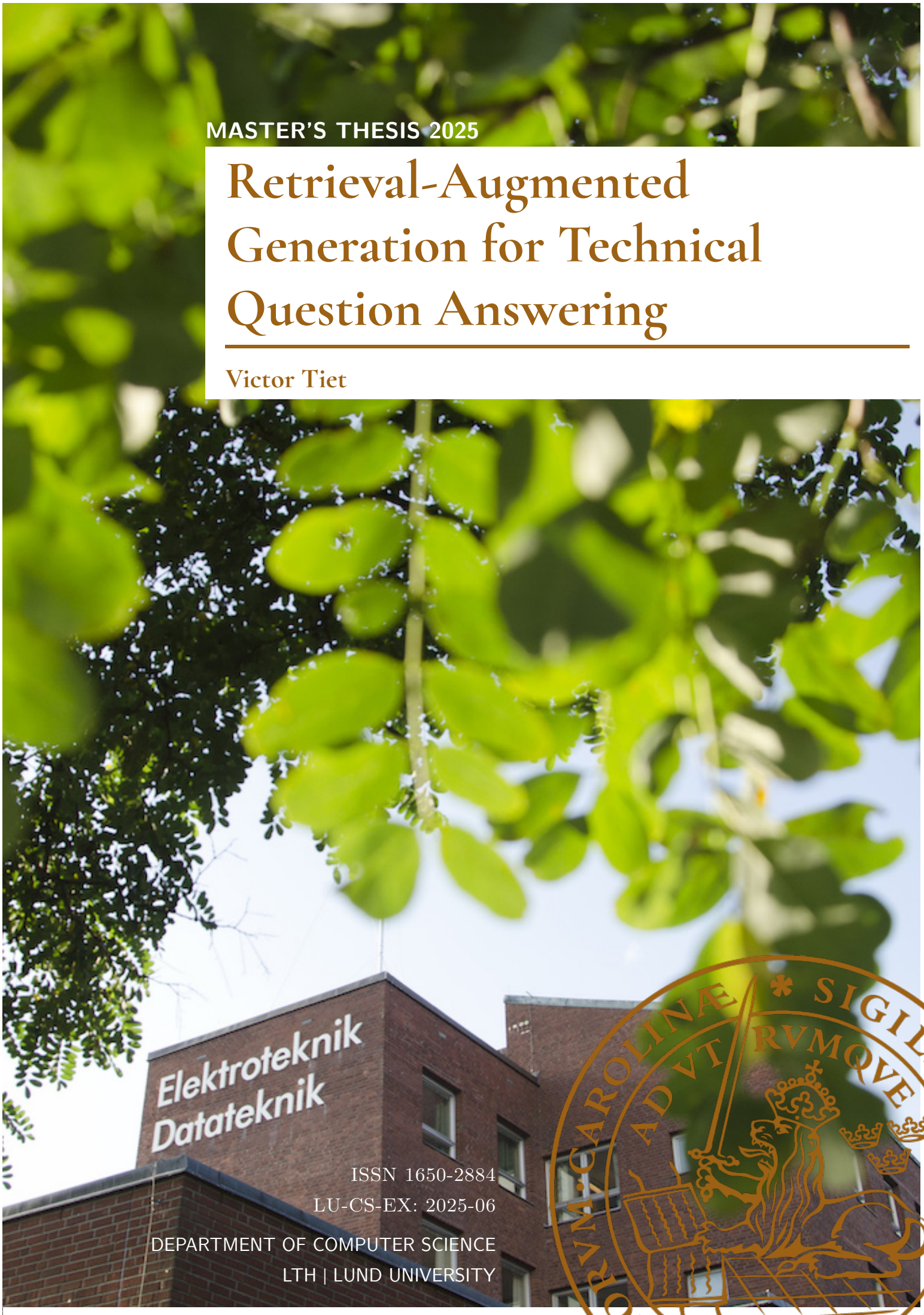


MASTER'S THESIS 2025

Retrieval-Augmented Generation for Technical Question Answering

Victor Tiet

ISSN 1650-2884
LU-CS-EX: 2025-06

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2025-06

Retrieval-Augmented Generation for Technical Question Answering

Retrieval-augmented generation för den tekniska domänen

Victor Tiet

Retrieval-Augmented Generation for Technical Question Answering

Victor Tiet
vi0521ti-s@student.lu.se

March 29, 2025

Master's thesis work carried out at Softhouse Consulting AB.

