring only 2,000 training samples compared to 46,000 for competing methods like SELF-RAG. Evaluated on various datasets, the approach demonstrates superior performance, particularly in fact verification tasks, while improving both reliability through better handling of noisy retrievals and traceability through explicit citation generation.

Li et al. conducted a comprehensive evaluation of RAG system components [34] through 74 experiments across nine research questions, using TruthfulQA and MMLU datasets to assess performance variations across different RAG setups. The study introduced and tested several advanced RAG designs. Query expansion, where the input query is expanded into multiple keyword phrases relevant to answer the query, Contrastive In-Context Learning which includes correct and incorrect examples from the evaluation data as the knowledge base, and Focus Mode that performs sentence-level retrieval and ranking.

Results demonstrated that Contrastive In-Context Learning achieved the strongest performance improvements, significantly outperforming baseline RAG systems, while Focus Mode ranked second by prioritizing precise, relevant context over comprehensive coverage. Contrary to common assumptions, the study found that knowledge base size and document chunk variations had minimal impact on performance, with context quality and relevance proving more critical than quantity.

The focus mode shows that reranking is a valid approach to improve RAG generation results. Query expansion seems like a promising way to improve full-text search for keywords in the context of this thesis.

The approaches range from training-intensive methods like SELF-RAG to lightweight alternatives like Self-Reasoning and plug-and-play solutions like CRAG. Hybrid approaches combining dense and sparse vectors with cross-encoder reranking, such as mGTE, demonstrate superior performance across different task types.

Reranking in general presents an interesting direction for improving RAG systems when the retrieval process itself fails to return good results. This could be the case when the initial retrieval returns suboptimal results. Even when not using web search, as is often the case with QA systems used in practice, refining search results before the generation process seems to be a promising direction.

In CRAG's web search approach, they are using an LLM to pick the keywords to search for, a solution which could be applicable for searching using traditional full-text search as well. Interestingly, they are only using 10 documents - when having a reranker system to rate retrieved documents, it would be possible to retrieve 100 documents, rank them and then use the top 10 among the 100 ranked.

Using these approaches from the different methods provides a promising direction for the topics in this thesis.

2.5.6 Iteratively improving RAG results

More recently, Xi et al. present OmniThink [35], a machine writing framework that emulates human-like iterative research processes by continuously expanding on retrieved information through alternating expansion and reflection cycles. The method introduces an Information Tree that hierarchically organizes retrieved information and a Conceptual Pool that distills insights to progressively expand both information and cognition boundaries during the writing process. Experimental results on the WildSeek dataset demonstrate superior performance over existing methods across metrics of relevance, breadth, depth, and novelty, with the authors introducing a new Knowledge Density metric to measure the ratio of meaningful content to total text volume.

Similar to CRAG, SEAKR (Self-aware Knowledge Retrieval) [36] introduces an adaptive retrieval-augmented generation approach that leverages the internal states of large language models to dynamically determine when to retrieve external knowledge and how to integrate it. The method extracts self-aware uncertainty by computing the Gram determinant and using it to trigger retrieval when thresholds are exceeded. SEAKR incorporates three adaptive mechanisms: self-aware retrieval for deciding when to search, self-aware re-ranking for selecting the most uncertainty-reducing knowledge snippets from retrieved candidates, and self-aware reasoning for choosing between different synthesis strategies. Their approach achieved substantial improvements over existing adaptive RAG methods, with ablation studies revealing that dynamic knowledge integration strategies contributed more to performance gains than the retrieval decision mechanism alone. The tuning-free approach demonstrates better generalization across tasks compared to fine-tuned alternatives, though it requires access to model internal states and incurs computational overhead from multiple generation sampling.

While these iterative approaches show promise, OmniThink's multi-cycle process is primarily suited for long-form content generation, and SEAKR's multiple generation sampling incurs computational overhead that may limit use in latency-sensitive QA scenarios.

2.6 Search approaches and algorithms

Robertson and Zaragoza present a theoretical exposition of the Probabilistic Relevance Framework (PRF) [37], which provides the formal foundation for BM25, one of the most successful document retrieval algorithms in information retrieval. The framework models document relevance as a hidden probabilistic variable, enabling systems to rank documents by their estimated probability of relevance to a given query through a principled mathematical derivation. The paper extends the basic BM25 algorithm to BM25F, which incorporates document structure and metadata (such as titles, abstracts, and anchor text) through weighted field combinations, making it particularly effective for web search and structured document collections.

Singh et al. [38] define and systematize Agentic RAG, asking how embedding autonomous agents (reflection, planning, tool use, multi-agent collaboration) extends traditional RAG to enable adaptive retrieval, iterative refinement, and complex task orchestration. They contribute a taxonomy of architectures and workflow patterns, alongside a comparative analysis with prior RAG paradigms. The paper also surveys tools and frameworks and synthesizes applications across different domains.

Key conclusions highlight benefits in contextual precision and scalability via agentic orchestration, while noting challenges in coordination complexity, latency, and ethical deployment.

Generally, the idea to compose multiple LLMs with tools to improve RAG seems promising.

Anderson et al. present Lingua [39], a speech-to-speech interpretation system that addresses error propagation issues in cascaded ASR-MT-TTS pipelines by incorporating pre-existing speech scripts. The system uses a fuzzy matching algorithm based on Levenshtein distance to align real-time Automatic Speech Recognition (ASR) transcriptions with script sentences at the phonemic level, achieving F1 scores above 0.95 with an average lag of 0.72 seconds. This approach significantly improves translation accuracy while reducing latency compared to traditional cascaded systems, making it particularly suitable for live speech interpretation scenarios where scripts are available in advance.

As this approach covers speech-to-speech, it is not directly relevant for the topic of this thesis, but demonstrates a simple algorithm to find matches between the transcript and manuscript, which can be useful for general text-matching for RAG.

2.7 Benchmarking RAG systems

To find out if the performance of a RAG system is optimal, various benchmarks exist.

Friel et al. introduced RAGBench [40], a comprehensive benchmark dataset comprising 100k examples across five industry domains for evaluating Retrieval-Augmented Generation systems, addressing the lack of standardized evaluation criteria in the field. The authors developed the TRACe evaluation framework, which measures four key metrics: utilization, relevance, adherence, and completeness of RAG system components. Through extensive benchmarking, they demonstrated that fine-tuned specialized models (DeBERTa-v3-Large) consistently outperform LLM-based evaluation methods such as GPT-3.5 judges, RAGAS, and TruLens across most evaluation tasks. The study reveals that context relevance estimation presents particular challenges, requiring sophisticated understanding beyond semantic similarity to determine whether retrieved documents contain specific information necessary for accurate question answering.

Fleischer et al. introduce RAGFoundry [41], an open-source framework designed to address the complexity of implementing and evaluating RAG systems. The framework integrates the four key modules: data creation, training, inference, and evaluation into a unified workflow for RAG experimentation and evaluation. The data processing module employs components including loaders, retrievers, samplers, and prompters, while the training module supports

LoRA fine-tuning using the TRL framework. The evaluation module incorporates different local and global metrics for evaluation. The authors demonstrate the framework's effectiveness by fine-tuning Llama-3 and Phi-3 models across three knowledge-intensive question-answering datasets (TriviaQA, PubmedQA, ASQA).

This framework distinguishes itself from production-oriented RAG tools by focusing specifically on academic research needs and comprehensive evaluation capabilities.

Krishna et al. introduce FRAMES [42], a benchmark with 824 multi-hop Wikipedia questions evaluating retrieval-augmented generation across factuality, retrieval, and reasoning. State-of-the-art LLMs score 0.408 without retrieval, improve modestly with BM25, and reach 0.729 with oracle documents, where the LLM receives all documents used to create the question, simulating perfect retrieval. A multi-step retrieval-and-planning pipeline achieves 0.66, though errors persist in numerical and tabular reasoning. Future work targets stronger retrievers and process-supervised reasoning to close the performance gap.

While this presents an interesting approach, its focus on complex reasoning tasks is somewhat misaligned with the goals of this thesis. Its use of search queries is similar to the intended methodology.

2.8 Conclusion

This chapter looked at current research in RAG, revealing several key insights that inform the direction of this thesis.

The origins of RAG (Section 2.1) established that retrieval quality fundamentally determines system performance, a finding that remains true today. While fine-tuning approaches (Section 2.4) can inject knowledge into language models, research demonstrates that RAG consistently outperforms fine-tuning for knowledge tasks, making it a better approach for adding knowledge to LLMs.

Current research on improving retrieval (Section 2.5) reveals multiple promising directions. Graph-based approaches show benefits for structured data and hierarchical relationships, though they are only suitable when the data has certain characteristics. Methods that reduce retrieval needs by figuring out if the LLM can answer a question from its parametric memory work best when knowledge is stable and well-represented in training data, limiting their use for specialized domain-specific or frequently-updated knowledge.

The most relevant findings center on building better retrieval methods. Research shows that QA performance plateaus at 5-10 retrieved documents, and chunk sizes should not exceed 200 words to maintain embedding reliability. Since retrieving relevant documents proves more important than maximizing retrieval quantity, reranking techniques (Section 2.5.5) are a promising direction to ensure the limited documents provided to the generator are of highest relevance.

Multiple studies demonstrate that combining semantic embeddings with lexical matching (BM25), adding contextual information to chunks, and implementing reranking all contribute additively to retrieval performance. Notably, sparse retrieval methods like BM25 can outperform dense embedding-based approaches in certain scenarios (Section 2.6), suggesting that traditional full-text search algorithms remain competitive and deserve further investigation.

The surveyed benchmarking approaches (Section 2.7) provide standardized evaluation frameworks, with metrics emphasizing context relevance, utilization, and adherence being critical for assessing RAG system quality.

These findings suggest a promising research direction: investigating whether traditional full-text search approaches, enhanced with query expansion and reranking techniques, can achieve competitive performance with embedding-based RAG systems while avoiding the resource intensive process of creating and maintaining embeddings. This forms the central motivation for the work presented in subsequent chapters.

3 Method and Experiment Architecture

This chapter outlines the methodology used to conduct the experiments.

3.1 Dataset Construction and Filtering

The starting point was a multi-source dataset constructed of various questions with correct answers and the corresponding document corpus. It was originally filtered using `GPT-4o` to identify questions that cannot be answered from world knowledge alone [2]. The original contribution of this dataset was ensuring that answering these questions requires retrieval rather than relying on a model's parametric knowledge. This makes it particularly suitable to benchmark retrieval quality.

The original dataset comprised 11,758 questions[7] compiled from twelve different original datasets:

> We select 12 long-context QA datasets frequently used in studies comparing LC[8] and RAG: Natural Questions, 2WikiMultihopQA, HotpotQA, MuSiQue, MultiFieldQA, NarrativeQA, QASPER, QuALTY, Coursera, TOEFL-QA, and MultiDoc2Dial. We also include the NovelQA dataset, a high-quality, human-annotated resource derived from long-form novels.
>
> — [2]

The datasets have been selected to provide a high variety of different questions catering to different use cases.

[7] In their paper, the authors mentioned a total of 13,628 questions. The dataset provided with the paper only contains 11,758 questions. Hence, this thesis can only use 11,758 questions.

[8] Long Context

Natural Questions (NQ) [43] is a large-scale open-domain QA benchmark of real Google search queries paired with Wikipedia pages and annotated with both long and short answers or null labels. 2WikiMultihopQA [44] is a multi-hop QA dataset built from aligned Wikidata–Wikipedia evidence, where templated questions are accompanied by explicit reasoning paths across multiple entities and documents. HotpotQA [45] consists of 113K Wikipedia-based QA pairs whose diverse questions require reasoning over multiple supporting documents with sentence-level supporting fact annotations and comparison questions. MuSiQue [46] is a multi-hop QA benchmark formed by composing interdependent single-hop questions to questions that require reasoning over multiple steps, additionally including contrastive answerable/unanswerable variants. MultiFieldQA [47] comprises roughly 150 manually curated questions over long single documents from law, government, encyclopedias, and scientific articles, targeting long-context single-document comprehension. NarrativeQA [48] is a reading comprehension dataset based on full books and movie scripts, where questions require global narrative understanding across entire documents. QASPER [49] is an information-seeking QA dataset over 1,585 NLP papers with 5,049 questions authored by practitioners and supporting evidence. QuALITY [50] is a multiple-choice QA benchmark with about 5K-token passages and questions written by readers of the full text. NovelQA [51] is a long-context benchmark based on English novels with documents exceeding 200K tokens on average and human-authored questions with evidence annotations targeting deep understanding of text in LLMs. MultiDoc2Dial [52] is a dialogue dataset grounded in multiple datasets in which conversations are conditioned on multiple domain documents, requiring integration of information from multiple documents across four different domains. TOEFL-QA [53] is a multiple-choice listening comprehension dataset of 963 examples from the TOEFL test with relatively short narrative contexts assessing English understanding. The Coursera QA [54] dataset contains 172 multiple-choice questions with multiple correct answers about course materials with an average context length of ~9K tokens per document. It tests knowledge-based comprehension of instructional content and was made for use in long-context evaluation.

Table 1 shows the distribution and actual sources from the original data sources as outlined in [2].

Note that for multi-document datasets (marked “multi” in the Doc column), the original benchmark provides a single pre-concatenated context containing all relevant source documents rather than separate document records. This means that even for multi-hop reasoning datasets like HotpotQA, MuSiQue, and 2WikiMultihopQA, which originally require evidence from multiple Wikipedia articles, this experiment treats each merged context as a single retrievable unit. Consequently, document recall measures whether this merged context was retrieved, not whether individual supporting documents were found separately.

For this experiment, the dataset was reduced to 6,284 questions for practical reasons.

Dataset	T	Doc	Source	Avg Len	# Q	# Kept	% Kept
NQ	K	multi	Wikipedia	18,164.7	109	22	20
Coursera	K	multi	Coursera	7,934.3	172	54	32
NovelQA	C	single	books	67,000.0	210	109	52
2WikiMHQA	R	multi	Wikipedia	7,191.3	300	152	51
HotpotQA	R	multi	Wikipedia	10,602.7	200	93	47
MuSiQue	R	multi	Wikipedia	12,974.3	200	140	70
MultiFieldQA	C	single	papers, reports	5,706.1	150	121	81
NarrativeQA	C	single	books, films	25,274.2	200	171	86
QASPER	C	single	papers	5,350.3	224	221	99
QuALTY	C	single	stories	5,089.2	202	202	100
TOEFL-QA	C	single	exams	729.1	121	121	100
MultiDoc2Dial	C	multi	dialogue	3,076.9	158	158	100

Table 1: Overview of the original datasets as outlined in [2]: 'The column "T" represents dataset type with values "K" for "Knowledge", "R" for "reasoning", and "C" for "reading comprehension". […] We also report number of questions in each set (# Q), number and percentage of questions retained after filtering (# Kept and % Kept) out questions needing no context[…]'. "Avg Len" is the average size of the context that is provided to the model to answer the questions from each dataset in tokens.

Questions whose golden documents required more than estimated 100,000 tokens to process were removed, as the generation LLM used (`gpt-oss-120b`) supports only up to 128,000 tokens including overhead. Token counts have been approximated by counting the number of characters per document and dividing the resulting count by 4, since this method was a lot faster than counting the number of tokens in Python and precise enough for the experiment. This approximation is based on the observation that English text averages approximately 4 characters per token for GPT-style tokenizers. For standard English prose, this method typically yields estimates within 10-20% of the actual token count. The approximation tends to slightly underestimate token counts since technical content, code snippets, or non-English text may have different character-to-token ratios. Given that the threshold was set at 100,000 tokens with a model context of 128,000 tokens, this margin of error was acceptable for the filtering purpose.

The removal primarily affected questions from the NarrativeQA dataset, removing 370 questions. Questions from the NovelQA dataset were excluded before processing as its full-novel documents exceeded practical length limits.

In the original paper, documents which exceeded the model context were truncated from the end of the context. Truncating has the potential problem of removing important information when it is located in the end of the document. Therefore, and since this affected only 370 questions of the total 11,758, it made more sense for the experiment in this thesis to remove the too long questions instead of truncating them.

Some datasets contained thousands of questions (2WikiMultihopQA, HotpotQA, MuSiQue, NarrativeQA, QuALTY, QASPER) while others had fewer than 100 (e.g., Coursera with only

54). Each dataset was capped at 800 questions to achieve better balance. Table 2 shows the full overview over the number of questions per dataset before and after filtering. The final dataset spans eleven sub-datasets with more even representation, allowing for systematic comparison across retrieval methods.

The filtering choices were deliberate and reflect the intended scope of this research. This thesis focuses on QA tasks where source documents are moderately sized (under 100k tokens), covering the vast majority of practical RAG applications such as technical documentation, knowledge bases, and business documents. Full-novel or book-length retrieval represents a distinct problem domain with different characteristics: it typically requires long-range narrative comprehension, character tracking across hundreds of pages, and synthesis of information spread across very large spans of text. This led to the exclusion of the NovelQA dataset for the experiment.

While dense retrieval methods may offer advantages in such long-context scenarios, this thesis explicitely scopes its investigation to the more common use case of retrieval over document collections of moderate length. This scoping decision means the findings should be interpreted as applicable to typical enterprise and knowledge-management RAG deployments rather than to specialized literary or long-form narrative applications.

Beyond scope considerations, the reductions were also pragmatic (compute/runtime) and methodological (avoid long-context failures that dominate variance without informing retrieval quality).

For validation purposes, a sample of 25 questions per dataset (275 total) was selected for manual review.

Dataset	**# of questions in the original dataset**	**# of questions after filtering**
2WikiMultihopQA	884	800
Coursera	54	54
HotpotQA	1020	800
MultiDoc2Dial	158	158
MultiFieldQA	121	121
MuSiQue	1523	800
NarrativeQA	1709	800
Natural Questions	351	351
QASPER	2453	800
QuALTY	2523	800
TOEFL-QA	962	800
Total	**11,758**	**6,284**

Table 2: Number of questions per dataset before and after filtering.

3.2 Model Selection

The `gpt-oss-120b` model was used, a state-of-the-art open-source model released by OpenAI, with a 128k token context window [55]. This choice was driven by reproducibility concerns and institutional access. The model is used to generate the actual answer to the question as well as for query rewriting in some retriever variants (see Section 3.3.2).

Since the original dataset was filtered using OpenAI's `GPT-4o`, there may be questions that `gpt-oss-120b` can answer from world knowledge that `GPT-4o` could not, or vice versa. Further adding to this, `gpt-oss-120b` has its training cutoff in June 2024 [55], whereas `GPT-4o` was trained on data collected until October 2023 [56]. The `None` baseline (described in Section 3.3.2) helps to check for this. Re-running the original experiment used to create the original dataset would be needed to update it for `gpt-oss-120b`, however, this is out of scope for this thesis.

For vector embeddings, the `Qwen3-4B` embedding model was used, selected for its strong benchmark performance on the MTEB benchmark and leaderboard [57].

To rerank search results, the `Jina Reranker v3` model was used, a lightweight, high-performance model based on `Qwen3-0.6B` and released in October 2025 [58]. The model uses a novel architecture which processes all documents and the query in the same context window, enabling the model to cross-reference different documents against each other and not just to the query. It achieves a BEIR performance of 61.85 nDCG@10 with only 0.6B parameters [58].

3.3 Experiment Architecture

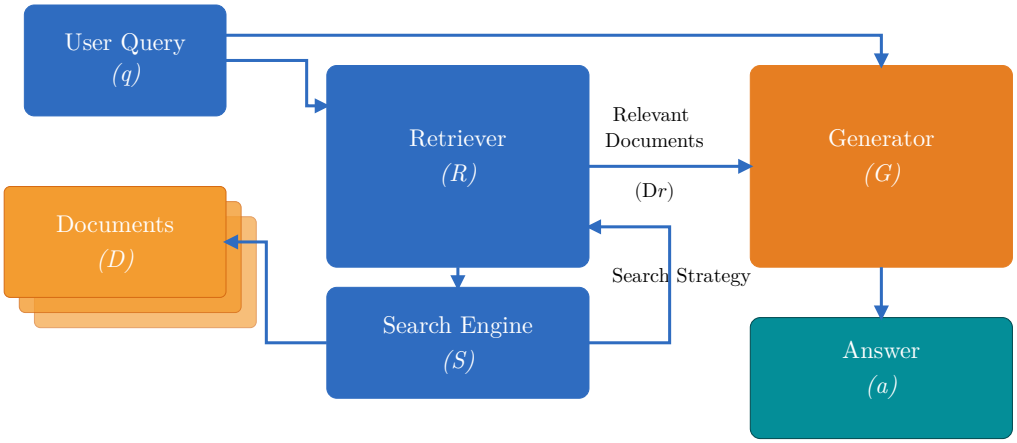

Figure 2: Based on the RAG definition in Figure 1, this shows the version used in the experiment in this thesis, adding a search engine which the retriever uses along with a search strategy to retrieve documents.

Figure 2 shows the architecture of the RAG system used in the experiment of this thesis. The implemented architecture separates the concepts of **retriever** and **search engine** to enable a systematic comparison of different components. This allows to create a matrix of experiments, testing different query strategies against different search algorithms. Decoupling the retrieval and search step makes it possible to freely change the underlying search engine, making the design very suitable for this experiment.

Compared to the architecture introduced in Section 2.2, this makes the actual search of the retrieval part a separate step.

3.3.1 Search Engines

Four primary search backends have been implemented:

1. **SQLite Full-Text Search**: Combines SQLite Full-Text Search with BM25 to rank the results [59]. Since many specialized full-text search databases on the market ultimately rely on BM25 or variants, this implementation in SQLite was treated as representative of this class of ranking algorithms. Other databases in this class are ParadeDB [60], Elasticsearch [61], SingleStore [62], and MongoDB [63].
2. **pgVector**: Uses PostgreSQL with the `pgVector` extension [64] to do semantic search using cosine distance search over embeddings generated by the `qwen3-4B` embedding model with 2,560 embedding dimensions. Data is stored using the `halfvec` data type and an Hierarchical Navigable Small World (HNSW) index, resulting in approximate nearest neighbor search when querying the embeddings.
3. **Typesense**: For each search, it first computes the frequency of search query tokens, the edit distance, and proximity to other words in the document corpus. In a second step, it uses a tie-breaking algorithm to rank results [65].
4. **Meilisearch**: Similar to Typesense, this database computes relevancy by different factors. By default these factors are number of matched query terms, typo distance, proximity between words, user-supplied attribute and sort order, and the similarity of the matched words with the query words in ascending order of importance [66]. Finally, results are sorted with a bucket sort algorithm according to the score calculated in the first step [67]. For the experiment in this thesis, the default ranking order has been used.

Additionally, two baseline implementations were included:

1. **None**: Returns no documents, serving as a lower baseline to verify the model cannot answer questions from parametric memory alone. If the used LLM can still answer questions correctly, it indicates the question is part of its world knowledge, even if it wasn't for `GPT-4o`.
2. **Golden**: Always returns only the correct document (the single context associated with each question, which for multi-document datasets contains all merged source texts), simulating a perfect retriever. This represents the upper baseline and achieves 100% document recall by definition.

Each search engine returns up to 10 documents per query by default, along with a relevance score to indicate how useful the result might be for the search term. In the case of SQLite Full-Text Search (FTS), this is the calculated BM25 score, for pgVector this is $\frac{1}{1+\text{ Cosine Distance}}$, Typesense and Meilisearch return their own relevancy scores as returned from their ranking algorithms. For the None search engine, since no documents are returned no score is returned either. The Golden search engine always returns a score of 1.

The score is passed along with the document to the LLM when it generates the response. This design choice was not ablated; future work could re-run a subset of configurations without score information in the generator context to investigate whether providing the scor across search engines influences the generation result. The full generation prompt can be found in Appendix A.1.

3.3.2 Retriever Types

Each search engine can be queried through four retriever architectures:

1. **Passthrough**: Forwards the query directly to the search engine without modification. When searching with the Embeddings Search engine, the Passthrough retriever is not used due to the nature of embeddings, which stores and returns only chunks of text. Therefore, Chunk and Passthrough in embeddings represent the same retriever types and only ChunkRetriever is used.
2. **Search**: Uses an LLM[9] to rewrite the query into a more effective search query before passing it to the search engine. The full prompt used to rewrite the query is listed in Appendix A.2.1.
3. **Keyword SearchRetriever**: Uses an LLM[9] to generate keywords from the question, then searches for each keyword separately. The full prompt used to rewrite the query is listed in Appendix A.2.2. Results are aggregated and filtered to the top-10 documents by search engine score. This approach aims to leverage the strengths of keyword-based search while using full-text search infrastructure. Note that the keyword generation prompt was designed with FTS in mind and was not specifically tuned for embedding-based retrieval, where natural language queries might perform better than isolated keywords.
4. **ChunkRetriever**: Similar to Passthrough, but this retriever returns document chunks rather than full documents. See Section 3.3.3 for details.
5. **Fulldocs**: This retriever is only used when using the Embeddings Search Engine. It returns the full documents whose chunks have been returned while searching for embeddings. Because all other search engines return full documents when the ChunkRetriever is not used, this retriever aims to control for cases when the LLM would generate a correct response simply because it had the full document available. The score of the retrieved full documents is the same as of the chunks that were originally retrieved.

All search-engine-specific index and ranking settings (tokenization, stopwords, stemming/ synonyms, BM25/HNSW parameters, and Typesense/Meilisearch ranking configuration) are documented in Appendix A.5.

3.3.3 Chunking Strategy

When building RAG systems, an important decision involves whether to return full documents or chunks. This is due to the "lost in the middle" phenomenon which suggests that models

[9]In this experiment, the same model as for all other LLM tasks, `gpt-oss-120b` was used. It would also be possible to use a smaller, faster LLM or a fine-tuned variant for search term or keyword generation with potentially different results.

may struggle with extremely long contexts. Other research on Long Context observed how LLMs struggle especially to use information that is in the middle of their context, opposed to information that is at the beginning or the end of the provided context [68]. This leads to the conclusion that Long Context is not a silver bullet to improving the answering capabilities in a RAG system.

Instead, providing smaller, more relevant chunks of documents may reduce context overhead and yield better results. To examine this hypothesis, the chunk retriever was added to only search and retrieve chunks of documents.

In the conducted experiment, these chunks were pre-generated and stored in each search engine for better performance, though runtime generation based on search matches is a potential alternative. Chunks were created with a maximum size of 512 characters and an overlap of 50 characters between consecutive chunks. The chunking algorithm prefers sentence boundaries when splitting, avoiding cuts in the middle of sentences where possible. TODO das klingt nicht richtig Ultimately, the chosen implementation depends on the search engine used and whether the results provided by it have enough precision to create good chunks. During retrieval, this is similar to the Passthrough retriever with the key difference of returning chunks instead of full documents.

For each search engine, full document retrieval and chunk-based retrieval were tested. This allows to test whether providing focused, relevant chunks outperforms full document context.

For embedding-based search, the Passthrough and Chunk retrievers are functionally equivalent, the Passthrough retriever for embeddings was therefore excluded from comparative analysis to avoid redundant data points.

3.4 Reranking

Reranking is a refinement step which uses a specially trained reranking model to reorder the retrieved results, putting the most relevant on the top of the list. Section 2.5.5 explains reranking in more detail. In theory, reranking should be able to improve especially noisy FTS results for usage in RAG.

In this experiment, reranking is applied using Jina Reranker v3 after initial retrieval. This model, released in October 2025 and based on the Qwen3-0.6B LLM, represents current state-of-the-art performance on the BEIR benchmark while remaining relatively lightweight [58].

Experiments were performed on all search configurations both with and without reranking on the full dataset to isolate its contribution. This allows evaluation of both base retrieval quality and the impact of reranking across different search engine choices.

In the architecture presented in Figure 2, reranking happens as part of the retriever.

3.5 Evaluation

The evaluation method employed for the experiment combines manual review with automated assessment.

First, approximately 1,000 generated answers with different retrieval methods for the previously selected 275 sample questions were evaluated manually as correct, incorrect or partially correct. This provides ground truth for automated validation.

Then, a model-as-judge approach was developed where an LLM[10] receives the question, correct answer, and generated answer, then classifies the response as correct, incorrect or partially correct.

Multiple-choice datasets (QuALITY, TOEFL-QA, Coursera) are evaluated identically to open-ended questions. The generation model outputs free-form text responses which may include option letters, option text, paraphrases, or combinations thereof (e.g., "B. she wore free-flowing costumes and D. she danced without shoes" or "The class was canceled because there weren't enough students enrolled (Option A)"). The expected answers for multiple-choice questions are stored as option letters (e.g., "B" or "BD" for multi-answer questions). The judge performs semantic comparison rather than exact string matching, determining whether the generated response conveys the same meaning as the expected answer. For multi-answer multiple-choice questions, the partially correct category captures cases where the model identifies some but not all correct options.

The model-as-judge approach was first introduced in 2023 by Zheng et al. [69]. Originally used to rate answers by different chat LLM, it matched the results of human annotators by 80%.

To run and optimize the prompt, the `dspy` Python library has been used. A `MIPROv2` optimizer was employed to improve the prompt, an optimizer designed to improve multi-stage Language Model Programs by refining both free-form instructions and few-shot demonstrations at the same time to maximize a final task metric. In the library, many prompt variants are generated which are then searched using a Bayesian surrogate model to approximate the optimal prompt and search through proposed prompt combinations [70]. Results are validated using validation data that was collected manually.

Additionally, `dspy` simplifies using and managing the prompts in the code, streamlining the implementation.

The full optimized evaluation prompt can be found in Appendix A.3.

After generating training data manually and testing various models and prompts, 92% accuracy was achieved using the automated approach. This 92% figure represents the optimization metric calculated during prompt tuning on the manually labeled training data used by `dspy`.

To independently verify the evaluation quality, 216 answers generated with the golden retriever that also had manually rated labels available from the original sample data collection

[10]Here, `gpt-oss-120b` has been used as well.

were compared. Answers where the model replied that it did not know the response were excluded[11]. For each question, the binary correctness judgments were compared between manual and automatic methods:

$$\text{Agreement Rate} = \left(\frac{\text{Number of Matching Ratings}}{\text{Total Number of Paired Ratings}}\right) \times 100\%$$

This verification yielded an 84.7% agreement rate with a Cohen's Kappa of 0.618, indicating "substantial agreement" according to the interpretation scale presented by Landis and Koch [71]. The difference between training accuracy (92%) and validation agreement (84.7%) is expected: the optimization metric measures performance on the training data used to tune the prompt, while the agreement rate reflects real-world performance on previously unseen answers evaluated after the experiment.

While fine-tuning could potentially improve this further, the achieved reliability was determined to be sufficient for large-scale evaluation in the context of this thesis. Cohen's Kappa accounts for agreement occurring by chance, making it a more robust measure than simple percent agreement, and a value of 0.618 is considered adequate for the experiment in this thesis where the goal is to identify relative performance differences rather than precise absolute measurements.

3.6 Experimental Infrastructure

All experimental runs, questions, and model answers are logged in a central database for analysis. The system is built for robustness, allowing multiple runs to be started without overwriting previous data.

The question-answering prompt used in the generator follows best practices but required tuning to prevent excessively long responses. It was empirically observed that when the model lacks relevant documents, it tends to generate really verbose but incorrect answers, particularly when only irrelevant documents are retrieved. However, when no documents are found, the model reliably responds with "I don't know" as instructed. This matches current research findings showing that LLMs perform worse when the context contains irrelevant information than when the context is empty [68].

3.7 Key Considerations and Limitations

The used dataset was originally filtered using `GPT-4o`, while this experiment uses `gpt-oss-120b`. Differences in model capabilities and knowledge cutoffs may affect which questions genuinely require retrieval.

Due to `gpt-oss-120b`'s 128k token context window, some documents exceed practical limits. The dataset was filtered by token size to address this, but context window limitations remain relevant even for newer long-context models due to performance degradation and the "needle in the haystack" problem.

[11] Cases where the model replied with "I do not know the answer to your question." as instructed in the system prompt.

All experiments were conducted on a single model with one embedding model. Results may not generalize to other model families.

In total, 32 configurations were evaluated: 4 search engines (SQLite FTS, pgVector, Typesense, Meilisearch) × 4 retrievers (Passthrough, Search, Keyword, Chunk) × 2 reranking options (with and without), minus the 2 redundant embedding Passthrough configurations, plus the 2 full document retreival configurations, plus 2 baselines (Golden and None). No prompts or hyperparameters were tuned after inspecting experimental results. The prompts for query rewriting and keyword generation were designed a priori based on task requirements, and search engine parameters (e.g., returning top-10 results) were fixed before experiments began. Hyperparameter optimization was outside the scope of this thesis, which focused on comparing retrieval strategies under consistent conditions rather than optimizing individual configurations.

4 Results

This chapter presents the experimental results obtained from the evaluation outlined in the previous section. These results are used to answer the research question of whether full-text search can be a viable alternative to embedding-based search in RAG systems. The findings are categorized into results across all datasets in Section 4.1, document recall analysis in Section 4.1.3, timing analysis in Section 4.2, and the best configurations per dataset Section 4.3 and Section 4.4.

4.1 Overall Performance Across Search Engines

This section presents the broad results of the experiment. It is expected that using external data via RAG contributes significantly to the results of the QA tasks.

Table 3 presents the comprehensive results across all search engine configurations. The search engine, retriever, and reranking columns describe the retrieval pipeline components: the search engine (see Section 3.3.1), the retriever type (see Section 3.3.2), and whether reranking was applied (see Section 3.4). Answer quality is captured through four percentage-based metrics: the percentage of correctly retrieved answer documents (Doc Recall), the percentage of questions answered correctly, partially correct, and the combined correct plus partial rate, all evaluated using the model-as-judge approach described in Section 3.5. Performance characteristics are reported through average retrieval time, average completion time, and average total response time, all measured in seconds and computed across the full set of evaluated queries.

Search Engine	Retriever	Reranked	Doc Recall	% Correct	% Partial	% Correct+Partial	Avg Retrieval (s)	Avg Completion (s)	Avg Response (s)
Golden	Passthrough	No	100.00%	69.40%	5.52%	74.91%	0.0097	1.8689	1.8787
None	Passthrough	No	0.00%	7.65%	1.40%	9.05%	0.0000	0.9521	0.9521
Embeddings	Chunk	No	40.07%	24.59%	3.51%	28.10%	0.2264	1.2641	1.4905
		Yes	40.12%	24.06%	3.41%	27.47%	0.3291	1.6924	2.0215
	Fulldocs	No	33.24%	28.58%	3.06%	31.64%	0.4957	9.3811	9.8772
		Yes	37.65%	31.19%	3.39%	34.58%	26.3811	8.3206	34.7021
	Keyword	No	6.71%	8.06%	2.50%	10.56%	1.4247	1.0384	2.4631
		Yes	6.54%	7.15%	2.16%	9.31%	2.4345	1.2453	3.6798
	Search	No	27.83%	17.98%	2.61%	20.59%	0.2364	1.1580	1.3944
		Yes	27.59%	16.96%	2.78%	19.75%	0.3448	1.1391	1.4840
Full-Text Search	Chunk	No	0.52%	7.70%	1.48%	9.18%	0.0044	0.9581	0.9625
		Yes	0.53%	8.10%	1.38%	9.48%	0.0539	1.3200	1.3739
	Keyword	No	8.29%	14.82%	2.52%	17.34%	2.0209	8.5963	10.6173
		Yes	10.50%	16.12%	2.72%	18.84%	6.9150	14.0422	20.9572
	Passthrough	No	20.80%	22.46%	2.71%	25.17%	0.0867	3.8680	3.9548
		Yes	27.64%	25.27%	2.93%	28.20%	1.7182	4.5196	6.2381
	Search	No	21.21%	22.77%	2.46%	25.23%	1.4125	3.9697	5.3822
		Yes	25.65%	24.52%	2.50%	27.02%	3.1187	4.2398	7.3585
Meilisearch	Chunk	No	12.53%	4.46%	1.10%	5.56%	0.0095	1.0940	1.1036
		Yes	12.56%	4.71%	1.24%	5.95%	0.1013	1.7684	1.8697
	Keyword	No	56.25%	39.18%	4.46%	43.64%	0.2804	9.7985	10.0789
		Yes	54.69%	40.44%	4.47%	44.91%	4.1565	22.8093	26.9658
	Passthrough	No	36.63%	29.30%	3.17%	32.47%	0.1901	9.4851	9.6753
		Yes	36.19%	29.66%	3.53%	33.20%	4.9603	9.7008	14.6613
	Search	No	46.64%	34.18%	3.75%	37.94%	0.1215	8.0988	8.2203
		Yes	47.45%	36.97%	3.42%	40.39%	3.0814	8.3723	11.4537
Typesense	Chunk	No	11.96%	3.43%	0.80%	4.23%	0.2070	0.9512	1.1583
		Yes	11.90%	3.64%	0.62%	4.26%	0.4930	1.3147	1.8078
	Keyword	No	46.21%	37.32%	3.69%	41.01%	1.2029	8.4977	9.7006
		Yes	46.16%	37.29%	3.77%	41.06%	7.3586	28.9181	36.2768
	Passthrough	No	32.84%	29.60%	3.23%	32.83%	0.7932	4.6709	5.4644
		Yes	33.78%	30.49%	3.06%	33.55%	5.0499	4.8642	9.9146
	Search	No	39.35%	34.29%	3.18%	37.48%	0.3036	4.1061	4.4098
		Yes	39.26%	33.91%	2.99%	36.90%	2.6623	4.4194	7.0817

Table 3: Performance per search engine in all run configurations. Values marked in dark green are the best overall, values in light green are the best per search engine, values marked dark red are the worst overall, values in light red are the worst per search engine - for configurations except Golden and None. For Doc Recall, % Correct, % Partial, % Correct + Partial, higher is better, for average retrieval time, average completion time, and average response time, lower is better.

Dataset	Doc Recall	% Correct	% Partial	% Correct+Partial	Avg Retrieval (s)	Avg Completion (s)	Avg Response (s)
2WikiMultihopQA	35.32%	28.42%	0.86%	29.28%	2.2755	6.0229	8.2985
Coursera	28.14%	20.83%	40.95%	61.78%	1.7595	4.6152	6.3747
HotpotQA	44.58%	33.21%	0.91%	34.12%	1.8512	7.1827	9.0340
MultiDoc2Dial	24.02%	14.19%	6.61%	20.80%	2.8846	4.0879	6.9727
MultiFieldQA	35.63%	29.61%	3.19%	32.81%	1.9170	5.1421	7.0591
MuSiQue	33.02%	24.55%	1.81%	26.36%	2.0748	7.8541	9.9290
NarrativeQA	25.66%	9.60%	1.57%	11.17%	2.7864	5.0024	7.7889
Naturalquestion	47.78%	29.55%	4.08%	33.63%	1.7119	6.4107	8.1227
QASPER	17.94%	16.12%	6.55%	22.67%	1.4757	5.1588	6.6346
QuALTY	21.53%	20.75%	0.25%	21.01%	2.5239	4.0926	6.6166
TOEFL-QA	24.64%	32.23%	4.01%	36.24%	2.4068	3.7990	6.2058

Table 4: Summary per dataset across all run configurations. Doc Recall shows the average percentage of questions where the gold document was retrieved. Values marked in dark green are the best overall, values in light green are the 2nd best overall, values marked dark red are the worst overall, values in light red are the 2nd worst overall. For Doc Recall, % Correct, % Partial, % Correct + Partial, higher is better, for average retrieval time, average completion time, and average response time, lower is better.

The baseline configurations established the performance bounds for the evaluation: The Golden retriever, which returned only the correct source document, achieved 69.40% correctness, while the None retriever, which did not add any documents to the context, achieved 7.65% correctness. This indicates that 7.65% of questions can be answered correctly by `gpt-oss-120b` based solely on its parametric knowledge. This is more than what was previously filtered questions by `GPT-4o` when curating the original dataset – the questions filtered in the dataset could not be answered at all by `GPT-4o`. These general results across all different configurations provide a first insight into the overall trends of the experiment and are used to spot any outliers.

Similarly, Table 4 shows the results grouped by dataset across all retrieval configurations.

Meilisearch with keyword search and reranking achieved the highest performance among the tested retrieval methods at 40.44% correctness. The difference to without reranking was minimal, only 1.26% lower at 39.18% correctness. Typesense demonstrated similar performance characteristics as Meilisearch, including only a marginal difference between with or without reranking for keyword search – achieving 37.29% correctness with reranking and 37.32% without reranking.

Full-text search methods via SQLite and BM25 showed varied performance depending on the retrieval strategy. Simple direct search via the passthrough retriever yielded 25.27% correctness with reranking, on par with search in embeddings. With search query rewriting, performance slightly dropped to 24.52% correctness with reranking and 22.77% without reranking. Keyword-based full-text search achieved 16.12% correctness with reranking and 14.82% without reranking. These results suggest that the search query generation strategy performs comparably to direct passthrough search, likely due to the nature of full-text search mechanisms. Reranking has a small effect (roughly 2%), but the difference is modest.

Chunk-based embeddings achieved 24.06% correctness with reranking and 24.59% without reranking. When passing the full document of the retrieved chunks to the LLM, a slighly higher 28.58% correctness was observed. With reranking, these results were improved by almost 7% to 31.19%, representing the best results for the Embeddings Search Engine.

Keyword-based embedding search performed worst for the embedding search engine at 7.15% correctness with reranking, falling slightly below the baseline of 7.65%. Without reranking, performance is slightly better at 8.06%. This suggests that incorrect source retrieval actively degraded model performance.

Chunk-based retrieval with Meilisearch and Typesense performed notably poorly. Meilisearch chunk retrieval achieved only 4.71% correctness with reranking and 4.46% without reranking, Typesense chunk retrieval achieved 3.64% correctness with reranking and 3.43% without reranking. Both fall below the None baseline. These results indicate that the wrong chunks were retrieved, introducing noise that led the model to very wrong answers and degraded LLM performance below its baseline parametric knowledge.

Despite poor chunk performance, direct search with Meilisearch achieved respectable results at 29.66% correctness with reranking and 29.30% without reranking, on par with full document embeddings search and better than standard full-text search. When search queries were generated, Meilisearch achieved an even better score of 36.97% correctness with reranking and 34.18% without reranking.

4.1.1 Best Embedding vs. Full-Text Search

To analyse whether a systematic performance difference exists between embedding-based and full-text search approaches, the best-performing configuration from each category was compared across all evaluated datasets. Table 5 presents the results of the best full-text search approach (one of SQLite FTS, Meilisearch, or Typesense) against the best embedding-based approach for each dataset.

The results reveal a consistent and substantial performance gap favoring full-text search methods. This difference becomes especially clear when examining only the % Correct metric, as illustrated in Figure 3. Full-text approaches outperform embedding-based retrieval on all datasets except TOEFL-QA, and tie on Coursera with a correctness of 35.19%. On TOEFL-QA, embeddings achieve 65% correct versus 46.62% for full-text search (an 18.38% advantage).

Only on the QuALITY dataset passing chunks from embeddings results performed better, in all other cases passing the full document to the LLM yielded better results.

The most substantial gaps are observed on multi-hop reasoning datasets. 2WikiMultiHopQA exhibits the largest disparity (62.59% vs. 19.38%, a 43.21% difference), followed by MultiFieldQA (73.55% vs. 35.54%, a 38.01% difference) and HotpotQA (64.50% vs. 31.87%, a 32.63% difference). For 2WikiMultiHopQA and MultiFieldQA, keyword-based matching appears to provide more reliable retrieval than semantic similarity.

Mid-range gaps are observed on MultiDoc2Dial (37.97% vs. 17.09%, a 20.88% difference) and QuALTY (44.38% vs. 31.13%, a 13.25% difference). Smaller gaps appear on MuSiQue (40.50% vs. 34.62%, a 5.88% difference), Naturalquestion (51.85% vs. 48.15%, a 3.70% difference), QASPER (26.62% vs. 22.00%, a 4.62% difference), and NarrativeQA (18.41% vs. 13.75%, a 4.66% difference), while Coursera is a tie at 35.19%.

Across all datasets, full-text search approaches outperform embeddings by an average of 13.50%.