Supervisors: Olof Bengtsson, olof.bengtsson@softhouse.se
Marcus Klang, marcus.klang@cs.lth.se

Examiner: Jacek Malec, Jacek.Malec@cs.lth.se

Abstract

Today, many companies have access to technical documentation in various formats. Simultaneously, transformer-based language models have enabled new frameworks and applications. One such application is retrieval-augmented generation (RAG). This thesis explores different RAG configurations for Question Answering in the technical domain and identifies the optimal setup. Additionally, we examine how domain differences can affect performance.
To achieve this, experiments are conducted on pre-annotated datasets, TechQA and SQuAD, using retrievers like Jina-Embeddings-v3 and readers like Llama-3. Performance is evaluated through metrics such as Recall@k, F1, and BERTScore. Results indicate that Jina-Embeddings-v3 excels on TechQA ($Recall@5 = 0.625$), while BM25 performs best on SQuAD ($Recall@5 = 0.833$). Furthermore, Phi-3 Mini-128K-Instruct emerges as the optimal reader for TechQA, achieving a BERTScore of 0.865.
While some differences were observed, further research is required to fully understand the impact of domain specificity on RAG performance.

Keywords: retrieval-augmented generation, TechQA, question-answering, information retrieval, language models

Acknowledgements

First and foremost, I would like to thank Softhouse Consulting AB for providing me with the opportunity to conduct this thesis in collaboration with Fieldly AB. Special gratitude should be expressed to the people at Softhouse and particularly my supervisor Olof Bengtsson, for the continuous feedback and support throughout this work. Moreover, I express my thanks to the people at Fieldly AB for providing me with the resources needed and being overall welcoming, especially Salomeh Kiani Johnsson and Lars Dahlén for all the administrative help. Lastly, I would like to thank my academic supervisor, Marcus Klang, for all the invaluable feedback and guidance throughout this thesis. Thank you!

Contents

Chapter 1
Introduction

Today, with the rise of large language models such as ChatGPT and Claude, there has been an increased appetite for generative question-answering-based systems, which can generate coherent and easily understandable answers. However, even with the rapid progress of large generative language models that have been trained on large corpora of text, the answers are not always reliable and can sometimes be misleading. Moreover, if the question involves specific proprietary information, e.g. internal company information, then the model will have an even harder time generalizing a correct answer. To mitigate this, a novel architecture (called retrieval-augmented generation) was proposed by [Lewis et al., 2020], where a pre-trained generative language model is fed relevant information (also referred to as *context*) together with the question. With the relevant context, the system could now, in theory, output answers to task-specific questions that are distinct in nature more accurately than a model without context.

A client of Softhouse Consulting AB, Fieldly AB, is looking to leverage this architecture for an internal question-answering product. Currently, sales personnel rely on a few colleagues for up-to-date information about features and product implementations, which could potentially be automated using a retrieval-augmented generative system. The company has data in the form of workflow Jira tickets and internal self-service documentation. These tickets and documents are technical in nature, often describing product features and implementation phases, using very concise and compact language.

A significant amount of research has already been conducted on QA systems using well-established pre-annotated datasets such as TriviaQA, SQuAD, and HotpotQA. However, there exists a major distinction between the aforementioned datasets and the data provided by Fieldly, mainly the domain difference. The data Fieldly offers is technical in nature, both in structure and language, while the more well-established datasets revolve around general trivia knowledge and popular science questions (often based on Wikipedia articles). These contrasts manifest themselves in more numeric characters (e.g., software versions, color codes,

etc.), hyperlinks, and more concise language. Moreover, due to the lack of publicly accessible technical documentation and a higher knowledge bar associated with generating questions on such data, far less research has been conducted in this domain. Our thesis aims to bridge this gap by exploring optimal configurations for building performative retrieval-augmented generative systems on technical data.

1.1 Research questions

The questions addressed in this report are as follows:

1. Which retrieval methods work best for retrieval-augmented generation in the technical domain?

2. Which generative models are the most accurate, efficient and user satisfactory for retrieval-augmented generation in the technical domain?

3. What are the performance differences for a RAG handling data in the technical domain compared to other domains such as trivia and popular science?