Dataset	Search Engine	Retriever	Reranked	Doc Recall	% Correct	% Partial	% Correct+Partial	Avg Retrieval (s)	Avg Completion (s)	Avg Response (s)
2WikiMultihopQA	Embeddings	Fulldocs	No	18.62%	19.38%	1.38%	20.75%	0.5817	8.1221	8.7043
	Typesense	Keyword	No	82.67%	62.59%	1.50%	64.09%	0.8358	8.7734	9.6093
Coursera	Embeddings	Fulldocs	No	48.15%	35.19%	38.89%	74.07%	0.4420	8.3087	8.7509
	Typesense	Keyword	Yes	50.00%	35.19%	42.59%	77.78%	6.1540	22.3476	28.5016
HotpotQA	Embeddings	Fulldocs	No	33.88%	31.87%	0.88%	32.75%	0.4864	12.3024	12.7892
	Typesense	Search	Yes	76.50%	64.50%	1.50%	66.00%	1.1555	4.2944	5.4500
MultiDoc2Dial	Embeddings	Fulldocs	Yes	22.15%	17.09%	5.70%	22.78%	31.4385	3.9699	35.4089
	Meilisearch	Keyword	Yes	65.82%	37.97%	12.66%	50.63%	3.8125	24.7262	28.5387
MultiFieldQA	Embeddings	Fulldocs	Yes	39.67%	35.54%	1.65%	37.19%	22.3970	6.8473	29.2445
	Meilisearch	Keyword	No	82.64%	73.55%	4.96%	78.51%	0.2541	9.3839	9.6380
MuSiQue	Embeddings	Fulldocs	Yes	44.75%	34.62%	2.12%	36.75%	25.0087	13.5106	38.5196
	Meilisearch	Search	Yes	50.75%	40.50%	1.62%	42.12%	2.7113	11.6012	14.3124
NarrativeQA	Embeddings	Fulldocs	Yes	27.50%	13.75%	1.12%	14.88%	38.1082	6.5069	44.6156
	Meilisearch	Passthrough	No	40.67%	18.41%	2.11%	20.52%	0.2267	8.4513	8.6781
Naturalquestion	Embeddings	Fulldocs	Yes	70.09%	48.15%	5.70%	53.85%	10.0871	11.9983	22.0855
	Meilisearch	Keyword	Yes	84.05%	51.85%	5.70%	57.55%	3.9187	13.7215	17.6403
QASPER	Embeddings	Fulldocs	Yes	24.38%	22.00%	9.00%	31.00%	13.4800	8.7037	22.1838
	Meilisearch	Keyword	Yes	33.38%	26.62%	10.25%	36.88%	2.9298	8.4380	11.3678
QuALTY	Embeddings	Chunk	No	34.44%	31.13%	0.12%	31.25%	0.2256	1.2094	1.4350
	Meilisearch	Keyword	Yes	53.62%	44.38%	0.12%	44.50%	4.0963	11.4563	15.5526
TOEFL-QA	Embeddings	Fulldocs	Yes	71.00%	65.00%	5.75%	70.75%	22.8609	5.1679	28.0291
	Meilisearch	Keyword	Yes	50.88%	46.62%	6.38%	53.00%	4.4158	13.4178	17.8336

Table 5: Results for the best Full-Text Search configuration (One of BM25-based SQLite Full-Text Search, Meilisearch or Typesense) vs. the best embedding-based configuration. Doc Recall shows the percentage of questions where the gold document was retrieved. Values marked in dark green are the best overall, values in light green are the best per search engine, values marked dark red are the worst overall, values in light red are the worst per search engine - for configurations except Golden and None. For Doc Recall, % Correct, % Partial, % Correct + Partial, higher is better, for average retrieval time, average completion time, and average response time, lower is better.

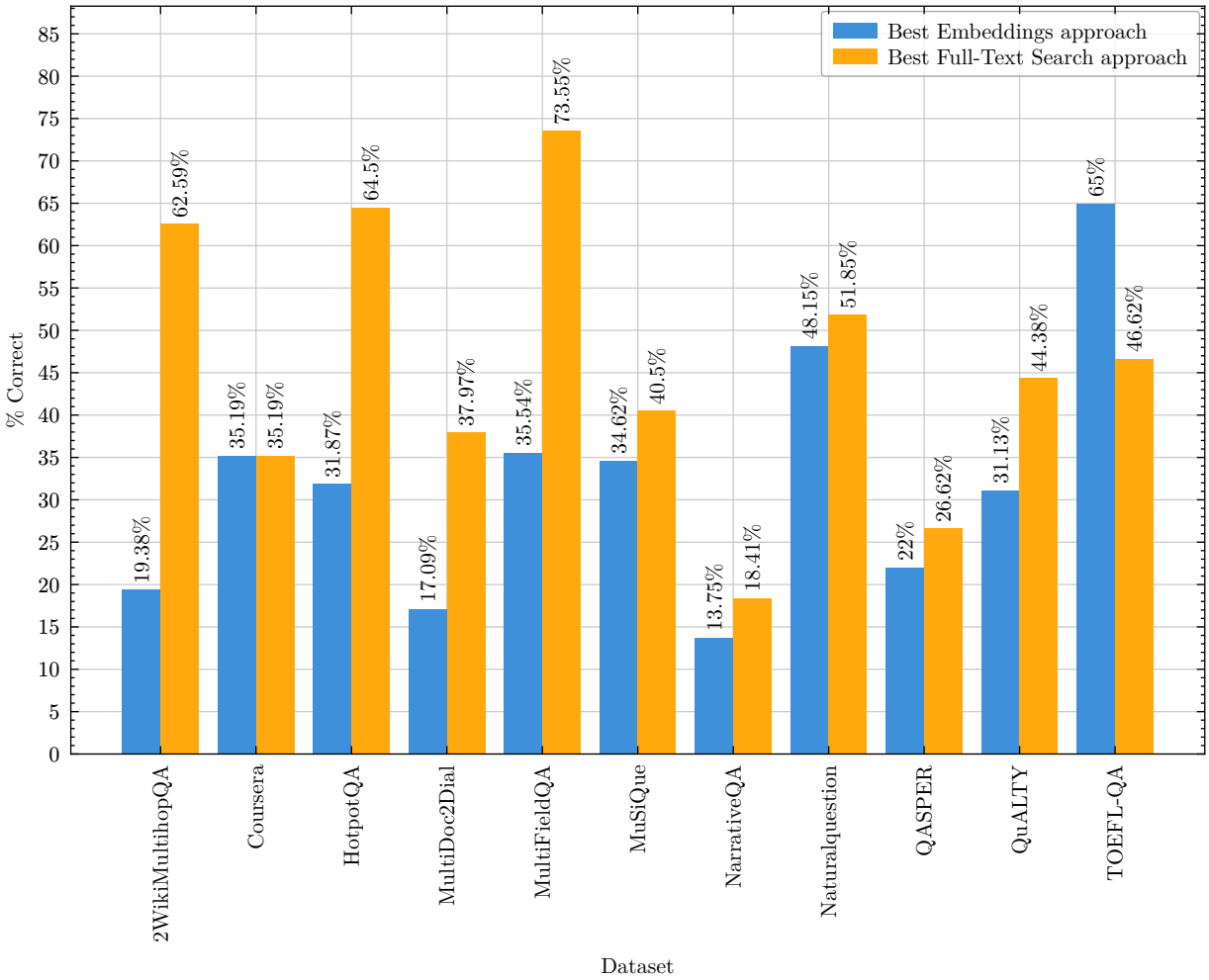

Figure 3: The best embedding-based and best Full-Text search approaches with their % Correct, as shown in Table 5.

4.1.2 Statistical Significance

To verify that the observed performance differences are not due to chance, statistical tests were performed on the key comparison between full-text search and embedding-based retrieval.

The best full-text search configuration (Meilisearch with keyword generation and reranking) achieved 40.44% accuracy (95% CI: 39.23%–41.65%, n=6,284), compared to 31.19% (95% CI: 30.06%–32.35%, n=6,284) for the best embedding-based configuration (full document retrieval with reranking). The confidence intervals were computed using the Wilson score method, which provides more accurate coverage for proportions than the normal approximation. The non-overlapping confidence intervals indicate the difference is statistically meaningful.

Since both methods answered the same set of questions, McNemar's test for paired data was performed. McNemar's test is designed for paired nominal data and specifically examines whether the disagreements between methods are systematic. The test focuses on questions where the two methods disagreed and determines whether one method systematically outperforms the other. The result ($\chi^2 = 163.2$, $p < 0.001$) confirms that full-text search systematically outperforms embeddings: FTS correctly answered 1,321 questions that embeddings missed, while

embeddings only succeeded on 740 questions where FTS failed, representing a net advantage of 581 questions (9.2% of the test set).

These results demonstrate that the performance advantage of full-text search over embedding-based retrieval is not attributable to random variation but represents a genuine and substantial difference in retrieval effectiveness.

4.1.3 Document Recall

To separate retrieval failure from generation failure, document recall was measured for each retrieval configuration, shown in the Doc Recall column in all results tables (Table 4, Table 3, Table 6). Document recall measures if the source document containing the answer was retrieved, regardless of whether the LLM generated a correct answer from it. It can differ between configurations with and without reranking, even when using the same search engine and retriever, see Section 5.6 for an explanation.

The Golden retriever achieves 100% recall by definition, while the None retriever achieves 0% as expected. Looking at the results per retrieval configuration in Table 3, even the Golden retriever only achieves 69.40% answer correctness.

Among the evaluated retrieval methods, Meilisearch with keyword generation achieved the highest document recall at 56.25%, followed by Typesense with keyword generation achieved 46.21% document recall. Embedding-based retrieval peaked at 40.12% document recall with chunk retrieval, while only 37.65% when retrieving full documents, both with reranking. Full-text search with chunk retrieval showed the lowest recall at just 0.52%.

Figure 4 visualizes the relationship between document recall and answer correctness across retrieval configurations. The scatter plot shows a positive correlation: configurations with higher document recall tend to achieve higher correctness rates. The strength of this relationship varies by search engine type: embedding-based configurations at approximately 40% recall achieve 17-25% correctness, while full-text search engines show higher correctness rates at comparable recall levels.

For the case where embedding search was used but the full document was passed to the LLM, the results appear to be in line with FTS methods, even though they achieved lower % correct.

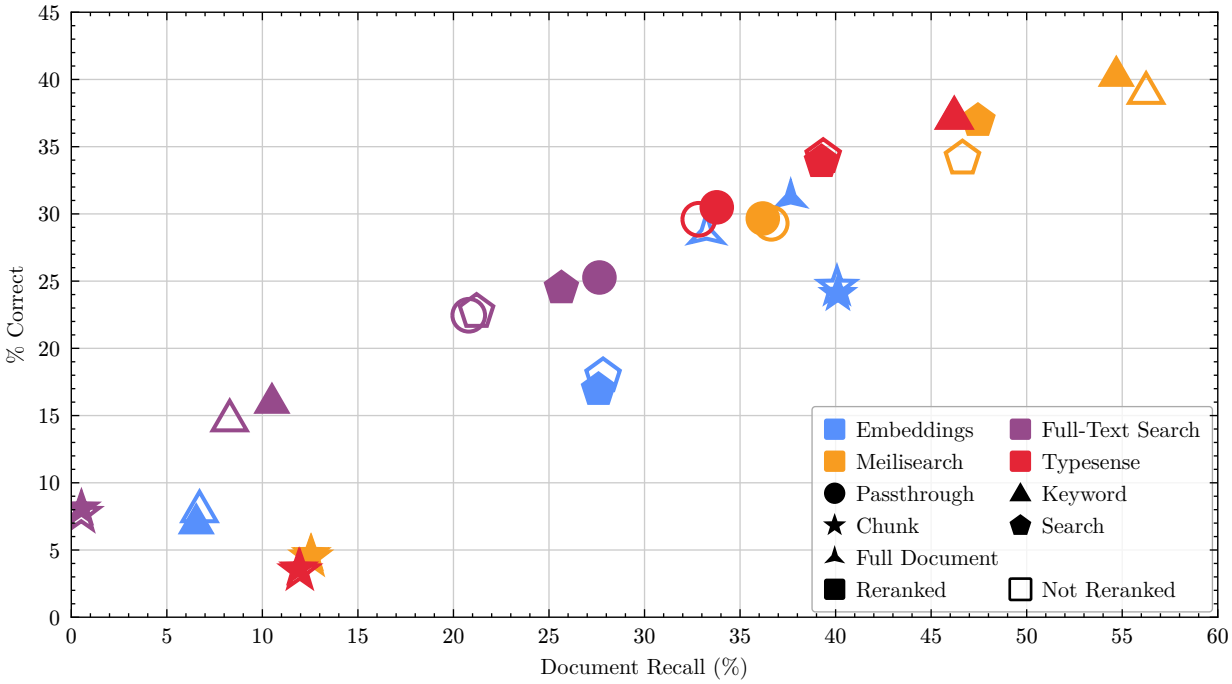

Figure 4: The relationship between document recall and % correct. Each point in the diagram represents a search engine configuration, colored by search engine type and distinguished by shape. Golden and None baselines are excluded.

4.2 Timing Analysis

Analyzing the response times of different configurations was done to see whether the retrieval method used has an effect on response times. If one retrieval method would turn out to be excellent in retrieval quality but takes a really long time to answer queries, that would render it unusable for real-time chat applications – even though the qualitative results might be better. Or, viewed from another angle, if the retrieval quality is high but response times are poor, further research could be conducted to see if the response time can be improved.

It is expected that adding different RAG mechanisms adds latency, especially when combined with reranking. Latency with the None retriever should be almost zero, with the Golden retriever it should be near-zero.

The timing measurements reported in this section are descriptive observations. The experiment relied on a university-hosted GPU instance where other users may have accessed the same resources simultaneously, introducing variability in response times. Therefore, timing results should be interpreted as indicative patterns rather than precise performance benchmarks.

Timing results across search engines (as shown in Table 3) revealed consistent patterns across configurations. Chunk-based retrieval consistently demonstrated the fastest retrieval times, likely because less data needed to be retrieved and transferred from the database. Full-text search with chunks showed the fastest overall performance.

The best-performing configurations when looking at % Correct, Meilisearch and Typesense with keyword retrieval, exhibited notably longer total response times compared to other

methods in the observed data. All of this can be attributed to the longer completion time when using these search engines.

In the observed measurements, Meilisearch keyword retrieval showed average total response times of 26.97 seconds with reranking and 10.07 seconds without reranking. Of this, 22.81 seconds have been spent in generation with and 9.78 seconds without reranking. Typesense showed an average of 36.28 seconds for keyword search with reranking and 9.70 seconds without. Similarly to Meilisearch, 28.92 seconds of the 36.28 seconds have been spent on average on the generation with reranking, vs. 8.50 seconds without reranking.

This extended generation time suggests that more documents were retrieved and processed by the language model. Figure 5 shows the the generation time compared to the number of retrieved documents for Typesense with keyword generation reranked as configuration with the longest completion times. The diagram shows an upward trend of the median response time (indicated by the line in the box charts), suggesting an association between number of documents retrieved and response time, which is consistent with this hypothesis.

To control for the potential confounding effect of different document return counts across configurations, Figure 6 presents the Pearson correlation between average content length and average completion time. Content length serves as a proxy for document return volume, since configurations returning more documents would be expected to have higher total content length.

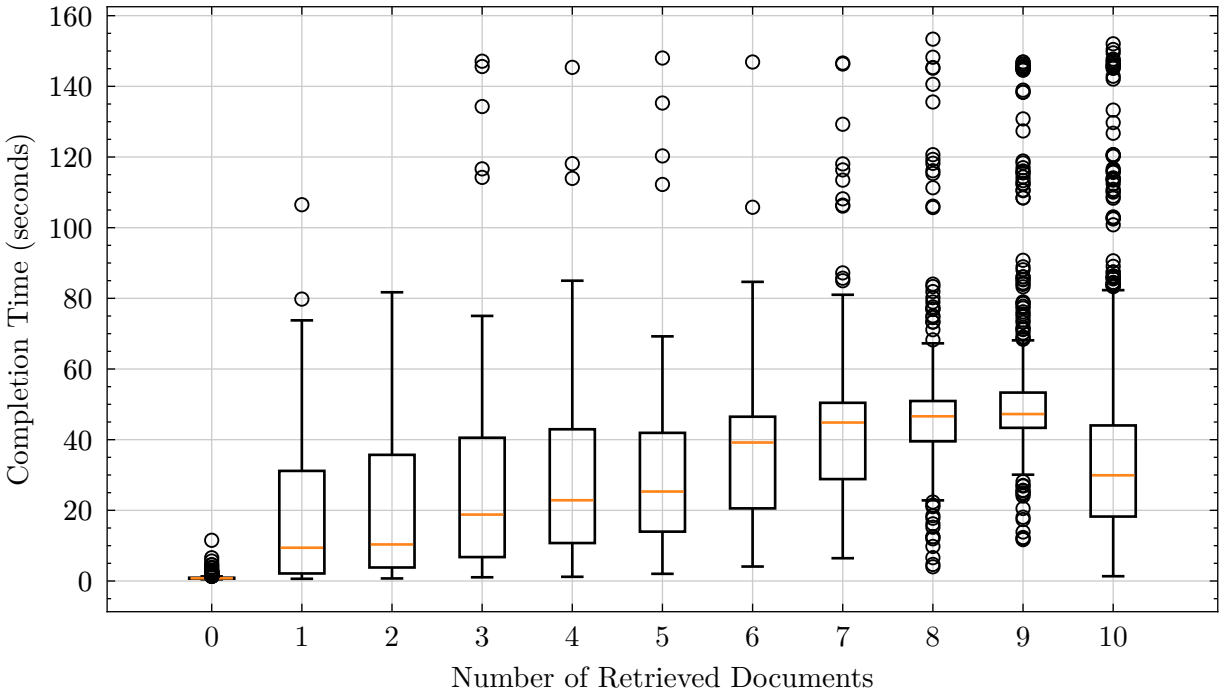

Figure 5: The number of retrieved documents vs. the completion time in seconds for the Typesense configuration with keyword search and reranking.

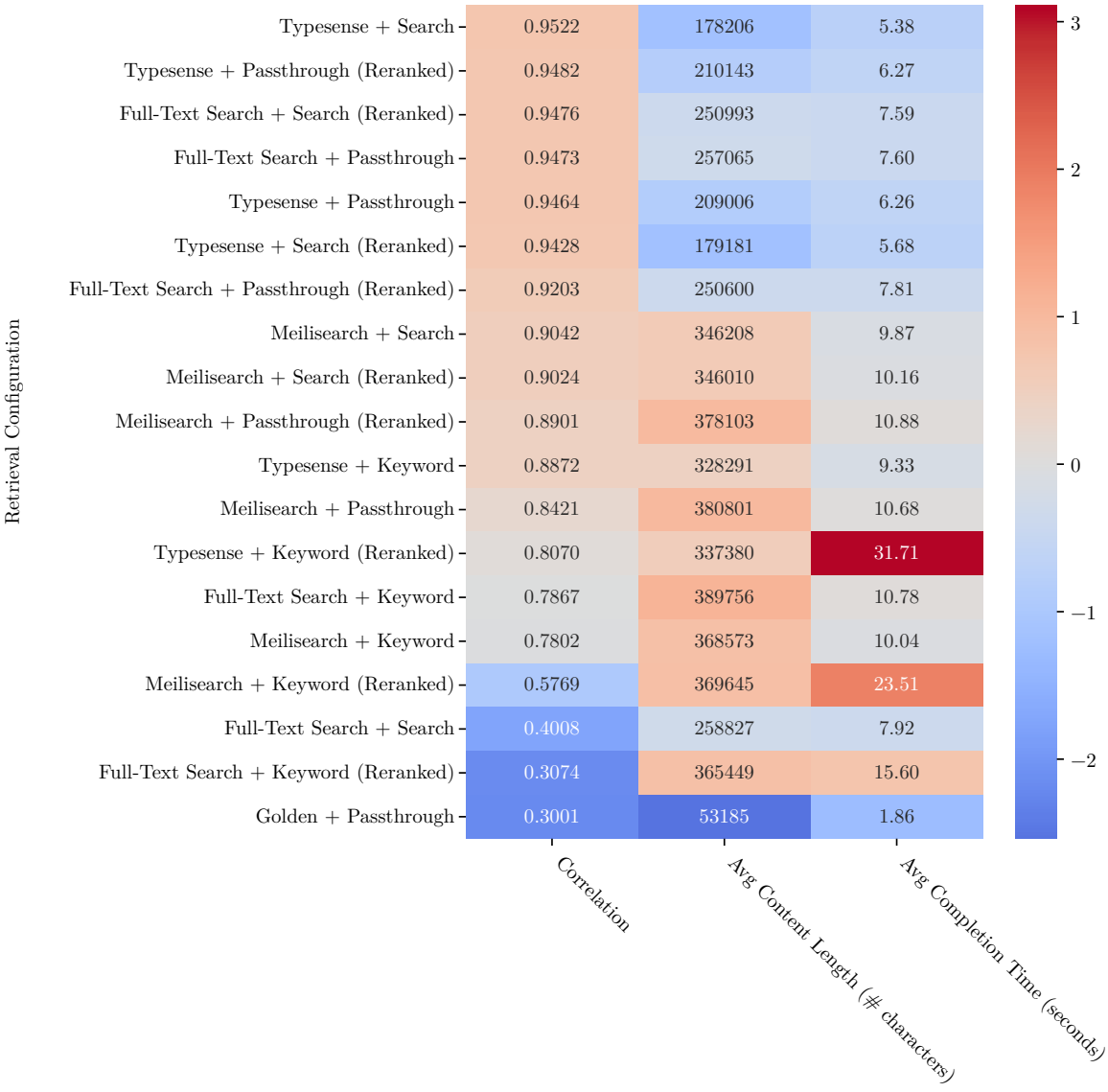

Figure 6: Pearson correlation between average content length and average completion time per retrieval configuration. Content length is defined as the sum of character counts across all documents used to answer a question. Chunk-based retrieval configurations, including embedding-based approaches, are excluded because character counts reflect full document lengths rather than the subset of text contained in retrieved chunks.

The analysis in Figure 6 reveals that document return volume does not systematically explain performance differences between retrieval methods. No clear correlation between average content length and completion time is visible for Typesense or Meilisearch configurations. While the Typesense keyword search with reranking configuration shows a high correlation of 0.8070, this does not explain why this configuration's completion times are approximately three times higher than other configurations with comparable or greater retrieved content. The causal mechanism for these timing differences remains unclear; infrastructure variability from shared GPU resources may be a contributing factor. This finding suggests that the observed accuracy

differences between full-text search and embedding-based retrieval are not confounded by systematic differences in document return counts.

Golden and None retrievers expectedly demonstrated the fastest performance since they spent practically no time in retrieval at all and only need to return the correct or no document.

Looking at observed retrieval times per dataset, the results varied considerably across datasets and configurations. In the measurements, NaturalQuestion exhibited the longest average response time at 40 seconds with the Typesense keyword search and reranking. The observed time with Meilisearch for the same dataset resulted in observed averages of 17.6 and 8.5 seconds with and without reranking.

MultiFieldQA, which achieved the highest overall score, had an observed average response time of 9.6 seconds for the best-performing configuration (Meilisearch with keyword and no reranking). With reranking, the otherwise same configuration returned an average of 32 seconds.

4.3 Performance by Dataset

Examining query performance on a per-dataset basis helps identify whether some document collections yield better results than others, irrespective of the retrieval method used. It is expected that all datasets will show improvement with retrieval compared to the None baseline, though some may benefit more than others. The results might show that some datasets work better with embeddings and some work better with full-text search.

Table 4 summarizes performance across datasets, revealing substantial performance variation across datasets. Because the results represent aggregates across all configurations, the results are lower than the best results from Table 3. HotpotQA achieved the highest correctness at 33.21%, while NarrativeQA achieved the lowest at 9.60%. This represents a substantial performance gap across different datasets.

The performance distribution across datasets suggests that certain question types or document structures are more amenable to retrieval-augmented generation than others. It could also mean that the quality of documents in the dataset is higher for some than others.

4.4 Top 3 Configurations by Dataset

Going deeper into the results from Section 4.3, the top 3 configurations by dataset will reveal whether full-text search works better than embedding-based search in general or only for a certain type of questions. Since the overall results turned out to favor full-text search (Meilisearch with keyword generation), it is expected that this pattern continues when investigating the top 3 configuration by dataset.

Because it would be impractical to look at all results across all datasets at once, only the top 3 have been selected. Individual results for all datasets and all configurations can be found in Appendix A.4.

Dataset	Rank	Search Engine	Retriever	Reranked	Doc Recall	% Correct	% Partial	% Correct+Partial	Avg Retrieval (s)	Avg Completion (s)	Avg Response (s)
2WikiMultihopQA	-	Golden	Passthrough	No	100.00%	73.82%	1.87%	75.69%	0.0078	1.8018	1.8097
	-	None	Passthrough	No	0.00%	8.73%	0.62%	9.35%	0.0000	0.9279	0.9279
	1	Typesense	Keyword	No	82.67%	62.59%	1.50%	64.09%	0.8358	8.7734	9.6093
	2	Meilisearch	Search	Yes	78.38%	58.88%	1.25%	60.12%	2.5100	8.7439	11.2540
	3		Search	No	79.18%	58.48%	1.12%	59.60%	0.0979	8.6500	8.7479
Coursera	-	Golden	Passthrough	No	100.00%	37.04%	42.59%	79.63%	0.0048	1.5833	1.5880
	-	None	Passthrough	No	0.00%	20.37%	46.30%	66.67%	0.0000	1.1910	1.1910
	1	Typesense	Keyword	Yes	50.00%	35.19%	42.59%	77.78%	6.1540	22.3476	28.5016
	2	Embeddings	Fulldocs	No	48.15%	35.19%	38.89%	74.07%	0.4420	8.3087	8.7509
	3	Meilisearch	Keyword	Yes	72.22%	33.33%	42.59%	75.93%	4.0349	23.6872	27.7221
HotpotQA	-	Golden	Passthrough	No	100.00%	85.38%	1.36%	86.74%	0.0086	1.9872	1.9959
	-	None	Passthrough	No	0.00%	9.79%	0.87%	10.66%	0.0000	1.3115	1.3115
	1	Typesense	Search	Yes	76.50%	64.50%	1.50%	66.00%	1.1555	4.2944	5.4500
	2		Search	No	77.20%	62.83%	0.87%	63.69%	0.1213	4.1882	4.3095
	3		Passthrough	Yes	73.00%	62.38%	0.88%	63.25%	2.0931	4.7807	6.8739
MultiDoc2Dial	-	Golden	Passthrough	No	100.00%	53.80%	13.92%	67.72%	0.0028	1.5098	1.5127
	-	None	Passthrough	No	0.00%	1.90%	1.90%	3.80%	0.0000	0.8614	0.8614
	1	Meilisearch	Keyword	Yes	65.82%	37.97%	12.66%	50.63%	3.8125	24.7262	28.5387
	2		Keyword	No	64.56%	34.81%	14.56%	49.37%	0.2984	8.2198	8.5182
	3	Typesense	Keyword	Yes	51.90%	33.54%	12.66%	46.20%	10.5234	16.6333	27.1570
MultiFieldQA	-	Golden	Passthrough	No	100.00%	85.12%	9.09%	94.21%	0.0050	1.6034	1.6085
	-	None	Passthrough	No	0.00%	1.65%	0.00%	1.65%	0.0000	0.7428	0.7428
	1	Meilisearch	Keyword	No	82.64%	73.55%	4.96%	78.51%	0.2541	9.3839	9.6380
	2		Keyword	Yes	83.47%	66.12%	7.44%	73.55%	3.4251	28.6092	32.0344
	3	Typesense	Keyword	Yes	66.94%	55.37%	5.79%	61.16%	6.4518	25.2383	31.6901
MuSiQue	-	Golden	Passthrough	No	100.00%	58.08%	2.10%	60.17%	0.0099	2.7316	2.7416
	-	None	Passthrough	No	0.00%	8.38%	1.23%	9.62%	0.0000	1.3640	1.3640
	1	Meilisearch	Search	Yes	50.75%	40.50%	1.62%	42.12%	2.7113	11.6012	14.3124
	2	Typesense	Search	Yes	44.75%	40.00%	1.75%	41.75%	2.0991	7.1443	9.2435
	3		Search	No	45.75%	38.35%	2.10%	40.44%	0.2193	6.7612	6.9805
NarrativeQA	-	Golden	Passthrough	No	100.00%	43.23%	5.34%	48.57%	0.0315	2.4032	2.4347
	-	None	Passthrough	No	0.00%	1.36%	0.00%	1.36%	0.0000	0.7501	0.7501
	1	Meilisearch	Passthrough	No	40.67%	18.41%	2.11%	20.52%	0.2267	8.4513	8.6781
	2		Passthrough	Yes	44.12%	17.75%	3.75%	21.50%	5.9395	9.4647	15.4044
	3		Keyword	Yes	42.62%	17.75%	2.62%	20.38%	4.8547	17.1963	22.0510
Naturalquestion	-	Golden	Passthrough	No	100.00%	66.38%	3.99%	70.37%	0.0120	1.8048	1.8168
	-	None	Passthrough	No	0.00%	13.68%	3.70%	17.38%	0.0000	0.9976	0.9976
	1	Meilisearch	Keyword	Yes	84.05%	51.85%	5.70%	57.55%	3.9187	13.7215	17.6403

Dataset	Rank	Search Engine	Retriever	Reranked	Doc Recall	% Correct	% Partial	% Correct+Partial	Avg Retrieval (s)	Avg Completion (s)	Avg Response (s)
	2		Keyword	No	82.05%	49.00%	6.27%	55.27%	0.2783	8.3035	8.5819
	3	Embeddings	Fulldocs	Yes	70.09%	48.15%	5.70%	53.85%	10.0871	11.9983	22.0855
QASPER	-	Golden	Passthrough	No	100.00%	68.90%	15.85%	84.76%	0.0044	1.5973	1.6017
	-	None	Passthrough	No	0.00%	0.24%	0.12%	0.37%	0.0000	0.7120	0.7120
	1	Meilisearch	Keyword	Yes	33.38%	26.62%	10.25%	36.88%	2.9298	8.4380	11.3678
	2		Keyword	No	34.02%	25.85%	10.00%	35.85%	0.1571	7.4543	7.6114
	3	Typesense	Keyword	Yes	23.88%	24.75%	7.50%	32.25%	3.7813	20.8751	24.6565
QuALTY	-	Golden	Passthrough	No	100.00%	71.81%	0.74%	72.55%	0.0060	1.6074	1.6135
	-	None	Passthrough	No	0.00%	4.29%	0.00%	4.29%	0.0000	0.7746	0.7746
	1	Meilisearch	Keyword	Yes	53.62%	44.38%	0.12%	44.50%	4.0963	11.4563	15.5526
	2		Keyword	No	50.98%	40.20%	0.37%	40.56%	0.3843	9.1129	9.4972
	3		Search	Yes	37.38%	33.62%	0.50%	34.12%	3.3260	6.2848	9.6108
TOEFL-QA	-	Golden	Passthrough	No	100.00%	88.89%	7.24%	96.13%	0.0011	1.1093	1.1105
	-	None	Passthrough	No	0.00%	19.60%	3.12%	22.72%	0.0000	0.8422	0.8422
	1	Embeddings	Fulldocs	Yes	71.00%	65.00%	5.75%	70.75%	22.8609	5.1679	28.0291
	2		Chunk	No	71.16%	63.05%	5.87%	68.91%	0.2161	1.2062	1.4224
	3		Chunk	Yes	71.12%	62.50%	6.12%	68.62%	0.2963	1.5969	1.8932

Table 6: The top 3 Search engines with the highest % Correct grouped per dataset, including the Golden and None results for reference. Doc Recall shows the percentage of questions where the gold document was retrieved. Values marked in dark green are the best overall, values in light green are the 2nd best overall, values marked dark red are the worst overall, values in light red are the 2nd worst overall. For Doc Recall, % Correct, % Partial, % Correct + Partial, higher is better, for average retrieval time, average completion time, and average response time, lower is better. Marked values do not include the Golden and None results.

Table 6 presents the three best-performing search engine configurations for each dataset, additionally the Golden and None baselines for reference. The results show variation in performance across different datasets and configurations.

It is directly clear that Typesense and Meilisearch consistently rank among the top performers across most datasets. This is in line with the results from Table 3.

MultiFieldQA delivered the best overall result across all datasets. Meilisearch with keyword search and without reranking achieved 73.55% correct answers, approaching the Golden baseline of 85.12% for this dataset. It achieved 66.12% correct answers with keyword search and reranking as the second best, showing a gap of 7.43% to the configuration without reranking. This suggests that reranking may actually harm performance in certain contexts.

Typesense with keyword search and reranking reached 55.37% correct answers as the third-best configuration for this dataset.

NarrativeQA resulted in the worst overall score for a dataset with only 18.41% correctness with Meilisearch direct search without reranking. This indicates that NarrativeQA represents a particularly challenging dataset for retrieval-augmented generation.

However, even with Golden retrieval, this dataset only performed at 43.23% correctness and is thus the second-worst for Golden retrieval, indicating a challenging dataset in general for LLMs, not entirely only for RAG.

For the Coursera dataset, the number of fully correct answers with the Golden baseline was 37.04%, while the best search engine (Typesense with keywords and reranking) achieved 35.19%, demonstrating close proximity to the baseline. Notably, the None baseline for the same dataset achieved 20.37% correct answers, suggesting that this dataset contains questions that are already partially represented in the world knowledge of the used `gpt-oss-120b` LLM.

While Meilisearch and Typesense dominate the top 3 results across datasets, only the results from the TOEFL-QA dataset could be answered better with an embeddings approach with a score of 65.00% for full-document embeddings with reranking. Chunk-based embeddings without reranking achieved 63.05% correct answers. With reranking, the same chunk-based configuration achieved 62.50%, about 0.55% lower than without reranking. In the same dataset, the best performing configurations for Meilisearch and Typesense (both with keyword search and reranking) achieved only a score of 46.6% and 45.6%, showing a huge gap of roughly 18% between embeddings and full-text search approaches.

On the Coursera dataset, embedding-based search with full documents returned performed very similar to Typesense and Meilisearch, each with keyword search reranked.

The highest percentage of partially correct answers was observed in the Coursera dataset, where Meilisearch with keywords and without reranking achieved 46.30% partially correct responses. When combining correct and partially correct results (% Correct + Partial), this configuration reached 77.78%, approaching the Golden baseline of 79.63% for this dataset for % Correct + Partial.

To highlight the difference of different results per dataset, Figure 7 shows the performance per dataset for the overall best, Meilisearch with keyword enhancement and reranked. It becomes clear that there are huge performance differences between the best and worst datasets and between the datasets overall.

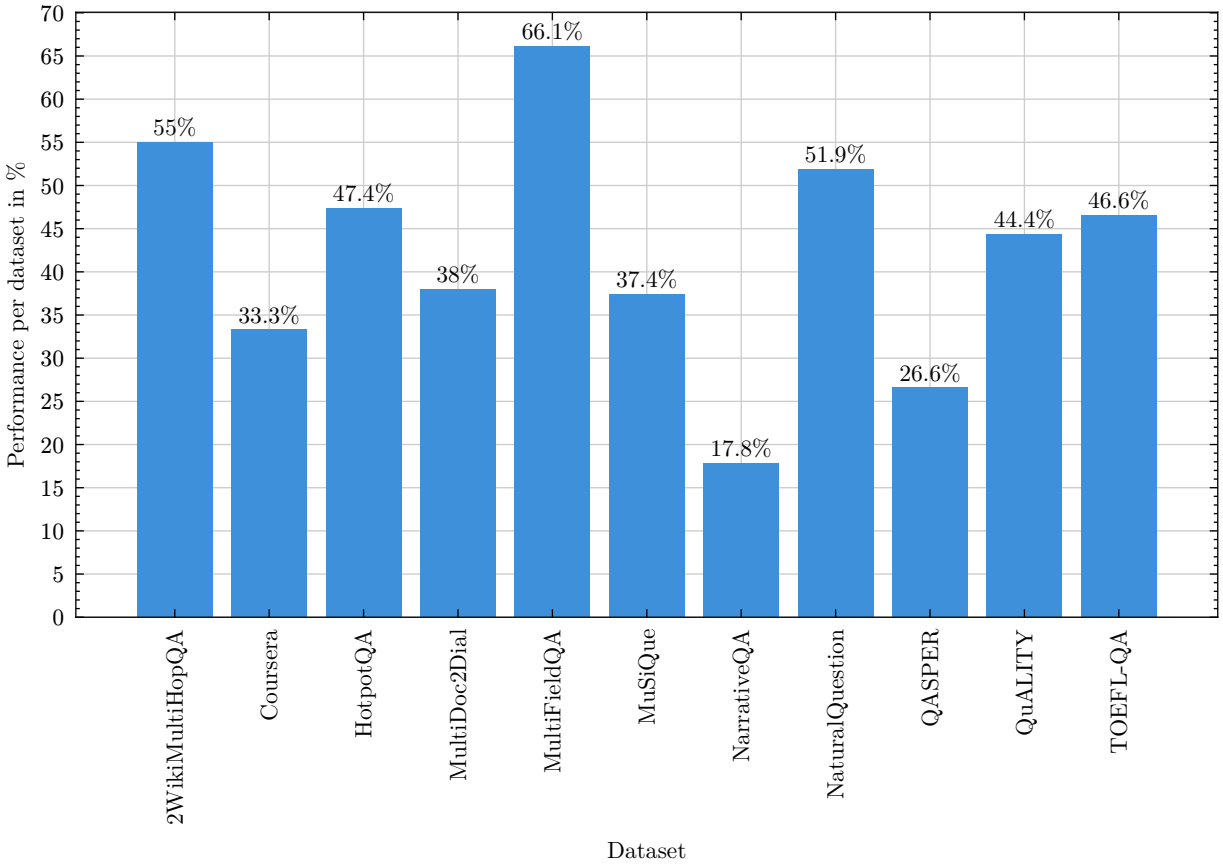

Figure 7: The performance per dataset for the overall best retrieval configuration as shown in Table 3: Meilisearch with keyword search reranked.

4.4.1 Baseline Performance With Perfect Retrieval

The gap between Golden retrieval and best-performing retrieval configurations varied considerably across datasets. Figure 8 shows this difference, plotted by dataset.

It is expected that larger gaps indicate more room for improvement in retrieval quality. This helps contextualize whether current retrieval performance is already perfect, or if substantial improvements can still be made to achieve retrieval quality.

Most notable are the results from the QASPER and Coursera dataset, where QASPER has the biggest difference of 42.28%, and Coursera has only 1.85% difference.

The other datasets perform between 10% and 30%.

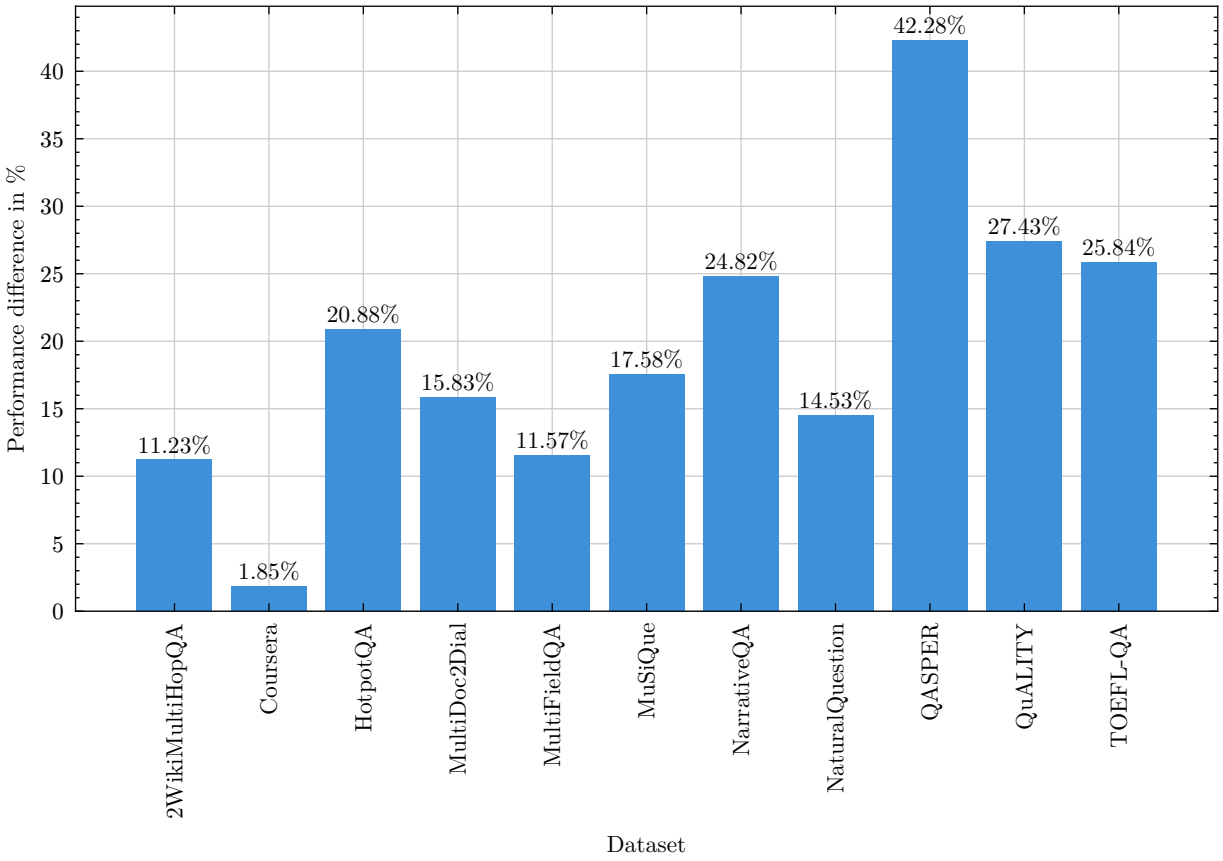

Figure 8: The performance difference from % Correct between Golden and best-performing configuration, per dataset.

4.4.2 Baseline Performance Without Retrieval

To isolate the contribution of retrieval to question-answering performance, the baseline results obtained with the None retriever were examined. Certain data sets demonstrated substantial performance even without any retrieval (None baseline). Should retrieval prove unnecessary for generating satisfactory results, the entire RAG architecture could be eliminated, greatly simplifying the system. The None baseline is evaluated to determine whether this is the case.

Coursera and TOEFL-QA scored around 20% correctness, closely followed by NaturalQuestion with 13.68% and HotpotQA with almost 10%. 2WikiMultiHopQA and MuSiQue both scored around 8%. QuALITY sits in the middle with 4.29% retrieval correctness. Finally, MultiDoc2Dial, MultiFieldQA, NarrativeQA and QASPER scored only around 1%, indicating that performance can be enhanced by a lot with retrieval versus without retrieval.

4.5 Overlap in Correctly Retrieved Documents Between Search Engines

To determine the potential for hybrid retrieval approaches which combine the results from multiple search configurations, the overlap in correctly retrieved documents between different search engine configurations was analyzed using the Jaccard index. The Jaccard index measures the similarity between two sets as the size of their intersection divided by the size of their union, ranging from 0 (no overlap) to 1 (complete overlap).

Pairwise Jaccard statistics across all search engine configurations show huge variation: the average Jaccard index is 0.230, with a minimum of 0.002 and a maximum of 1[12]. The low average indicates that different search engine configurations retrieve different documents correctly, suggesting that combining engines could improve overall retrieval performance.

The best single engine configuration, Meilisearch with Keyword search and no reranking, retrieved the correct document for 3,570 questions. When combining all engine configurations (the union of correctly retrieved documents), 5,616 questions had the correct document retrieved by at least one configuration. This represents 89.4% of the total 6,284 questions and indicates a theoretical potential improvement of 33.15% over the best single engine.

Table 7 presents the top 30 search engine combinations ranked by potential retrieval benefit. The benefit percentage measures the gain from combining two engines compared to using the better one alone. It is calculated as:

$$\text{Benefit} = \frac{\text{union size} - \max(\text{size}_A, \text{size}_B)}{\max(\text{size}_A, \text{size}_B)} \times 100$$

Unlike the Jaccard index, which only measures set similarity, the benefit percentage directly answers how many additional correct retrievals a combination would provide over using the best single engine alone.

The highest benefit of 75.0% is achieved by combining BM25-based full-text search with keyword generation and Typesense chunk retrieval both reranked. Notably, Typesense chunk retrieval performed poorly as an individual configuration (see Section 4.1), yet it retrieves documents that BM25-based full-text search misses. The second-highest benefit of 73.8% comes from the same BM25 configuration combined with Typesense chunk retrieval without reranking. The third-highest benefit of 73.0% combines embeddings with keyword search and BM25 full-text search with keyword generation.

Examining the top 30 combinations shows that embedding-based configurations frequently appear in high-benefit pairings. Combinations of embeddings with various full-text search methods (BM25, Meilisearch, Typesense) consistently show benefits above 55%. This pattern is visible in the benefit percentage heatmap shown in Figure 9.