1.2 Related work

Open-domain question answering is a well-known methodology that involves answering factual questions using a knowledge bank as source [Zhang et al., 2023]. A relatively newer approach within this area is retrieval-augmented generation, where a generative component augments its answer using relevant contexts retrieved from an external corpus [Lewis et al., 2020]. This method has been enabled by the recent rapid development of capable generative language models, built using the Transformer architecture first proposed by [Vaswani et al., 2017]. Examples of such language models are Llama 3 [Dubey et al., 2024] and Phi-3 [Abdin et al., 2024], which exhibit strong reasoning capabilities together with the ability to produce coherent outputs. Moreover, by leveraging traditional information retrieval methods like BM25 and newer approaches, such as dense passage retrieval (first introduced by [Karpukhin et al., 2020]), relevant contexts from the knowledge corpus can subsequently be used to generate answers in an open-domain fashion. To evaluate open-domain question-answering systems, pre-annotated datasets can be used. These datasets can have different characteristics and contain domain-specific data, such as TechQA [Castelli et al., 2020], which contains questions and answers in the technical domain.

1.3 Scope

This thesis will investigate the performance of retrieval-augmented generation (RAG) in question-answering within the technical domain. It will assess the effectiveness of various retriever methods on this type of data and provide insights into the capabilities of some state-of-the-art language models. Due to the large variations of capable language models being offered, four models with distinct characteristics and training regimes will be examined

and evaluated. Additionally, this thesis will aim to deepen the understanding of building performative RAG systems using real-world technical data while also exploring how domain-specific differences can impact performance. The latter will be done by comparing results from two datasets in different domains, in our case TechQA and SQuAD.

1.4 Outline

This thesis will evaluate different language models and retrievers that can be used in a retrieval-augmented generative system. We will conduct experiments component-wise, i.e., for each component, and then follow up with an end-to-end evaluation. In parallel, we will also explore how the domain affects performance. This will be achieved using two public datasets in different domains, together with Fieldly's own data (see section 3.1). The methodology for both reader and retriever evaluations are presented under section 3, while results are presented in section 4. This is followed by an overarching Discussion 5, and finally Conclusion 6.

Chapter 2
Background

2.1 Retrieval-augmented generation

Retrieval-augmented generation (RAG) is a newer approach within the open-domain question-answering field. First proposed by [Lewis et al., 2020], the overall approach is illustrated in Figure 2.1. There are mainly two components interacting within a RAG: the retriever and the reader. The retriever functions as an information gatherer, where relevant information is retrieved conditioned on a question. The information typically consists of text documents encoded using either a transformer-based embedding model (also known as *dense passage retrieval*) or a bag-of-words technique (e.g., TF-IDF).

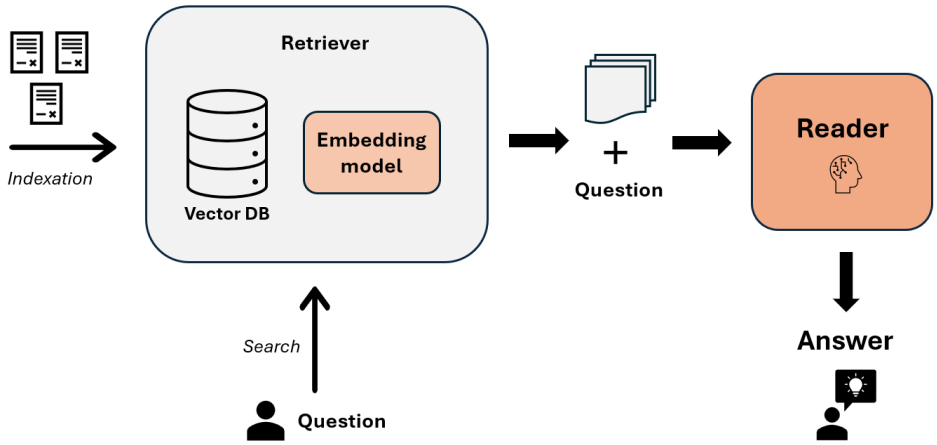


Figure 2.1: Illustration of retrieval-augmented generation.

The indexed documents can be stored in a vector database, which, although not required, enables a faster and more efficient retrieval process. When a user then formulates a question, the retriever encodes it using the same method and performs a similarity search (usually cosine similarity) on the indexed corpus to identify the most relevant contexts. Subsequently, the retrieved relevant contexts and the question are passed to a reader component, consisting of a generative language model, which produces the final answer.

This methodology builds upon previous work conducted on the same type of system, that combined both parametric and non-parametric memory in question-answering tasks. Here parametric memory means knowledge or information stored in the parameters of a language model, while non-parametric refers to the retrieval of externally stored knowledge used by the model. This setup helps address the limitations of language models when they lack knowledge in certain specialized areas. Compared to previous work, which mainly focused on extractive question-answering (where the answer is extracted from a retrieved context), [Lewis et al., 2020] instead implements a similar methodology using sequence-to-sequence models for generative answering. This has laid the foundation for a framework that is today commonly known as *RAG*.