[12]Combining Chunk-based with and without reranking resulted in a Jaccard index of 1.0, which is not surprising.

Engine 1	Engine 2	Jaccard	Overlap	Exclusive	Benefit
FTS Keyword Reranked	Typesense Chunk Reranked	0.076	99	1210	75.0%
FTS Keyword Reranked	Typesense Chunk	0.075	99	1222	73.8%
Embeddings Keyword	FTS Keyword	0.046	42	868	73.0%
Embeddings Search Reranked	FTS Passthrough Reranked	0.156	469	2533	72.8%
Embeddings Keyword Reranked	FTS Keyword	0.037	33	871	71.9%
Embeddings Search	FTS Passthrough Reranked	0.158	478	2549	71.2%
Embeddings Search Reranked	FTS Search Reranked	0.138	405	2536	69.6%
FTS Keyword Reranked	Meilisearch Chunk Reranked	0.085	114	1221	69.2%
FTS Keyword Reranked	Meilisearch Chunk	0.085	114	1228	68.6%
Embeddings Search	FTS Search Reranked	0.147	432	2516	66.7%
Embeddings Full Docs	Typesense Passthrough	0.222	758	2657	63.5%
Embeddings Full Docs	Typesense Passthrough Reranked	0.226	777	2658	61.8%
Embeddings Full Docs Reranked	Meilisearch Passthrough	0.234	888	2914	60.7%
FTS Keyword	Typesense Chunk Reranked	0.062	74	1126	60.4%
FTS Keyword	Typesense Chunk	0.061	74	1138	59.5%
Embeddings Search Reranked	FTS Search	0.117	322	2436	59.1%
Embeddings Search Reranked	FTS Passthrough	0.114	312	2430	58.1%
Embeddings Search	Typesense Passthrough	0.171	562	2728	57.9%
Embeddings Chunk Reranked	Typesense Search	0.262	1043	2932	57.7%
Embeddings Full Docs Reranked	Meilisearch Passthrough Reranked	0.244	911	2818	57.6%
Embeddings Search Reranked	Typesense Passthrough	0.163	535	2748	57.5%
Embeddings Search	FTS Search	0.119	331	2452	57.4%
Embeddings Full Docs Reranked	Typesense Search Reranked	0.250	966	2901	56.7%
Embeddings Chunk	Typesense Search	0.265	1055	2932	56.7%
Embeddings Chunk Reranked	Typesense Search Reranked	0.263	1039	2910	56.6%
Embeddings Chunk	Typesense Search Reranked	0.262	1039	2934	56.1%
Embeddings Search	Typesense Passthrough Reranked	0.174	578	2735	56.1%
Embeddings Search	FTS Passthrough	0.120	330	2428	56.0%
Embeddings Full Docs Reranked	Typesense Search	0.249	970	2923	55.9%
FTS Keyword	Meilisearch Chunk Reranked	0.071	87	1141	55.6%

Table 7: Top 30 search engine combinations ranked by potential retrieval benefit. Overlap shows questions where both engines retrieved the correct document. Exclusive shows questions where only one of the two engines retrieved the correct document. Benefit percentage indicates improvement over the better single engine.

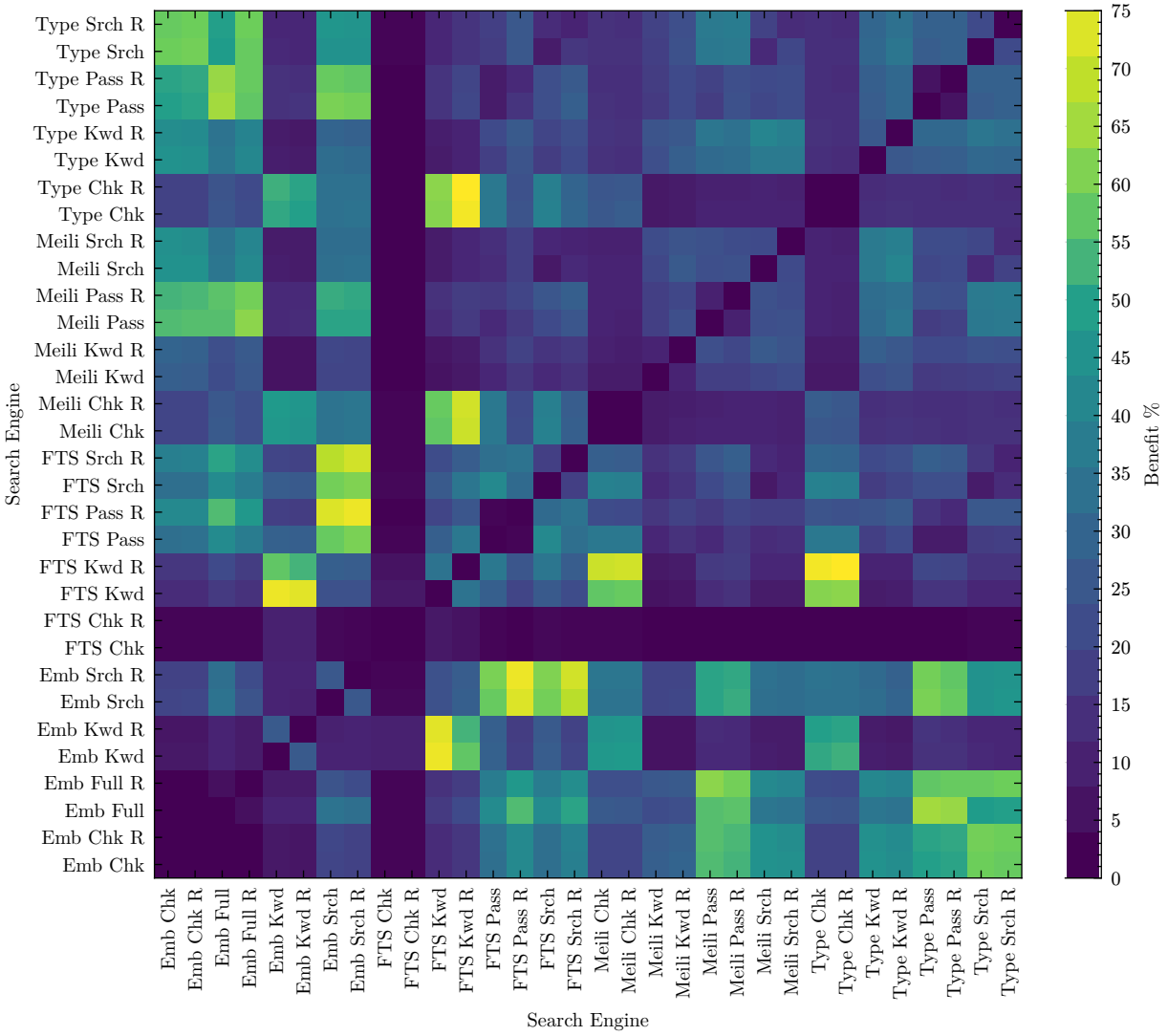

Figure 9: Benefit percentage heatmap showing potential gain from combining pairs of search engines. Abbreviations: Emb=Embeddings, FTS=Full-Text Search, Meili=Meilisearch, Type=Typesense, Chk=Chunk, Kwd=Keyword, Pass=Passthrough, Srch=Search, Full=Full Document, R=Reranked.

The heatmap reveals that most pairwise combinations provide limited benefit, with the majority of cells showing low percentages near zero. Few combinations exceed 50% benefit. Full-text search with chunk retrieval shows uniformly low benefit when combined with any other configuration, consistent with the poor individual performance of this configuration observed in Section 4.1.

The combinations of embedding-based methods (the 8 lower rows) with full-text search methods (left two third columns) show consistently higher benefits than combinations within the same retrieval paradigm. Embedding-based search with keyword generation in some cases shows lower benefit when combined with full-text search methods, as both approaches rely on keyword matching.

4.6 Conclusion

This chapter compared full-text search and embedding-based retrieval across eleven datasets. The overall results (Section 4.1) established performance bounds: the Golden retriever achieved

69.40% correctness, while the None retriever baseline achieved 7.65% from parametric knowledge alone. Meilisearch with keyword search and reranking achieved the best overall performance at 40.44% correctness.

Full-text search outperformed embeddings on all datasets except TOEFL-QA, with Coursera as a tie, and an average improvement of 13.50%. The gap was particularly pronounced on multi-hop reasoning datasets such as 2WikiMultiHopQA (43.21% difference) and HotpotQA (32.63% difference). Chunk-based retrieval with Meilisearch and Typesense fell below the None baseline, indicating that incorrect chunks actively degrade LLM performance.

The per-dataset analysis (Section 4.3, Section 4.4) revealed substantial variation across datasets and retrieval configurations, ranging from 73.55% for Meilisearch with keyword search and no reranking for the MultiFieldQA dataset to 18.41% for Meilisearch with passthrough and no reranking in the NarrativeQA dataset. Reranking produced mixed results, improving some configurations while reducing correctness in others.

The overlap analysis (Section 4.5) demonstrated that different retrieval methods correctly retrieve different documents, with an average Jaccard index of 0.230. Combining all configurations theoretically achieves 89.4% document recall, suggesting substantial potential for hybrid retrieval approaches.

These findings demonstrate that full-text search represents a viable and often superior alternative to embedding-based retrieval for RAG systems.

5 Discussion

This chapter interprets the experimental results from the previous chapter.

It begins with the overall finding on full-text search performance in Section 5.1, then examines dataset-specific variations (Section 5.2), reranking effects (Section 5.3), timing implications (Section 5.4), chunk-based retrieval failures (Section 5.5), document recall and generation failure (Section 5.6), retrieval dependency (Section 5.7), and the potential for hybrid retrieval (Section 5.8). The chapter concludes with practical recommendations in Section 5.9.

5.1 Full-Text Search Performance Compared to Embeddings

The results support the hypothesis that full-text search can perform better in a RAG-Setting than relying on searching through embeddings, though this finding is not universal across all datasets. Specialized full-text search databases demonstrate competitive or superior performance relative to embedding-based approaches.

With appropriate query formulation, these dedicated full-text search engines substantially outperform embedding-based retrieval. Meilisearch with keyword search generation achieved 40.44% correctness compared to the best embedding method at 31.19% correctness, a 9.25% improvement. Importantly, both full-text search and embedding-based retrieval were tested with equivalent query preprocessing strategies, including keyword generation. While full-text

search benefits substantially from keyword preprocessing, embedding-based retrieval actually performed worse with keyword generation (7.15% correctness) than with direct chunk retrieval (24.59% correctness). This indicates that the advantage of full-text search with preprocessing is not due to embeddings lacking equivalent preprocessing. Full-text search appears to be fundamentally better suited to leverage LLM-generated keywords as search terms.

A notable exception is the TOEFL-QA dataset, where embedding-based retrieval outperformed full-text search (65.00% vs. 46.62%), demonstrating that semantic search retains advantages for certain query types requiring deeper language understanding. This exception is discussed further in Section 5.2 and Section 5.9.

Both Typesense and Meilisearch, which implement search and ranking algorithms beyond traditional BM25, demonstrated at least comparable performance to SQLite-based full-text search with BM25, validating their utility as retrieval backends for RAG systems. When using passthrough or search query modes, the observed ~4% improvement (29.30%–29.60% vs. 25.27%) is modest enough that it may not generalize without further statistical testing, which is beyond the scope of this thesis. However, when combined with keyword generation, the advantage of modern search engines becomes more pronounced, with Meilisearch achieving 40.44% compared to BM25′s best result of 16.12%.

Simple full-text search with BM25 achieves comparable performance of 25.27% when combined with reranking compared to embeddings chunk retrieval at 24.59%, with even lower complexity than having to use an external database like Meilisearch or Typesense.

With embedding-based retrieval, the results vary slightly when the full document was passed to the LLM. With the full document, a 28.58% correctness was achieved, vs. 24.59% with only retrieved chunks (both without reranking). This marginal difference indicates the LLM can generate better answers when it has the full document, but the improvement is not as good as what other databases achieve.

The difference in Document Recall between full document and chunks can be explained by the possibility that retrieving full documents may result in a longer total context that pushed actually relevant documents out of the final list, particularly when those relevant documents were ranked lower in the retrieval order.

5.2 Dataset-Specific Performance Variation

The substantial performance variation across datasets (overall worst of 18.41% in NarrativeQA vs overall best of 73.55% in the MultiFieldQA dataset, each with Meilisearch and the best result per dataset) indicates that retrieval-augmented generation effectiveness depends heavily on dataset characteristics. This suggests that certain types of questions, document structures, or reasoning requirements are more amenable to retrieval augmentation. Understanding these characteristics when building RAG systems could inform both system design and dataset selection for future work.

For the NarrativeQA dataset, the Golden retriever achieves 43.23% correct answers (the lowest Golden baseline among all datasets), and the best-performing search engine reaches only 18.41% correctness. This dataset shows poor suitability for retrieval-based question answering overall. The low performance for the best overall configuration suggests that NarrativeQA involves contexts that are too large or complex for effective retrieval, or that the nature of narrative questions requires different retrieval strategies than those employed in this thesis.

5.3 Reranking Effects

The marginal or sometimes negative effects of reranking (particularly for Meilisearch and Typesense keyword search) suggest that modern search engines may already provide sufficiently good ranking. For MultiFieldQA, Meilisearch with keyword search and reranking (66.12% correct) performed worse than the same configuration without reranking (73.55% correct), potentially by demoting relevant documents.

Other reranking approaches may yield different results, though Jina Reranker v3 represents current state-of-the-art performance. The marginal improvements observed suggest that for full-text search engines like Meilisearch and Typesense, the initial ranking quality is already sufficiently high that reranking provides limited additional value.

On the other hand, reranking improved results for BM25-based full-text search implementations by a moderate 2.81%, from 22.46% to 25.27% and for embeddings with full documents from 28.58% to 31.19%. This indicates that reranking may still provide a little value when the initial retrieval ranking is less sophisticated, as is the case with traditional BM25 implementations.

5.4 Timing Implications

As noted in Section 4.2, the timing analysis presented here is descriptive, and results should be interpreted as observed patterns rather than precise benchmarks.

A potential confounding factor in comparing retrieval methods is that different configurations may return different numbers of documents, which could affect both accuracy and timing. To control for this, the Pearson correlation between average content length and completion time was analyzed (Figure 6). The analysis reveals no systematic relationship between document return volume and performance differences across retrieval methods. The notably longer completion times observed for the best-performing configurations (Meilisearch and Typesense with keyword search) do not appear to be related to more documents being passed to the LLM in these configurations.

A possible factor to explain these higher response times is infrastructure load. Because the experiment relied on a university-hosted instance, other users accessing the same GPU resources simultaneously may have introduced variability in response times.

The observed extended average response time for NarrativeQA (22 seconds with Meilisearch) is consistent with the hypothesis that context size plays a role, as processing larger contexts

would be expected to require more computational time, though this interpretation is subject to the timing measurement limitations noted above.

In the observed measurements, retrieving documents from embeddings appeared faster (roughly 200ms) than retrieving full documents from an external search engine like Meilisearch.

5.5 Failures of Chunk-Based Retrieval with Full-Text Search

The particularly poor performance of chunk-based retrieval with Meilisearch and Typesense below the None baseline represents an important failure mode specific to full-text search engines. When incorrect chunks are retrieved, they seem to actively mislead the LLM, resulting in worse performance than providing no retrieval augmentation at all.

Notably, this problem does not affect embedding-based retrieval: chunks with embeddings achieved 24.59% correctness. A plausible hypothesis for this difference relates to how each retrieval method operates. Embedding-based search captures semantic meaning from text, allowing smaller chunks to be matched based on conceptual similarity. Full-text search, however, relies on keyword matching and requires sufficient surrounding context to ensure relevant terms appear together in the indexed text. When documents are chunked too aggressively for full-text search, individual chunks may lack the keyword density needed for accurate matching, leading to retrieval of irrelevant passages. This explanation is consistent with the observed behavior but has not been empirically validated through keyword density analysis.

For practical RAG implementations using full-text search, indexing complete documents or larger passages appears preferable to fine-grained chunking strategies that work well with embeddings.

5.6 Document Recall and Generation Failure

The document recall results reveal a distinction between retrieval failure and generation failure. Even the Golden retriever, with perfect document recall, only achieves 69.40% answer correctness. This indicates that approximately 30% of incorrect answers in the best retrieval case result from generation failure rather than retrieval failure. The LLM fails to extract the correct answer even when provided with the relevant document.

This finding has large implications for RAG system optimization. Improving retrieval quality can only address errors caused by missing relevant documents. The substantial portion of errors attributable to generation failure requires different interventions, such as improved prompting strategies, better context presentation, or more capable generation models.

The relationship between document recall and answer correctness also differs markedly between retrieval methods (see Figure 4). Embedding-based configurations show a flatter relationship: even when document recall increases substantially (from approximately 7% to 40%), the percentage of correct answers does not increase proportionally, remaining in the 17-25% range. In contrast, full-text search engines (BM25-based SQLite, Meilisearch, Typesense) show

a steeper, more consistent positive relationship, continuing to achieve higher correctness rates as recall increases to 40%.

This pattern suggests that full-text search not only retrieves documents more effectively but also retrieves them in a form more amenable to answer extraction. One possible explanation is that full-text search always returns the full document and the used LLM is sufficiently capable of extracting the correct facts required for the answer from the whole document. Embedding-based retrieval may return semantically related documents where the answer is expressed in different terms, requiring more sophisticated reasoning to connect the query to the answer.

The difference in recall between reranking configurations can differ because the reranker operates on all documents returned by the initial retrieval stage and re-orders them based on relevance scores before selecting only the top k documents[13]. If the gold document was initially retrieved but ranked outside the top k positions, reranking may either promote it into the final set, improving recall, or fail to do so. Conversely, if the gold document was in the top 10 before reranking, reranking could potentially push it out if other documents (wrongly) score higher.

5.7 Retrieval Dependency per Dataset

When looking at the results for the retrieval of the None baseline, two distinct categories are visible:

High baseline performance datasets (e.g., TOEFL-QA and Coursera) achieved around 20% correct answers without any retrieval, indicating that these questions may already be well-represented in the LLMs training data or represent question types that the model handles well inherently. These datasets show limited benefit from retrieval augmentation, as the relative improvement is constrained by the already substantial baseline.

Low baseline performance datasets (e.g., MultiFieldQA with 1.65% None baseline) demonstrate strong dependence on retrieval, with search engines providing substantial performance gains over the baseline. These represent scenarios where external knowledge is critical for answering questions.

Of note here is that the dataset was originally filtered with `GPT-4o` but the experiment has been run with the newer `gpt-oss-120b`. Both have different knowledge cutoff times, leading to the conclusion that the latter model has more knowledge than `GPT-4o`.

TOEFL-QA achieved the highest performance with the Golden retriever at 88.89% correct answers, though this must be interpreted in light of its already strong 19.60% None baseline. Since the TOEFL-QA dataset tests for English understanding and text generation rather than only QA performance [53], this indicates the strong results may be attributed to the advanced linguistic abilities of the used `gpt-oss-120b` LLM. Current models demonstrate sufficient proficiency in English understanding to already perform well on this benchmark without requiring retrieval, which could explain the observed high performance in the experimental results.

[13] In the experiment in this thesis, up to 10 documents were ultimatly returned

MultiFieldQA demonstrates the best balance of high absolute performance (85.12% Golden, 73.55% best search engine) combined with low baseline performance (1.65%), indicating genuine value added by retrieval.

Across datasets, Meilisearch and Typesense consistently appeared as top-performing search engines, with keyword-based search generally outperforming other retrieval methods. The effectiveness of different configurations varied by dataset, suggesting that optimal retrieval strategies may be task dependent.

5.8 Potential for Hybrid Retrieval

The overlap analysis in Section 4.5 reveals that different search engine configurations retrieve different documents correctly. The low average Jaccard index of 0.230 indicates that combining engines could improve overall retrieval performance. This complementary behavior suggests that full-text search and embedding-based retrieval may capture different aspects of relevance, making hybrid approaches attractive.

The theoretical upper bound shows that combining all configurations could correctly retrieve documents for 89.4% of questions, compared to 56.25% for the best single configuration. While achieving this theoretical maximum is unrealistic in practice, even partial combinations show substantial potential gains. The highest-benefit pairings (exceeding 70%) combine methods from different retrieval paradigms, often embedding-based methods with full-text search. This becomes very clear when examining the top 30 combinations as shown in Table 7, the list is dominated by combinations which include embedding-based retrieval methods.

This pattern is explained by the fundamental difference in how these methods operate. Embedding-based search matches documents based on semantic similarity in vector space, capturing conceptual relationships even when exact keywords differ. Full-text search relies on keyword matching and term frequency, excelling when queries and documents share vocabulary. When one method fails to retrieve the correct document, the other may succeed because it operates on different matching principles.

Importantly, the heatmap in Figure 9 shows that not all combinations are beneficial. Most pairwise combinations provide limited improvement, with many cells near zero. Arbitrary combination of search engines is unlikely to improve results; instead, combinations should be chosen deliberately to leverage complementary strengths. Full-text search with chunk retrieval shows uniformly low benefit regardless of the pairing, consistent with its poor individual performance.

5.9 Assessment and Recommendations

The optimal choice between full-text and semantic search methods depends a lot on the characteristics of both the document corpus and expected query patterns.

Clear performance advantages for full-text search were shown across most tested datasets, with particularly strong results observed when query preprocessing steps were added. Compet-

itive performance by semantic search using embeddings was exhibited only in specific contexts, most notably in the TOEFL-QA dataset, where the advantage is explained by the requirement for deeper semantic understanding of vague or ambiguous queries.

5.9.1 Trade-offs Between Accuracy and Efficiency

The results suggest a trade-off between retrieval quality and response time. Configurations employing retrieval generally achieved higher accuracy but incurred increased latency in the observed measurements. For MultiFieldQA, the observed 23-second increase in average response time (from 9 to 32 seconds) when using retrieval with Meilisearch yielded substantial accuracy improvements, suggesting that this trade-off may be worthwhile for accuracy-critical applications. The appropriateness of this trade-off depends on the specific use case and latency requirements.

5.9.2 Context-Dependent Recommendations

Based the results from the experiment, FTS should be used to implement search in a RAG system when expected queries look for specific facts or concrete knowledge, and documents are suitable for keyword-based full-text search. This is usually the case when searching for specific keywords would yield relevant documents and consistent vocabulary is used in most of the documents.

Because the semantic search alternative to full-text search approaches was outperformed most of the time and with a wide margin across many different datasets, full-text search represents a strong default choice for most practical applications of RAG, particularly those involving factual queries and keyword-friendly document corpora.

On the other hand, embedding-based semantic search has advantages over full-text search when queries are inherently vague or conceptual in nature. In these cases, semantic understanding beyond surface-level keyword matching is required by the documents. This becomes even more visible when a mismatch between search queries and the document corpus is present and context and meaning are more important than exact term matching.

This pattern is illustrated clearly by the TOEFL-QA results, as semantic understanding over keyword matching is favored by the dataset's characteristics.

In any case, an important finding is that incorporating a query preprocessing step, such as keyword generation or query reformulation, significantly improves retrieval performance. Even when searching in a full-text search index with the user query directly produces results similar to embedding-based approaches, the addition of preprocessing creates measurably better outcomes.

5.9.3 Dataset-Specific Considerations

Performance varies considerably across different dataset types, highlighting the importance of evaluating retrieval methods against the documents and queries that are used.

For the best results, practitioners should analyze the structure, vocabulary, and content organization of their specific document collection, then consider how users typically formulate questions and what information needs drive those queries and choose a search engine setup based on that. Comparing this to the datasets tested in the experiment of this thesis helps identify analogous scenarios.

Ideally, based on the assessment of queries and documents, a small evaluation dataset should be curated to validate performance for the use case, effectively re-running the experiment of this thesis on a smaller scale. The results of the experiment can then be used to compare multiple different implementations against each other to make an informed decision about the architecture required.

The NarrativeQA dataset presents an edge case worth noting. Its combination of large document sizes and complex narrative structure poses challenges for both retrieval paradigms. This requires alternative approaches beyond the straightforward architecture that was tested in this thesis.

5.9.4 Limitations and Caveats

These recommendations apply within the scope and constraints of the presented experimental design.

5.9.4.1 Single Model Dependency

Importantly, all experiments were conducted using a single LLM (`gpt-oss-120b`) for generation, query preprocessing, keyword extraction, and answer evaluation. The relative performance of full-text search versus embedding-based retrieval may differ with other model families (e.g., Claude, Llama, Gemini), smaller or larger models, or models with different training data and capabilities. Similarly, only one embedding model (`Qwen3-4B`) was tested for semantic search; other embedding models such as OpenAI's text-embedding may yield different results. The findings reported here are therefore specific to this model configuration and may not generalize to all RAG implementations without further validation.

5.9.4.2 Evaluation Methodology

The same model (`gpt-oss-120b`) was used for both answer generation and automated evaluation, which could theoretically introduce systematic bias. However, the evaluation was conducted in a stateless manner: each answer was evaluated independently without the model having access to or context of other generated answers. The model received only the question, the correct reference answer, and the generated answer to evaluate, with no information about which retrieval method produced the answer or how other answers were rated. This design mitigates concerns about self-preferential rating, as the evaluation model cannot identify its own outputs or adjust ratings based on retrieval method.

The 84.7% agreement rate with manual labels provides empirical validation that the automated evaluation produces reliable results despite using the same underlying model. Since the

stateless evaluation design excludes method-specific bias, actual measurement error is expected to be random, which softens rather than inflates observed effect sizes. Nonetheless, using separate model families for generation and evaluation in future work could provide additional validation and reduce any potential for shared systematic blind spots in both generation and evaluation.

5.10 Conclusion

While full-text search is strongly favored by the results in the majority of scenarios, the recurring answer to "which method should be used?" remains: it depends. The dependence, however, is systematic and predictable based on document corpus and query characteristics.

Within the scope examined here, the findings are sufficiently clear and interpretable to provide actionable guidance for practitioners designing RAG systems with similar model configurations. For most implementations, full-text search represents a robust default choice, with semantic search reserved for scenarios where semantic understanding demonstrably outweighs the benefits of keyword matching. Practitioners using different LLMs should validate these findings against their specific model configuration before making architectural decisions.

6 Conclusion

This thesis investigated whether full-text search can serve as a viable alternative to embedding-based retrieval in RAG systems. The motivation originated from the considerable infrastructure complexity that embedding-based approaches introduce, including embedding model deployment, vector database management, and similarity search mechanisms. Full-text search, by contrast, offers a more straightforward implementation path where databases handle indexing automatically without requiring external embedding pipelines.

To address this research question, an experiment was conducted using a multi-source dataset comprising 6,284 questions across eleven different QA benchmarks. The dataset was originally filtered for different research on Long-Context RAG using `GPT-4o` to identify questions that cannot be answered from world knowledge alone, ensuring that retrieval is genuinely required. Four primary search backends were evaluated: SQLite Full-Text Search with BM25, pgVector for embedding-based semantic search, and the specialized full-text search engines Typesense and Meilisearch. Each search engine was tested with different retriever architectures including direct passthrough, LLM-based query rewriting, keyword generation, and chunk-based retrieval. The `gpt-oss-120b` model served as both the generation model and for query preprocessing tasks.

6.1 Key Findings

The experimental results demonstrate that, within the tested configuration using `gpt-oss-120b` for generation and `Qwen3-4B` for embeddings, full-text search can not only match but outperform embedding-based retrieval. Meilisearch with keyword generation and reranking achieved the

highest overall correctness at 40.44%, compared to 31.19% for the best embedding-based configuration (full-document retrieval with reranking).

Specialized full-text search databases consistently outperformed traditional BM25-based implementations. Typesense and Meilisearch, which employ ranking algorithms beyond term frequency metrics, both achieved 29.60% and 29.30% correctness with direct search compared to 28.58% for embeddings (with full documents) and 25.27% for BM25-based SQLite Full-Text Search. This suggests that modern search engine implementations provide meaningful advantages for retrieval tasks, even when not further optimized.

Query preprocessing emerged as a critical factor for retrieval quality. Combining full-text search with LLM-generated keywords significantly improved results across most configurations.

Embedding-based retrieval was also tested with equivalent keyword preprocessing but did not benefit from it. Keyword-based embedding search achieved only 7.15% correctness compared to 24.59% for direct chunk retrieval. This asymmetry indicates that the advantage of full-text search with preprocessing stems from the fundamental suitability of keyword-based search to leverage LLM-generated terms, rather than from an unfair comparison where only one method received preprocessing.

Reranking with Jina Reranker v3, a state-of-the-art model on the BEIR benchmark, provided only marginal improvements and in some cases degraded performance. For Meilisearch with keyword search, reranking improved correctness by 1.26%. In certain dataset configurations, such as MultiFieldQA, reranking actually decreased performance by 7.43%. Since these results were obtained with a current best-in-class reranking model, they suggest that modern search engines may already provide sufficiently effective ranking, limiting the potential gains from additional reranking steps.

Performance varied considerably across datasets. MultiFieldQA achieved 73.55% correctness with the best configuration, while NarrativeQA reached only 18.41%. The TOEFL-QA dataset represented the sole exception where embeddings outperformed full-text search (65.00% vs. 46.62%), likely due to the semantic nature of language comprehension questions. These variations underscore that optimal retrieval strategies depend on dataset characteristics including document structure, query patterns, and the type of reasoning required.

Chunk-based retrieval with full-text search performed poorly, often falling below the no retrieval baseline. This indicates that incorrect chunking strategies can actively mislead the language model, resulting in worse performance than providing no retrieved context at all.

6.2 Future Work

The overlap analysis in Section 4.5 demonstrates substantial potential for hybrid retrieval approaches. The highest-benefit pairings combine methods from different retrieval paradigms, particularly embedding-based search with full-text search. This suggests that embedding-based semantic matching and keyword-based full-text search capture complementary aspects

of relevance. Future work could investigate practical hybrid retrieval strategies that leverage these complementary strengths without requiring exhaustive combination of all configurations.

Reranking did not yield the expected performance improvement, despite using Jina Reranker v3, a state-of-the-art model. While this suggests the limitation lies in the fundamental interaction between modern search engine ranking and reranking rather than model choice, a systematic evaluation across multiple reranking models could confirm whether alternative approaches might be more effective for specific dataset characteristics or search engine configurations. Additionally, analyzing whether gold documents are demoted by reranking could explain the observed performance degradation in certain configurations.

The chunk sizes used in this experiment (max. 512 characters with 50 character overlap) proved too small for effective full-text search. Future work could systematically test larger chunk sizes (e.g., 2000-4000 characters) to identify whether a threshold exists where full-text search on chunks becomes competitive with full-document retrieval. Such investigation could clarify whether the observed chunk-based failure is specific to the chunk sizes tested or represents a broader limitation, which could impact the relative performance comparison between retrieval methods.

Ablating the inclusion of relevance scores in the generator prompt could quantify whether heterogeneous scoring across search engines influences outcomes.

A further direction for future work is validating these findings across different model families. This thesis used `gpt-oss-120b` exclusively for generation, query preprocessing, and evaluation. Testing with other LLM families such as Claude, Llama, or Gemini would establish whether the observed advantages of full-text search generalize beyond this specific model or are particular to its characteristics. Similarly, evaluating alternative embedding models beyond `Qwen3-4B` could reveal whether the performance gap between full-text search and semantic search varies with embedding quality. Using different models for generation versus evaluation could also provide additional validation of the automated evaluation methodology.

A Appendix

A.1 Answer Prompt

The prompt used by the experiment implementation to answer questions:

```
Use the following information to assist the user:
-------------
{results_str}
-------------
You are an AI assistant that helps people find information. Only use the information
given to you.
Focus on directly providing the answer to the question. You don't need to explain
your answer at length, a very short explanation is sufficient.
Do not make up an answer.
If you do not know the answer to a question, respond by saying verbatim "I do not
know the answer to your question."
```

`{results_str}` is replaced with a concatenated string of the results with the text and score and separated by `---`. A result string can look like this (texts are shortened for brevity):

```
score: 0.88323
text: Alexander M. Patch American High School (also known as "Patch American High
School" or "Patch High School") was an English language high school on Patch Barracks
in Stuttgart, ...
---

score: 0.87303
text: List of NFL franchise post-season droughts Playoff Droughts 0Team0 Last earned
appearance in post-season Seasons Buffalo Bills ^ 1999 AFC Wild Card Playoffs 17
Cleveland Browns ^ 2002 AFC Wild Card Playoffs 14 Los Angeles Rams ^ 2004 NFC
Divisional ...
---

score: 0.6323
text: Houston Astros The Astros clinched their first division title as a member of
the American League West division, and first division title overall since 2001. ...
---

score: 0.559
text: Henry IV (11 November 1050 – 7 August 1106) was Holy Roman Emperor from 1084 to
1105, king of Germany from 1054 to 1105, king of Italy and Burgundy from 1056 to
1105, and duke of Bavaria from 1052 to 1054. ...
```

A.2 Query rewriting prompts

Both the Search Query and Keyword prompts are divided into a System prompt and user message. They are implemented using the `dspy` Python library to simplify the implementation.

In both cases, the `{user_query}` placeholder is replaced at runtime with the input query.

A.2.1 Search Query Prompt

The prompt that is used to rewrite the user query into a more suitable search query.

The system prompt looks like this:

```
Your input fields are:
1. `query` (str):
Your output fields are:
1. `output_query` (str):
All interactions will be structured in the following way, with the appropriate values
filled in.

[[ ## query ## ]]
{query}

[[ ## output_query ## ]]
{output_query}

[[ ## completed ## ]]
In adhering to this structure, your objective is:
        Return a query for searching documents in a database that might contain the
answer to the query.
```

The user message looks like this:

```
[[ ## query ## ]]
{user_query}

Respond with the corresponding output fields, starting with the field `[[ ##
output_query ## ]]`, and then ending with the marker for `[[ ## completed ## ]]`.
```

As an example, for a user query of `Do both films Lifeforce (film) and Via Pony Express have the directors that share the same nationality?` the response might look like this:

```
[[ ## output_query ## ]]
"Lifeforce (film) director nationality" OR "Via Pony Express director nationality"

[[ ## completed ## ]]
```

A.2.2 Keyword Prompt

The Keyword prompt is used to create keywords for search based on the user query.

The system prompt looks like this:

```
Your input fields are:
1. `query` (str):
Your output fields are:
1. `keywords` (list[str]):
All interactions will be structured in the following way, with the appropriate values
filled in.

[[ ## query ## ]]
```

```
{query}

[[ ## keywords ## ]]
{keywords}        # note: the value you produce must adhere to the JSON schema:
{"type": "array", "items": {"type": "string"}}

[[ ## completed ## ]]
In adhering to this structure, your objective is:
        Return a list of keywords for searching documents in a database that might
contain the answer to the query.
```

The user message looks like this:

```
[[ ## query ## ]]
{user_query}

Respond with the corresponding output fields, starting with the field `[[ ## keywords
## ]]` (must be formatted as a valid Python list[str]), and then ending with the
marker for `[[ ## completed ## ]]`.
```

As an example, for a user query of `Do both films Lifeforce (film) and Via Pony Express have the directors that share the same nationality?` the response might look like this:

```
[[ ## keywords ## ]]
["Lifeforce film director", "Via Pony Express director", "director nationality",
"Lifeforce director nationality", "Via Pony Express director nationality", "film
directors nationality comparison", "British director Lifeforce", "American director
Via Pony Express", "film director nationality Lifeforce", "film director nationality
Via Pony Express"]
[[ ## completed ## ]]
```

A.3 Evaluation Prompt

This is the prompt that's produced after optimizing it with `dspy` as outlined in Section 3.5. It is divided into a System prompt and User message.

The System prompt:

```
Your input fields are:
1. `question` (str):
2. `correct_answer` (str):
3. `provided_answer` (str):
Your output fields are:
1. `correctness` (Literal['correct', 'partially_correct', 'incorrect']): ${reasoning}
All interactions will be structured in the following way, with the appropriate values
filled in.

[[ ## question ## ]]
{question}

[[ ## correct_answer ## ]]
```

```
{correct_answer}

[[ ## provided_answer ## ]]
{provided_answer}

[[ ## correctness ## ]]
{correctness}        # note: the value you produce must exactly match (no extra
characters) one of: correct; partially_correct; incorrect

[[ ## completed ## ]]
In adhering to this structure, your objective is:
        Given the fields `question`, `correct_answer`, `provided_answer`, produce the
fields `correctness`.
```

The User message:

```
[[ ## question ## ]]
{question}

[[ ## correct_answer ## ]]
{correct_answer}

[[ ## provided_answer ## ]]
{provided_answer}
```

The placeholders `{question}`, `{correct_answer}` and `{provided_answer}` are replaced with appropriate values at runtime.

A potential user message and response might look like this:

```
[[ ## question ## ]]
What may happen if the VR headset lenses are exposed to sunlight or strong light?

[[ ## correct_answer ## ]]
Exposure to sunlight or strong light may cause permanent yellow spot damage on the
screen.

[[ ## provided_answer ## ]]
Exposure to direct sunlight or strong light may cause permanent yellow spot damage on
the screen. Such screen damage is not covered by the warranty.

Assistant message:

[[ ## correctness ## ]]
correct

[[ ## completed ## ]]
```

A.4 Search Engine Results by Dataset

Column abbreviations:

- **Eng.** (Engine): Emb = Embeddings, FTS = Full-Text Search, Meili = Meilisearch, Type = Typesense, Gold = Golden, None = No retrieval
- **Ret.** (Retriever): P = Passthrough, C = Chunk, K = Keyword, S = Search, F = Full Docs
- **RR** (Reranked): Y = Yes, N = No
- **Retr.**, **Compl.**, **Resp.**: Average retrieval, completion, and response times in seconds

Eng.	Ret.	RR	% Cor.	% Par.	% C+P	Retr.	Compl.	Resp.
Gold	P	N	73.8%	1.9%	75.7%	0.01	1.8	1.8
None	P	N	8.7%	0.6%	9.4%	0.00	0.9	0.9
Emb	C	N	8.4%	0.7%	9.1%	0.23	1.1	1.4
		Y	7.0%	0.9%	7.9%	0.36	1.5	1.9
	F	N	19.4%	1.4%	20.8%	0.58	8.1	8.7
		Y	18.8%	0.9%	19.6%	35.68	7.3	43.0
	K	N	4.5%	0.7%	5.2%	1.12	1.0	2.1
		Y	3.9%	0.8%	4.6%	1.71	1.2	2.9
	S	N	7.0%	0.4%	7.4%	0.24	1.0	1.3
		Y	4.5%	0.2%	4.8%	0.34	1.0	1.3
FTS	C	N	9.9%	0.6%	10.5%	0.00	0.9	0.9
		Y	9.8%	0.8%	10.5%	0.34	1.3	1.6
	K	N	19.6%	0.6%	20.2%	2.35	10.0	12.4
		Y	18.8%	1.0%	19.8%	6.25	12.0	18.3
	P	N	35.8%	1.1%	36.9%	0.14	4.3	4.5
		Y	40.9%	0.6%	41.5%	2.18	5.1	7.2
	S	N	37.0%	0.6%	37.7%	1.69	5.0	6.7
		Y	40.8%	0.9%	41.6%	3.34	5.0	8.4
Meili	C	N	0.2%	0.1%	0.4%	0.01	1.1	1.1
		Y	0.6%	0.0%	0.6%	0.11	1.8	1.9
	K	N	56.0%	1.6%	57.6%	0.26	10.7	11.0
		Y	55.0%	1.2%	56.2%	4.41	34.6	39.1
	P	N	43.3%	1.2%	44.5%	0.13	11.6	11.7
		Y	41.8%	1.2%	43.0%	3.60	11.6	15.2
	S	N	58.5%	1.1%	59.6%	0.10	8.6	8.7
		Y	58.9%	1.2%	60.1%	2.51	8.7	11.3
Type	C	N	0.5%	0.0%	0.5%	0.22	1.0	1.2
		Y	0.2%	0.0%	0.2%	0.53	1.4	1.9
	K	N	62.6%	1.5%	64.1%	0.84	8.8	9.6
		Y	57.1%	1.1%	58.2%	6.40	28.6	35.0
	P	N	45.6%	0.9%	46.5%	0.36	4.4	4.8
		Y	44.6%	1.0%	45.6%	3.42	4.4	7.9
	S	N	57.9%	1.0%	58.9%	0.22	4.3	4.6
		Y	56.2%	1.2%	57.5%	1.74	4.2	5.9

Table 8: 2WikiMultihopQA

Eng.	Ret.	RR	% Cor.	% Par.	% C+P	Retr.	Compl.	Resp.
Gold	P	N	37.0%	42.6%	79.6%	0.00	1.6	1.6
None	P	N	20.4%	46.3%	66.7%	0.00	1.2	1.2
Emb	C	N	24.1%	44.4%	68.5%	0.29	1.1	1.4
		Y	24.1%	35.2%	59.3%	0.35	1.7	2.1
	F	N	35.2%	38.9%	74.1%	0.44	8.3	8.8
		Y	25.9%	42.6%	68.5%	18.51	7.2	25.7
	K	N	7.4%	44.4%	51.9%	1.64	1.3	2.9
		Y	5.6%	35.2%	40.7%	2.25	1.4	3.7
	S	N	20.4%	46.3%	66.7%	0.27	1.4	1.6
		Y	18.5%	37.0%	55.6%	0.35	1.2	1.6
FTS	C	N	20.4%	40.7%	61.1%	0.01	1.2	1.2
		Y	22.2%	42.6%	64.8%	0.01	1.7	1.7
	K	N	20.4%	50.0%	70.4%	1.38	6.8	8.2
		Y	22.2%	44.4%	66.7%	6.50	10.5	16.9
	P	N	25.9%	42.6%	68.5%	0.01	1.2	1.3
		Y	22.2%	48.1%	70.4%	0.05	1.4	1.5
	S	N	22.2%	42.6%	64.8%	1.30	1.4	2.7
		Y	16.7%	42.6%	59.3%	1.04	1.6	2.6
Meili	C	N	11.1%	31.5%	42.6%	0.01	1.1	1.1
		Y	7.4%	31.5%	38.9%	0.11	1.8	1.9
	K	N	31.5%	46.3%	77.8%	0.28	8.2	8.5
		Y	33.3%	42.6%	75.9%	4.03	23.7	27.7
	P	N	13.0%	50.0%	63.0%	0.24	9.1	9.3
		Y	18.5%	46.3%	64.8%	4.35	9.2	13.6
	S	N	20.4%	51.9%	72.2%	0.18	7.4	7.6
		Y	22.2%	48.1%	70.4%	2.73	6.9	9.6
Type	C	N	7.4%	29.6%	37.0%	0.41	1.1	1.5
		Y	5.6%	13.0%	18.5%	0.82	1.3	2.2
	K	N	24.1%	48.1%	72.2%	0.80	7.8	8.6
		Y	35.2%	42.6%	77.8%	6.15	22.3	28.5
	P	N	18.5%	46.3%	64.8%	1.12	3.5	4.6
		Y	16.7%	38.9%	55.6%	5.23	3.6	8.8
	S	N	22.2%	40.7%	63.0%	0.37	2.0	2.4
		Y	24.1%	29.6%	53.7%	1.44	2.4	3.8

Table 9: Coursera

Eng.	Ret.	RR	% Cor.	% Par.	% C+P	Retr.	Compl.	Resp.
Gold	P	N	85.4%	1.4%	86.7%	0.01	2.0	2.0
None	P	N	9.8%	0.9%	10.7%	0.00	1.3	1.3
Emb	C	N	21.8%	0.1%	21.9%	0.24	1.4	1.6
		Y	20.8%	0.4%	21.1%	0.36	1.9	2.2
	F	N	31.9%	0.9%	32.8%	0.49	12.3	12.8
		Y	30.4%	0.8%	31.1%	24.75	10.8	35.6
	K	N	3.0%	0.7%	3.7%	1.28	1.2	2.4
		Y	1.6%	0.6%	2.2%	1.89	1.4	3.3
	S	N	12.0%	0.2%	12.3%	0.24	1.2	1.5
		Y	11.0%	0.9%	11.9%	0.33	1.3	1.7
FTS	C	N	7.4%	0.7%	8.2%	0.00	1.3	1.3
		Y	8.9%	0.4%	9.2%	0.01	1.8	1.8
	K	N	18.3%	1.4%	19.7%	2.90	11.8	14.7
		Y	16.9%	0.6%	17.5%	6.42	13.0	19.4
	P	N	50.6%	1.4%	51.9%	0.05	4.8	4.9
		Y	51.5%	1.1%	52.6%	1.19	5.3	6.5
	S	N	47.8%	1.0%	48.8%	1.72	4.9	6.6
		Y	48.8%	0.4%	49.1%	2.94	4.9	7.9
Meili	C	N	11.4%	0.2%	11.6%	0.01	1.3	1.3
		Y	11.1%	0.8%	11.9%	0.10	2.0	2.1
	K	N	47.6%	1.2%	48.8%	0.24	12.4	12.6
		Y	47.4%	2.0%	49.4%	3.90	43.2	47.1
	P	N	44.9%	1.4%	46.2%	0.13	11.6	11.7
		Y	45.2%	1.8%	47.0%	3.54	11.6	15.2
	S	N	50.8%	1.5%	52.3%	0.09	9.2	9.3
		Y	57.1%	1.4%	58.5%	2.38	9.4	11.8
Type	C	N	9.2%	0.7%	9.9%	0.22	1.0	1.3
		Y	8.9%	0.1%	9.0%	0.53	1.4	1.9
	K	N	50.6%	0.5%	51.1%	0.85	10.8	11.7
		Y	42.0%	1.5%	43.5%	5.66	41.4	47.0
	P	N	60.5%	1.9%	62.3%	0.33	4.8	5.1
		Y	62.4%	0.9%	63.2%	2.09	4.8	6.9
	S	N	62.8%	0.9%	63.7%	0.12	4.2	4.3
		Y	64.5%	1.5%	66.0%	1.16	4.3	5.5