2.2 Generative Language Models

A fundamental prerequisite to the emergence of RAG systems can mostly be attributed to the rapid development of generative language models in recent years. Many of these models demonstrate impressive abilities to comprehend and generate human language, making them well-suited for use in chat-like applications [Chang et al., 2024]. Although many of them are large in nature, a larger size does not necessarily guarantee better performance. Many other factors, such as training regiment, data, and model architecture, also play an important role. An example of this is Phi-3, a series of highly capable language models specifically designed to be smaller in size while still being able to compete with much larger models [Abdin et al., 2024]. Another example of a smaller model is Flan-T5, which was specifically designed to perform well at following instructions on a range of tasks [Chung et al., 2024]. This was achieved using a very comprehensive fine-tuning process. Following this, much larger models have also recently been developed and published with the goal of serving as general-purpose models for a wide range of downstream applications. An example of that is the Llama 3 series, which was trained using a sizable training corpus (15 trillion tokens) and notable compute budget, approximately 50 times larger than its predecessor [Dubey et al., 2024]. Lastly, due to the large costs associated with training these models, alternative approaches have also emerged lately, where knowledge and performance of much larger models are *distilled* into a smaller model. This approach enables a more cost- and time-efficient training regime by eliminating the need for human annotators, thereby resulting in a more efficient training process. An example of a model utilizing this regime is Zephyr-7b [Tunstall et al., 2023].

2.2.1 Scaling Laws

A key reason many language models have grown as large as they are today derives from an observation first described in [Kaplan et al., 2020]. In the paper, the authors argue that the performance of transformer-based language models depends heavily on factors such as the number of parameters, the size of the training dataset, and the amount of computing used for training rather than on the model's shape (e.g., network depth or width). The authors specifically point out that the correlation between performance and the mentioned factors seems to follow a power-law relationship, which holds for several orders of magnitude. These relationships are today commonly referred to as the *scaling laws*. More specifically, the authors concluded that scaling the model size should be prioritized over scaling the dataset size when operating under a fixed compute budget.

2.2.2 Compute-Optimal Training

Recently, scaling up models to larger and larger sizes has resulted in impressive performance gains and emerging abilities. However, larger models also tend to have longer inference times and increased hardware requirements (mainly GPU and RAM) for hosting the models. With the increasing costs associated with these models, more attention has lately been focused on trying to improve model performance without increasing the size of the model. This has resulted in new research trying to find the "compute-optimal" training regime, that is, find the optimal dataset and model size under a fixed compute budget. One such article published by DeepMind [Hoffmann et al., 2022] presented results contradicting some conclusions drawn in [Kaplan et al., 2020]. The authors of the DeepMind paper argue that LLMs at the time were over-dimensioned for their performance and that smaller models, trained on more training tokens, could potentially match the performance of these larger models. More specifically, the size of the training data should scale equally to the size of the model (contradicting previous assumptions that model size should take precedence). To prove this, the authors trained a smaller version of the *Gopher* LLM, that was four times smaller but used four times more training data. The resulting model, called *chinchilla*, outperformed the original Gopher model, while enjoying the benefits smaller models have, such as faster inference and smaller memory footprint.

2.2.3 Transformers

Today, a fundamental component in many large language models is the *Transformer*. First introduced in [Vaswani et al., 2017], it addressed many of the challenges and limitations associated with the sequence architectures commonly used at the time, such as recurrent neural networks (RNNs) and long short-term memory (LSTM). It incorporates a mechanism called *attention*, first proposed in [Bahdanau et al., 2014], and uses an encoder-decoder style architecture without the need for any recurrence (see Figure 2.2). When first introduced, attention was meant to solve the information bottleneck posed by the information pass between the encoder and decoder in RNNs. Instead of relying on the last hidden state vector from the encoder, attention enables the decoder to use all hidden state vectors from the encoder thus enabling a more contextualized representation of the input [Schmidt, 2019]. Transformers

take this a step further by discarding all recurrent architecture and relying entirely on the attention mechanism to learn from the training data.