Table 10: HotpotQA

Eng.	Ret.	RR	% Cor.	% Par.	% C+P	Retr.	Compl.	Resp.
Gold	P	N	53.8%	13.9%	67.7%	0.00	1.5	1.5
None	P	N	1.9%	1.9%	3.8%	0.00	0.9	0.9
Emb	C	N	12.7%	8.2%	20.9%	0.23	1.2	1.5
		Y	12.0%	6.3%	18.4%	0.34	1.5	1.9
	F	N	10.8%	5.1%	15.8%	0.50	4.8	5.3
		Y	17.1%	5.7%	22.8%	31.44	4.0	35.4
	K	N	7.0%	7.6%	14.6%	1.70	1.1	2.8
		Y	4.4%	3.8%	8.2%	2.53	1.2	3.7
	S	N	11.4%	7.0%	18.4%	0.24	1.1	1.4
		Y	11.4%	8.9%	20.3%	0.31	1.1	1.4
FTS	C	N	5.1%	3.8%	8.9%	0.01	0.9	0.9
		Y	5.1%	2.5%	7.6%	0.03	1.2	1.3
	K	N	10.8%	3.8%	14.6%	1.50	5.8	7.3
		Y	9.5%	5.1%	14.6%	10.74	7.1	17.9
	P	N	7.0%	7.6%	14.6%	0.38	3.2	3.6
		Y	10.1%	6.3%	16.5%	5.20	3.6	8.8
	S	N	5.7%	2.5%	8.2%	1.03	2.3	3.3
		Y	11.4%	6.3%	17.7%	3.71	2.7	6.4
Meili	C	N	4.4%	1.9%	6.3%	0.01	1.0	1.0
		Y	5.7%	2.5%	8.2%	0.10	1.5	1.6
	K	N	34.8%	14.6%	49.4%	0.30	8.2	8.5
		Y	38.0%	12.7%	50.6%	3.81	24.7	28.5
	P	N	11.4%	3.8%	15.2%	0.28	6.3	6.6
		Y	12.0%	3.8%	15.8%	6.67	6.5	13.1
	S	N	17.7%	10.8%	28.5%	0.16	7.4	7.6
		Y	22.2%	12.7%	34.8%	3.58	7.5	11.1
Type	C	N	3.8%	1.9%	5.7%	0.23	0.9	1.2
		Y	5.7%	1.3%	7.0%	0.51	1.2	1.7
	K	N	32.9%	9.5%	42.4%	2.50	6.1	8.6
		Y	33.5%	12.7%	46.2%	10.52	16.6	27.2
	P	N	10.8%	5.7%	16.5%	0.90	2.6	3.6
		Y	17.1%	5.1%	22.2%	8.90	3.2	12.1
	S	N	14.6%	9.5%	24.1%	0.53	2.8	3.3
		Y	14.6%	10.8%	25.3%	4.34	3.0	7.3

Table 11: MultiDoc2Dial

Eng.	Ret.	RR	% Cor.	% Par.	% C+P	Retr.	Compl.	Resp.
Gold	P	N	85.1%	9.1%	94.2%	0.01	1.6	1.6
None	P	N	1.7%	0.0%	1.7%	0.00	0.7	0.7
Emb	C	N	33.9%	4.1%	38.0%	0.24	1.0	1.3
		Y	33.9%	5.8%	39.7%	0.40	1.4	1.8
	F	N	34.7%	1.7%	36.4%	0.41	8.5	8.9
		Y	35.5%	1.7%	37.2%	22.40	6.8	29.2
	K	N	4.1%	5.8%	9.9%	1.61	0.9	2.5
		Y	1.7%	5.0%	6.6%	2.66	1.0	3.7
	S	N	14.9%	1.7%	16.5%	0.24	0.9	1.2
		Y	14.9%	3.3%	18.2%	0.33	0.9	1.3
FTS	C	N	0.8%	0.0%	0.8%	0.01	0.8	0.8
		Y	0.8%	0.0%	0.8%	0.01	1.0	1.0
	K	N	18.2%	2.5%	20.7%	1.61	6.9	8.5
		Y	19.0%	3.3%	22.3%	6.64	9.6	16.2
	P	N	26.4%	2.5%	28.9%	0.09	4.0	4.1
		Y	30.6%	1.7%	32.2%	1.90	4.5	6.4
	S	N	24.0%	0.8%	24.8%	0.93	3.5	4.4
		Y	31.4%	2.5%	33.9%	2.14	3.8	6.0
Meili	C	N	1.7%	0.8%	2.5%	0.01	1.0	1.0
		Y	1.7%	0.8%	2.5%	0.11	1.5	1.6
	K	N	73.6%	5.0%	78.5%	0.25	9.4	9.6
		Y	66.1%	7.4%	73.6%	3.43	28.6	32.0
	P	N	35.5%	3.3%	38.8%	0.21	9.6	9.8
		Y	38.8%	2.5%	41.3%	4.47	9.9	14.3
	S	N	52.1%	5.0%	57.0%	0.12	7.2	7.3
		Y	51.2%	5.0%	56.2%	2.53	7.5	10.1
Type	C	N	0.8%	0.0%	0.8%	0.12	0.8	1.0
		Y	0.8%	0.8%	1.7%	0.30	1.2	1.5
	K	N	54.5%	2.5%	57.0%	1.20	7.5	8.7
		Y	55.4%	5.8%	61.2%	6.45	25.2	31.7
	P	N	28.1%	4.1%	32.2%	0.63	4.2	4.8
		Y	32.2%	2.5%	34.7%	4.49	4.4	8.8
	S	N	47.1%	5.0%	52.1%	0.29	3.3	3.6
		Y	47.9%	4.1%	52.1%	2.17	3.7	5.9

Table 12: MultiFieldQA

Eng.	Ret.	RR	% Cor.	% Par.	% C+P	Retr.	Compl.	Resp.
Gold	P	N	58.1%	2.1%	60.2%	0.01	2.7	2.7
None	P	N	8.4%	1.2%	9.6%	0.00	1.4	1.4
Emb	C	N	21.1%	1.2%	22.3%	0.22	1.7	1.9
		Y	21.2%	1.1%	22.4%	0.35	2.3	2.6
	F	N	34.4%	2.0%	36.4%	0.51	15.2	15.7
		Y	34.6%	2.1%	36.8%	25.01	13.5	38.5
	K	N	6.9%	1.6%	8.5%	1.46	1.4	2.8
		Y	5.4%	1.6%	7.0%	2.52	1.6	4.2
	S	N	18.5%	1.0%	19.5%	0.23	1.5	1.7
		Y	16.6%	1.4%	18.0%	0.32	1.5	1.8
FTS	C	N	7.0%	1.6%	8.6%	0.00	1.4	1.4
		Y	7.9%	1.4%	9.2%	0.01	1.9	1.9
	K	N	19.7%	2.3%	22.1%	3.10	12.2	15.3
		Y	22.4%	2.9%	25.2%	6.72	13.2	19.9
	P	N	24.8%	2.3%	27.1%	0.05	4.0	4.0
		Y	28.4%	2.0%	30.4%	1.23	4.7	5.9
	S	N	29.3%	1.8%	31.2%	2.31	6.7	9.0
		Y	32.5%	1.9%	34.4%	3.87	6.4	10.3
Meili	C	N	4.1%	0.1%	4.2%	0.01	1.3	1.3
		Y	3.6%	0.5%	4.1%	0.10	2.3	2.4
	K	N	34.5%	3.2%	37.7%	0.25	14.0	14.2
		Y	37.4%	3.5%	40.9%	4.78	34.0	38.7
	P	N	31.4%	2.8%	34.3%	0.17	12.2	12.3
		Y	32.9%	2.9%	35.8%	4.16	12.5	16.7
	S	N	36.3%	3.2%	39.5%	0.12	11.6	11.7
		Y	40.5%	1.6%	42.1%	2.71	11.6	14.3
Type	C	N	1.7%	0.2%	2.0%	0.31	1.1	1.5
		Y	1.8%	0.1%	1.9%	0.70	1.5	2.2
	K	N	36.9%	2.5%	39.3%	1.00	12.8	13.8
		Y	37.1%	2.9%	40.0%	5.69	47.2	52.9
	P	N	33.7%	2.5%	36.1%	0.68	5.2	5.9
		Y	34.8%	2.2%	37.0%	3.51	5.4	8.9
	S	N	38.3%	2.1%	40.4%	0.22	6.8	7.0
		Y	40.0%	1.8%	41.8%	2.10	7.1	9.2

Table 13: MuSiQue

Eng.	Ret.	RR	% Cor.	% Par.	% C+P	Retr.	Compl.	Resp.
Gold	P	N	43.2%	5.3%	48.6%	0.03	2.4	2.4
None	P	N	1.4%	0.0%	1.4%	0.00	0.8	0.8
Emb	C	N	7.5%	2.3%	9.9%	0.23	1.1	1.3
		Y	7.1%	1.8%	8.9%	0.30	1.5	1.8
	F	N	9.8%	1.1%	10.9%	0.50	7.5	8.0
		Y	13.8%	1.1%	14.9%	38.11	6.5	44.6
	K	N	1.6%	1.2%	2.8%	1.35	0.9	2.3
		Y	1.0%	1.5%	2.5%	2.69	1.1	3.8
	S	N	3.0%	1.1%	4.1%	0.23	1.0	1.2
		Y	2.8%	1.2%	4.0%	0.46	1.0	1.4
FTS	C	N	1.6%	0.5%	2.1%	0.00	0.8	0.8
		Y	1.5%	0.0%	1.5%	0.01	1.0	1.0
	K	N	1.7%	0.2%	2.0%	1.65	8.1	9.8
		Y	4.1%	0.5%	4.6%	6.45	9.1	15.6
	P	N	6.7%	0.9%	7.6%	0.16	4.3	4.5
		Y	12.9%	2.2%	15.1%	3.39	6.0	9.4
	S	N	4.8%	0.9%	5.7%	1.22	3.5	4.7
		Y	8.6%	1.8%	10.4%	4.29	4.4	8.7
Meili	C	N	2.8%	0.2%	3.1%	0.01	1.0	1.0
		Y	2.6%	0.2%	2.9%	0.10	1.7	1.8
	K	N	17.2%	3.5%	20.7%	0.35	9.2	9.5
		Y	17.8%	2.6%	20.4%	4.85	17.2	22.1
	P	N	18.4%	2.1%	20.5%	0.23	8.5	8.7
		Y	17.8%	3.8%	21.5%	5.94	9.5	15.4
	S	N	14.8%	2.1%	16.9%	0.16	7.6	7.8
		Y	16.8%	1.9%	18.6%	3.77	8.6	12.4
Type	C	N	1.8%	0.0%	1.8%	0.08	0.9	1.0
		Y	2.0%	0.0%	2.0%	0.24	1.3	1.6
	K	N	12.7%	2.0%	14.7%	1.49	7.2	8.7
		Y	14.8%	2.6%	17.4%	9.39	25.3	34.7
	P	N	15.3%	2.4%	17.7%	0.58	4.9	5.5
		Y	16.4%	2.5%	18.9%	6.93	5.3	12.3
	S	N	14.4%	1.7%	16.2%	0.43	4.0	4.5
		Y	13.4%	1.5%	14.9%	4.51	4.8	9.3

Table 14: NarrativeQA

Eng.	Ret.	RR	% Cor.	% Par.	% C+P	Retr.	Compl.	Resp.
Gold	P	N	66.4%	4.0%	70.4%	0.01	1.8	1.8
None	P	N	13.7%	3.7%	17.4%	0.00	1.0	1.0
Emb	C	N	31.9%	6.3%	38.2%	0.21	1.2	1.4
		Y	31.9%	5.4%	37.3%	0.30	1.6	1.9
	F	N	47.6%	5.4%	53.0%	0.30	13.5	13.8
		Y	48.1%	5.7%	53.8%	10.09	12.0	22.1
	K	N	5.7%	3.1%	8.8%	1.34	0.9	2.2
		Y	7.4%	2.8%	10.3%	2.60	1.1	3.7
	S	N	27.9%	3.1%	31.1%	0.24	1.1	1.3
		Y	24.5%	4.0%	28.5%	0.32	1.1	1.4
FTS	C	N	11.7%	2.3%	14.0%	0.00	1.0	1.0
		Y	12.3%	2.8%	15.1%	0.01	1.3	1.3
	K	N	17.4%	3.4%	20.8%	1.65	9.1	10.8
		Y	18.8%	3.7%	22.5%	5.43	11.7	17.1
	P	N	28.5%	3.7%	32.2%	0.12	8.1	8.2
		Y	38.5%	5.1%	43.6%	3.35	9.1	12.5
	S	N	28.2%	3.7%	31.9%	1.04	6.0	7.0
		Y	31.3%	2.8%	34.2%	3.64	7.2	10.9
Meili	C	N	4.8%	1.1%	6.0%	0.01	0.9	0.9
		Y	5.4%	1.7%	7.1%	0.11	1.5	1.6
	K	N	49.0%	6.3%	55.3%	0.28	8.3	8.6
		Y	51.9%	5.7%	57.5%	3.92	13.7	17.6
	P	N	35.3%	4.3%	39.6%	0.17	10.5	10.7
		Y	37.6%	5.7%	43.3%	4.37	11.2	15.6
	S	N	41.6%	4.8%	46.4%	0.15	9.9	10.1
		Y	44.7%	4.3%	49.0%	3.50	10.5	14.0
Type	C	N	2.8%	0.6%	3.4%	0.11	0.8	0.9
		Y	3.7%	0.3%	4.0%	0.30	1.1	1.4
	K	N	41.0%	5.7%	46.7%	1.04	8.5	9.5
		Y	47.9%	5.4%	53.3%	6.57	33.4	40.0
	P	N	35.9%	3.7%	39.6%	0.41	7.8	8.2
		Y	35.6%	4.6%	40.2%	5.93	8.4	14.3
	S	N	35.6%	4.8%	40.5%	0.29	6.2	6.5
		Y	34.5%	4.3%	38.7%	3.24	6.8	10.0

Table 15: Naturalquestion

Eng.	Ret.	RR	% Cor.	% Par.	% C+P	Retr.	Compl.	Resp.
Gold	P	N	68.9%	15.9%	84.8%	0.00	1.6	1.6
None	P	N	0.2%	0.1%	0.4%	0.00	0.7	0.7
Emb	C	N	17.3%	9.1%	26.5%	0.21	1.2	1.4
		Y	17.6%	9.1%	26.8%	0.33	1.6	1.9
	F	N	21.2%	7.1%	28.4%	0.40	10.1	10.5
		Y	22.0%	9.0%	31.0%	13.48	8.7	22.2
	K	N	5.7%	3.8%	9.5%	1.60	1.0	2.6
		Y	5.5%	3.2%	8.8%	2.91	1.2	4.1
	S	N	13.2%	6.7%	19.9%	0.22	1.0	1.2
		Y	12.8%	6.1%	18.9%	0.32	1.1	1.4
FTS	C	N	2.1%	0.5%	2.6%	0.01	0.8	0.8
		Y	1.9%	0.8%	2.6%	0.02	1.1	1.1
	K	N	10.7%	5.4%	16.1%	1.29	7.7	9.0
		Y	13.9%	6.1%	20.0%	6.27	13.4	19.6
	P	N	14.6%	5.6%	20.2%	0.10	6.3	6.4
		Y	15.6%	6.0%	21.6%	2.40	6.9	9.3
	S	N	13.3%	6.3%	19.6%	0.98	5.1	6.1
		Y	12.8%	6.1%	18.9%	2.90	5.4	8.3
Meili	C	N	6.6%	4.1%	10.7%	0.01	1.4	1.4
		Y	6.8%	4.4%	11.1%	0.09	2.1	2.2
	K	N	25.9%	10.0%	35.9%	0.16	7.5	7.6
		Y	26.6%	10.2%	36.9%	2.93	8.4	11.4
	P	N	19.0%	7.4%	26.5%	0.12	10.3	10.4
		Y	20.5%	8.2%	28.7%	3.48	10.4	13.9
	S	N	18.8%	8.8%	27.6%	0.07	7.3	7.4
		Y	18.9%	7.2%	26.1%	2.13	7.1	9.3
Type	C	N	6.1%	2.4%	8.5%	0.07	1.1	1.2
		Y	6.5%	2.8%	9.2%	0.22	1.5	1.7
	K	N	23.0%	8.7%	31.7%	0.46	7.8	8.3
		Y	24.8%	7.5%	32.2%	3.78	20.9	24.7
	P	N	16.8%	7.2%	24.0%	0.28	6.8	7.1
		Y	18.5%	7.0%	25.5%	3.90	7.0	10.9
	S	N	19.8%	7.4%	27.2%	0.16	4.6	4.8
		Y	17.8%	7.0%	24.8%	1.80	4.7	6.5

Table 16: QASPER

Eng.	Ret.	RR	% Cor.	% Par.	% C+P	Retr.	Compl.	Resp.
Gold	P	N	71.8%	0.7%	72.5%	0.01	1.6	1.6
None	P	N	4.3%	0.0%	4.3%	0.00	0.8	0.8
Emb	C	N	31.1%	0.1%	31.2%	0.23	1.2	1.4
		Y	29.6%	0.5%	30.1%	0.30	1.7	2.0
	F	N	23.6%	0.1%	23.8%	0.57	6.2	6.8
		Y	28.9%	0.2%	29.1%	32.06	5.8	37.8
	K	N	14.7%	0.1%	14.8%	1.61	1.0	2.6
		Y	15.1%	0.1%	15.2%	2.70	1.2	3.9
	S	N	17.6%	0.1%	17.8%	0.26	1.2	1.5
		Y	16.5%	0.1%	16.6%	0.33	1.0	1.4
FTS	C	N	4.9%	0.0%	4.9%	0.01	0.8	0.8
		Y	5.9%	0.1%	6.0%	0.01	1.1	1.1
	K	N	11.0%	0.5%	11.5%	1.59	6.0	7.6
		Y	14.2%	0.6%	14.9%	8.32	22.1	30.4
	P	N	5.6%	0.0%	5.6%	0.02	0.9	0.9
		Y	6.8%	0.2%	7.0%	0.30	1.1	1.4
	S	N	7.8%	0.1%	8.0%	1.15	1.2	2.3
		Y	8.0%	0.0%	8.0%	2.03	1.4	3.4
Meili	C	N	3.9%	0.4%	4.3%	0.01	0.9	0.9
		Y	5.1%	0.0%	5.1%	0.10	1.4	1.5
	K	N	40.2%	0.4%	40.6%	0.38	9.1	9.5
		Y	44.4%	0.1%	44.5%	4.10	11.5	15.6
	P	N	28.7%	0.4%	29.0%	0.25	7.3	7.5
		Y	28.5%	0.1%	28.6%	6.08	7.5	13.5
	S	N	30.3%	0.4%	30.6%	0.12	5.8	6.0
		Y	33.6%	0.5%	34.1%	3.33	6.3	9.6
Type	C	N	3.6%	0.0%	3.6%	0.28	0.8	1.1
		Y	4.2%	0.1%	4.4%	0.64	1.2	1.8
	K	N	32.2%	0.5%	32.7%	1.84	6.5	8.3
		Y	33.1%	0.4%	33.5%	9.99	20.5	30.5
	P	N	16.8%	0.2%	17.0%	1.43	2.6	4.0
		Y	17.9%	0.2%	18.1%	7.29	2.8	10.0
	S	N	24.4%	0.5%	24.9%	0.41	2.4	2.8
		Y	23.1%	0.4%	23.5%	3.25	2.6	5.9

Table 17: QuALTY

Eng.	Ret.	RR	% Cor.	% Par.	% C+P	Retr.	Compl.	Resp.
Gold	P	N	88.9%	7.2%	96.1%	0.00	1.1	1.1
None	P	N	19.6%	3.1%	22.7%	0.00	0.8	0.8
Emb	C	N	63.0%	5.9%	68.9%	0.22	1.2	1.4
		Y	62.5%	6.1%	68.6%	0.30	1.6	1.9
	F	N	53.6%	5.1%	58.8%	0.51	5.5	6.1
		Y	65.0%	5.8%	70.8%	22.86	5.2	28.0
	K	N	22.0%	4.7%	26.7%	1.50	1.0	2.5
		Y	18.9%	4.0%	22.9%	2.50	1.1	3.6
	S	N	52.2%	4.9%	57.1%	0.23	1.2	1.5
		Y	52.6%	5.4%	58.0%	0.32	1.1	1.4
FTS	C	N	20.2%	3.2%	23.5%	0.01	0.8	0.8
		Y	19.9%	2.9%	22.8%	0.01	1.1	1.1
	K	N	21.6%	3.4%	25.0%	1.64	5.0	6.7
		Y	21.9%	3.5%	25.4%	7.95	18.8	26.8
	P	N	19.0%	3.6%	22.6%	0.00	0.9	0.9
		Y	17.5%	3.8%	21.2%	0.01	1.0	1.0
	S	N	20.3%	3.4%	23.7%	1.13	1.1	2.2
		Y	19.4%	2.9%	22.2%	2.40	1.4	3.8
Meili	C	N	1.9%	0.2%	2.1%	0.01	0.7	0.7
		Y	2.9%	0.4%	3.2%	0.10	1.3	1.4
	K	N	45.2%	5.6%	50.8%	0.33	7.0	7.3
		Y	46.6%	6.4%	53.0%	4.42	13.4	17.8
	P	N	20.6%	3.0%	23.6%	0.28	5.2	5.5
		Y	20.4%	3.0%	23.4%	7.96	4.8	12.8
	S	N	28.5%	3.9%	32.3%	0.17	6.0	6.2
		Y	31.4%	4.6%	36.0%	4.56	6.3	10.9
Type	C	N	1.4%	0.2%	1.6%	0.30	0.8	1.1
		Y	1.8%	0.4%	2.1%	0.68	1.1	1.7
	K	N	41.2%	5.4%	46.6%	1.79	6.3	8.0
		Y	45.6%	5.0%	50.6%	10.54	20.1	30.6
	P	N	20.7%	3.9%	24.6%	2.06	3.1	5.2
		Y	20.0%	4.1%	24.1%	7.13	3.3	10.4
	S	N	25.0%	3.9%	28.8%	0.53	2.0	2.5
		Y	24.5%	3.5%	28.0%	3.65	2.7	6.3

Table 18: TOEFL-QA

A.5 Search Engine Hyperparameters

This section lists the effective document-processing and ranking settings that affect retrieval results in the experiment implementation. In most cases, default settings were used. Only a small number of settings were explicitly set by the implementation (listed below).

- **Explicit (non-default or implementation-defined) settings**:
  - Top-k: `k = 10` for all search engines and retriever variants.
  - No dataset filtering was applied at query time (all queries run against the full combined corpus).
  - SQLite FTS query normalization: lowercase + extract Unicode word tokens via regex `\w+`.
  - Chunking for chunk indices/collections: max 512 characters, 50 characters overlap, prefer sentence boundaries.
  - pgvector: cosine distance search over `halfvec(2560)` embeddings with an HNSW index (without explicit HNSW parameter overrides).
- **SQLite Full-Text Search (FTS5 + BM25)**:
  - Index schema (full documents): `CREATE VIRTUAL TABLE content_fts USING fts5(content, content_id);`
  - Index schema (chunks): `CREATE VIRTUAL TABLE content_fts_chunks USING fts5(chunk, content_id UNINDEXED, chunk_index UNINDEXED, dataset UNINDEXED);`
  - Query preprocessing: the query is converted into a whitespace-separated list of lowercase Unicode word tokens using the regex `\w+` (i.e., punctuation is removed).
  - Tokenizer/stopwords/stemming/synonyms: no explicit tokenizer, stopword list, stemming, or synonym configuration was applied. This means the defaults from the engine were used.
  - Ranking: results are ordered by the FTS5 built-in `rank` column (default BM25 ranking), ascending.
- **pgVector (PostgreSQL + pgvector)**:
  - Docker image: `pgvector/pgvector:pg18-trixie`
  - Reported Version: `PostgreSQL 18.0 (Debian 18.0-1.pgdg13+3) on x86_64-pc-linux-gnu, compiled by gcc (Debian 14.2.0-19) 14.2.0, 64-bit`
  - pgVector version: `0.8.1`
  - Embedding model: Qwen3-4B embeddings with 2,560 dimensions; stored as `halfvec(2560)`.
  - Similarity metric: cosine distance (`<=>` operator).
  - Approximate Nearest Neighbor (ANN) index: `CREATE INDEX ... USING hnsw (embedding halfvec_cosine_ops)`.
  - HNSW parameters (`m`, `ef_construction`, `ef_search`): not explicitly set in the implementation (engine defaults for the used pgvector version).
- **Typesense**:
  - Docker image: `typesense/typesense:29.0`
  - Version: `29.0`

- Schema (full documents): `content` (string), `dataset` (string, facet), `content_id` (int32)
- Schema (chunks): `chunk` (string), `dataset` (string, facet), `content_id` (int32), `chunk_index` (int32)
- Query parameters: `q = <query>`, `query_by = content` (or `chunk`), `per_page = 10`
- Tokenization/stopwords/stemming/synonyms and ranking settings: no custom configuration was applied, using the defaults.

- **Meilisearch**:
  - Docker image: `getmeili/meilisearch:v1.22.3`
  - Version: `1.22.3`, commit `c36a3239ca387ae662e13ebea697919ca04e5c75`
  - Index setup: primary key `id`; documents contain `content` (or `chunk`), `dataset`, `content_id`
  - Query parameters: `q = <query>`, `limit = 10`
  - Tokenization/stopwords/stemming/synonyms: no custom configuration was applied (engine defaults).
  - Ranking: Meilisearch default ranking rules were used (no custom ranking rule order).

B List of Figures

C List of Tables

D List of Acronyms

ANN:	Approximate Nearest Neighbor
ASR:	Automatic Speech Recognition
ASR-MT-TTS:	Automatic Speech Recognition - Machine Translation - Text-to-Speech
RAG:	Retrieval Augmented Generation
LLM:	Large Language Model
NLP:	Natural Language Processing
BERT:	Bidirectional encoder representations from transformers
BM25:	Best Matching 25
HNSW:	Hierarchical Navigable Small World
PRF:	Probabilistic Relevance Framework
QA:	Question Answering
FTS:	Full-Text Search

E Bibliography

[1] S. E. Robertson and K. S. Jones, "Relevance weighting of search terms," *Journal of the American Society for Information Science*, vol. 27, no. 3, pp. 129–146, 1976, doi: https://doi.org/10.1002/asi.4630270302.

[2] Xinze Li, Yixin Cao, Yubo Ma, and Aixin Sun, "Long Context vs. RAG for LLMs: An Evaluation and Revisits," Dec. 27, 2024. [Online]. Available: https://arxiv.org/abs/2501.01880

[3] Patrick Lewis *et al.*, "Retrieval-Augmented Generation for Knowledge-Intensive NLP Tasks," Apr. 12, 2021. [Online]. Available: https://arxiv.org/abs/2005.11401

[4] Yunfan Gao *et al.*, "Retrieval-Augmented Generation for Large Language Models: A Survey," Mar. 27, 2024. [Online]. Available: https://arxiv.org/abs/2312.10997

[5] Panda Smith, "Build a search engine, not a vector DB." Accessed: Sept. 05, 2025. [Online]. Available: https://blog.elicit.com/search-vs-vector-db/

[6] Arvind Neelakantan *et al.*, "Text and Code Embeddings by Contrastive Pre-Training," Jan. 24, 2022. [Online]. Available: https://arxiv.org/abs/2201.10005

[7] Zach Nussbaum, John X. Morris, Brandon Duderstadt, and Andriy Mulyar, "Nomic Embed: Training a Reproducible Long Context Text Embedder," Feb. 03, 2025. [Online]. Available: https://arxiv.org/abs/2402.01613

[8] Zach Nussbaum, John X. Morris, Brandon Duderstadt, and Andriy Mulyar, "Nomic Embed: Training a Reproducible Long Context Text Embedder," Feb. 2025. [Online]. Available: https://arxiv.org/abs/2402.01613

[9] Anne Lauscher, Olga Majewska, Leonardo F. R. Ribeiro, Iryna Gurevych, Nikolai Rozanov, and Goran Glavaš, "Common Sense or World Knowledge? Investigating Adapter-Based Knowledge Injection into Pretrained Transformers," Oct. 11, 2020. [Online]. Available: https://arxiv.org/abs/2005.11787

[10] Yun Luo, Zhen Yang, Fandong Meng, Yafu Li, Jie Zhou, and Yue Zhang, "An Empirical Study of Catastrophic Forgetting in Large Language Models During Continual Fine-tuning." [Online]. Available: https://arxiv.org/abs/2308.08747

[11] Ruize Wang *et al.*, "K-ADAPTER: Infusing Knowledge into Pre-Trained Models with Adapters," Dec. 28, 2020. [Online]. Available: arxiv:2002.01808v5

[12] Oded Ovadia, Menachem Brief, Moshik Mishaeli, and Oren Elisha, "Fine-Tuning or Retrieval? Comparing Knowledge Injection in LLMs," Jan. 30, 2024. [Online]. Available: https://arxiv.org/abs/2312.05934

[13] Tianjun Zhang *et al.*, "RAFT: Adapting Language Model to Domain Specific RAG," June 05, 2024. [Online]. Available: https://arxiv.org/abs/2403.10131v2

[14] Sebastian Borgeaud *et al.*, "Improving language models by retrieving from trillions of tokens," Feb. 07, 2022. [Online]. Available: https://arxiv.org/abs/2112.04426

[15] "Introducing Contextual Retrieval," Sept. 19, 2024. [Online]. Available: https://www.anthropic.com/news/contextual-retrieval

[16] Timo Schick *et al.*, "Toolformer: Language Models Can Teach Themselves to Use Tools," Feb. 09, 2023. [Online]. Available: https://arxiv.org/abs/2302.04761

[17] Shunyu Yao *et al.*, "REACT: SYNERGIZING REASONING AND ACTING IN LANGUAGE MODELS," Mar. 10, 2023. [Online]. Available: https://arxiv.org/abs/2210.03629

[18] Zirui Guo, Lianghao Xia, Yanhua Yu, Tu Ao, and Chao Huang, "LIGHTRAG: SIMPLE AND FAST RETRIEVAL-AUGMENTED GENERATION," Oct. 08, 2024. [Online]. Available: https://arxiv.org/abs/2410.05779

[19] Masoomali Fatehkia, Ji Kim Lucas, and Sanjay Chawla, "T-RAG: LESSONS FROM THE LLM TRENCHES," June 06, 2024. [Online]. Available: https://arxiv.org/abs/2402.07483

[20] Zahra Sepasdar, Sushant Gautam, Cise Midoglu, Michael A. Riegler, and Pål Halvorsen, "Enhancing Structured-Data Retrieval with GraphRAG: Soccer Data Case Study," Sept. 26, 2024. [Online]. Available: https://arxiv.org/abs/2409.17580

[21] Bernal Jiménez Gutiérrez, Yiheng Shu, Yu Gu, Michihiro Yasunaga, and Yu Su, "HippoRAG: Neurobiologically Inspired Long-Term Memory for Large Language Models," Jan. 14, 2025. [Online]. Available: https://arxiv.org/abs/2405.14831

[22] Zhengbao Jiang *et al.*, "Active Retrieval Augmented Generation," Oct. 22, 2023. [Online]. Available: https://arxiv.org/abs/2305.06983

[23] Hervé Déjean, "Let your LLM generate a few tokens and you will reduce the need for retrieval," Dec. 16, 2024. [Online]. Available: https://arxiv.org/abs/2412.11536

[24] Brian J Chan, Chao-Ting Chen, Jui-Hung Cheng, and Hen-Hsen Huang, "Don't Do RAG: When Cache-Augmented Generation is All You Need for Knowledge Tasks," Feb. 23, 2025. [Online]. Available: https://arxiv.org/abs/2412.15605v2

[25] Alexandria Leto, Cecilia Aguerrebere, Ishwar Bhati, Mariano Tepper, Ted Willke, and Vy Ai Vo, "Toward Optimal Search and Retrieval for RAG," Nov. 11, 2024. [Online]. Available: https://arxiv.org/abs/2411.07396

[26] Sumit Soman and Sujoy Roychowdhury, "OBSERVATIONS ON BUILDING RAG SYSTEMS FOR TECHNICAL DOCUMENTS," Mar. 31, 2024. [Online]. Available: arxiv:2404.00657v1

[27] Orion Weller, Benjamin Van Durme, Dawn Lawrie, Ashwin Paranjape, Yuhao Zhang, and Jack Hessel, "Promptriever: Instruction-Trained Retrievers Can Be Prompted Like Language Models," Sept. 17, 2024. [Online]. Available: https://arxiv.org/abs/2409.11136

[28] Yue Yu *et al.*, "RankRAG: Unifying Context Ranking with Retrieval-Augmented Generation in LLMS," July 02, 2024. [Online]. Available: https://arxiv.org/abs/2407.02485

[29] Shi-Qi Yan, Jia-Chen Gu, Yun Zhu, and Zhen-Hua Ling, "Corrective Retrieval Augmented Generation," Oct. 07, 2024. [Online]. Available: https://arxiv.org/abs/2401.15884

[30] Xin Zhang *et al.*, "mGTE: Generalized Long-Context Text Representation and Reranking Models for Multilingual Text Retrieval," Nov. 12, 2024.

[31] Vladimir Blagojevic, "Enhancing RAG Pipelines in Haystack: Introducing DiversityRanker and LostInTheMiddleRanker," Aug. 2023.

[32] Akari Asai, Zeqiu Wu, Yizhong Wang, Avirup Sil, and Hannaneh Hajishirzi, "SELF-RAG: LEARNING TO RETRIEVE, GENERATE, AND CRITIQUE THROUGH SELF-REFLECTION," Oct. 17, 2023. [Online]. Available: https://arxiv.org/abs/2310.11511

[33] Yuan Xia, Jingbo Zhou, Zhenhui Shi, Jun Chen, and Haifeng Huang, "Improving Retrieval Augmented Language Model with Self-Reasoning," Dec. 19, 2024. [Online]. Available: https://arxiv.org/abs/2407.19813

[34] Siran Li, Linus Stenzel, Carsten Eickhoff, and Seyed Ali Bahrainian, "Enhancing Retrieval-Augmented Generation: A Study of Best Practices," Jan. 13, 2025. [Online]. Available: https://arxiv.org/abs/2501.07391

[35] Zekun Xi *et al.*, "OmniThink: Expanding Knowledge Boundaries in Machine Writing through Thinking," Feb. 20, 2025. [Online]. Available: https://arxiv.org/abs/2501.09751 v2

[36] Zijun Yao *et al.*, "SEAKR: Self-aware Knowledge Retrieval for Adaptive Retrieval Augmented Generation," June 27, 2024. [Online]. Available: https://arxiv.org/abs/2406.19215 v1

[37] Stephen Robertson and Hugo Zaragoza, "The Probabilistic Relevance Framework: BM25 and Beyond," 2009.

[38] Aditi Singh, Abul Ehtesham, Saket Kumar, and Tala Talaei Khoei, "AGENTIC RETRIEVAL-AUGMENTED GENERATION: A SURVEY ON AGENTIC RAG," Feb. 04, 2025. [Online]. Available: https://arxiv.org/abs/2501.09136

[39] Nathan J. Anderson, Caleb Wilson, and Stephen D. Richardson, "Lingua: Addressing Scenarios for Live Interpretation and Automatic Dubbing," Sept. 12, 2022.

[40] Robert Friel, Masha Belyi, and Atindriyo Sanyal, "RAGBench: Explainable Benchmark for Retrieval-Augmented Generation Systems," June 25, 2024. [Online]. Available: https://arxiv.org/abs/2407.11005

[41] Daniel Fleischer, Moshe Berchansky, Moshe Wasserblat, and Peter Izsak, "RAG Foundry: A Framework for Enhancing LLMs for Retrieval Augmented Generation," Aug. 05, 2024. [Online]. Available: https://arxiv.org/abs/2408.02545

[42] Satyapriya Krishna *et al.*, "Fact, Fetch, and Reason: A Unified Evaluation of Retrieval-Augmented Generation," Jan. 24, 2025. [Online]. Available: https://arxiv.org/abs/2409.12941

[43] T. Kwiatkowski *et al.*, "Natural Questions: A Benchmark for Question Answering Research," *Transactions of the Association for Computational Linguistics*, vol. 7, pp. 453–466, 2019, doi: 10.1162/tacl_a_00276.

[44] X. Ho, A.-K. Duong Nguyen, S. Sugawara, and A. Aizawa, "Constructing A Multi-hop QA Dataset for Comprehensive Evaluation of Reasoning Steps," in *Proceedings of the 28th International Conference on Computational Linguistics*, D. Scott, N. Bel, and C. Zong, Eds., Barcelona, Spain (Online): International Committee on Computational Linguistics, 2020, pp. 6609–6625. doi: 10.18653/v1/2020.coling-main.580.

[45] Z. Yang *et al.*, "HotpotQA: A Dataset for Diverse, Explainable Multi-hop Question Answering," in *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, E. Riloff, D. Chiang, J. Hockenmaier, and J. Tsujii, Eds., Brussels, Belgium: Association for Computational Linguistics, 2018, pp. 2369–2380. doi: 10.18653/v1/D18-1259.

[46] H. Trivedi, N. Balasubramanian, T. Khot, and A. Sabharwal, "♫ MuSiQue: Multihop Questions via Single-hop Question Composition," *Transactions of the Association for Computational Linguistics*, vol. 10, pp. 539–554, 2022, doi: 10.1162/tacl_a_00475.

[47] Y. Bai *et al.*, "LongBench: A Bilingual, Multitask Benchmark for Long Context Understanding," in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, L.-W. Ku, A. Martins, and V. Srikumar, Eds., Bangkok, Thailand: Association for Computational Linguistics, 2024, pp. 3119–3137. doi: 10.18653/v1/2024.acl-long.172.

[48] T. Kočiský *et al.*, "The NarrativeQA Reading Comprehension Challenge," *Transactions of the Association for Computational Linguistics*, vol. 6, pp. 317–328, 2018, doi: 10.1162/tacl_a_00023.

[49] P. Dasigi, K. Lo, I. Beltagy, A. Cohan, N. A. Smith, and M. Gardner, "A Dataset of Information-Seeking Questions and Answers Anchored in Research Papers," in *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, K. Toutanova, A. Rumshisky, L. Zettlemoyer, D. Hakkani-Tur, I. Beltagy, S. Bethard, R. Cotterell, T. Chakraborty, and Y. Zhou, Eds., Online: Association for Computational Linguistics, 2021, pp. 4599–4610. doi: 10.18653/v1/2021.naacl-main.365.

[50] R. Y. Pang *et al.*, "QuALITY: Question Answering with Long Input Texts, Yes!," in *Proceedings of the 2022 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, M. Carpuat, M.-C. de Marneffe, and I. V. Meza Ruiz, Eds., Seattle, United States: Association for Computational Linguistics, 2022, pp. 5336–5358. doi: 10.18653/v1/2022.naacl-main.391.

[51] C. Wang *et al.*, "NovelQA: Benchmarking Question Answering on Documents Exceeding 200K Tokens." [Online]. Available: https://arxiv.org/abs/2403.12766

[52] S. Feng, S. S. Patel, H. Wan, and S. Joshi, "MultiDoc2Dial: Modeling Dialogues Grounded in Multiple Documents," in *Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing*, Association for Computational Linguistics, 2021, pp. 6162–6176. doi: 10.18653/v1/2021.emnlp-main.498.

[53] B.-H. Tseng, S.-S. Shen, H.-Y. Lee, and L.-S. Lee, "Towards machine comprehension of spoken content: Initial TOEFL listening comprehension test by machine," in *INTERSPEECH*, 2016.

[54] C. An *et al.*, "L-Eval: Instituting Standardized Evaluation for Long Context Language Models," in *Proceedings of the 62nd Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, L.-W. Ku, A. Martins, and V. Srikumar, Eds., Bangkok, Thailand: Association for Computational Linguistics, 2024, pp. 14388–14411. doi: 10.18653/v1/2024.acl-long.776.

[55] "gpt-oss-120b & gpt-oss-20b Model Card," Aug. 05, 2025. [Online]. Available: https://arxiv.org/pdf/2508.10925

[56] "GPT-4o System Card," Aug. 08, 2024. [Online]. Available: https://cdn.openai.com/gpt-4o-system-card.pdf

[57] K. Enevoldsen *et al.*, "MMTEB: Massive Multilingual Text Embedding Benchmark," *arXiv preprint arXiv:2502.13595*, 2025, doi: 10.48550/arXiv.2502.13595.

[58] Feng Wang, Yuqing Li, and Han Xiao, "jina-reranker-v3: Last but Not Late Interaction for Listwise Document Reranking," Oct. 06, 2025. [Online]. Available: https://arxiv.org/abs/2509.25085

[59] "SQLite FTS5 Extension." Accessed: Nov. 12, 2025. [Online]. Available: https://sqlite.org/fts5.html

[60] "BM25." Accessed: Nov. 12, 2025. [Online]. Available: https://docs.paradedb.com/documentation/concepts/bm25

[61] "Ranking and reranking." Accessed: Nov. 12, 2025. [Online]. Available: https://www.elastic.co/docs/solutions/search/ranking

[62] "BM25." Accessed: Nov. 12, 2025. [Online]. Available: https://docs.singlestore.com/db/v9.0/reference/sql-reference/full-text-search-functions/bm-25/

[63] "Return the Score Details - Atlas - MongoDB Docs." Accessed: Nov. 12, 2025. [Online]. Available: https://www.mongodb.com/docs/atlas/atlas-search/score/get-details/#bm25

[64] "pgvector/pgvector: Open-source vector similarity search for Postgres." [Online]. Available: https://github.com/pgvector/pgvector

[65] "Ranking and Relevance." Accessed: Nov. 12, 2025. [Online]. Available: https://typesense.org/docs/guide/ranking-and-relevance.html#text-match-score-type

[66] "Built-in ranking rules." Accessed: Nov. 12, 2025. [Online]. Available: https://www.meilisearch.com/docs/learn/relevancy/ranking_rules

[67] "Relevancy." Accessed: Nov. 12, 2025. [Online]. Available: https://www.meilisearch.com/docs/learn/relevancy/relevancy

[68] N. F. Liu *et al.*, "Lost in the Middle: How Language Models Use Long Contexts," *Transactions of the Association for Computational Linguistics*, vol. 12, pp. 157–173, 2024, doi: 10.1162/tacl_a_00638.

[69] L. Zheng *et al.*, "Judging LLM-as-a-judge with MT-bench and Chatbot Arena," in *Proceedings of the 37th International Conference on Neural Information Processing Systems*, Red Hook, NY, USA: Curran Associates Inc., 2023.

[70] K. Opsahl-Ong *et al.*, "Optimizing Instructions and Demonstrations for Multi-Stage Language Model Programs," in *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, Y. Al-Onaizan, M. Bansal, and Y.-N. Chen, Eds., Miami, Florida, USA: Association for Computational Linguistics, Nov. 2024, pp. 9340–9366. doi: 10.18653/v1/2024.emnlp-main.525.

[71] J. R. Landis and G. G. Koch, "The Measurement of Observer Agreement for Categorical Data," *Biometrics*, vol. 33, no. 1, pp. 159–174, 1977, doi: 10.2307/2529310.