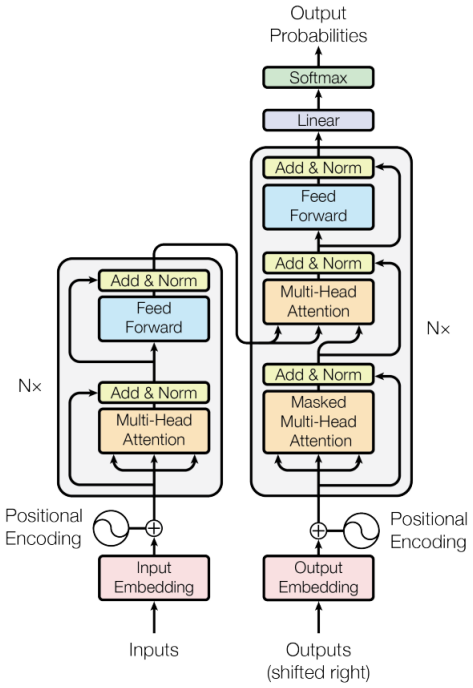


Figure 2.2: Original transformer architecture proposed by the authors of *Attention is all you need*. Extracted from the same paper [Vaswani et al., 2017].

Attention

To attain a deeper understanding of attention, there are three main concepts to keep in mind: queries, keys, and values [Vaswani et al., 2017]. Attention allows a model to find the correlation between tokens from two sequences using vector projections on the query and key space and subsequently perform scaled dot product between the projections. For the special case when both sequences are the same, we call it *self-attention* [Schmidt, 2019]. Starting from a token x and its corresponding embedding vector $\boldsymbol{E}_x$, the query vector for the token can be calculated using the query matrix $\boldsymbol{W}_x$ and multiplying it with $\boldsymbol{E}_x$. This is done for all tokens in the sequence. The resulting vector $\boldsymbol{Q}_x$ is called the query and is intended to "look up" some semantic relationship between x and tokens in the second sequence (for self-attention we would search for correlations within the same sequence). Having a query for all tokens in the first sequence, a key is subsequently calculated for each token in the second sequence by multiplying each token embedding $\boldsymbol{E}_y$ with the key matrix $\boldsymbol{W}_y$, returning a key vector $\boldsymbol{K}_y$. In the same way, a query is used to find correlations defined by the query matrix, a key is used to identify tokens that the *query is looking for*. Having both queries and keys, finding token pairs with high correlating queries and keys can then be achieved using a simple dot product. This results in a $n \times m$ matrix ($n \times n$ for self-attention) where n and m are the sequence lengths expressed in the number of tokens.

The correlations are then normalized using softmax, which yields weights (or probabilities) ranging from 0 to 1. The final step involves updating the actual token embeddings for a sequence to reflect the newly found query-key relationship. This is done using a values matrix W_v, which, when multiplied with a token embedding E_m, transforms the embedding to a value vector V_m in the same space. The value vectors for each token are then summed up using the key-query similarities as weights and the resulting vector is used to update the old embedding. It is worth mentioning that the just described procedure applies for *single head* attention, and in the case of the transformer, multiple such computations are performed in parallel, for different versions of W_q, W_k, and W_v. This way multiple correlations can be captured and averaged out to get the final updating vector. This is called *multi-head* attention [Vaswani et al., 2017] and is illustrated in Figure 2.3.

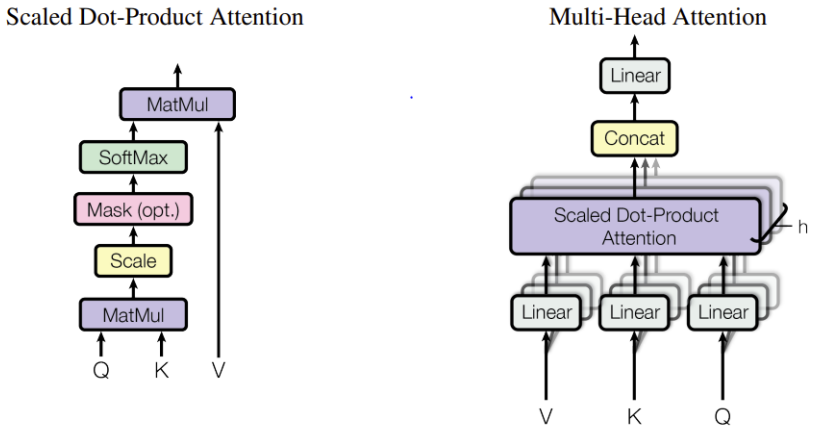


Figure 2.3: Illustration on how single- and multi-head attention is computed. Extracted from the original paper [Vaswani et al., 2017].

2.2.4 Phi-3

Phi-3 is a family of language models developed by Microsoft with parameter counts ranging from 3.8 billion to 14 billion parameters. The model has a decoder only architecture and it builds upon previous work done on the Phi-2 model, where the goal was to train smaller language models with similar capabilities to large language models. To do so, the authors use a training regiment they denote as *data optimal regime* [Abdin et al., 2024] (a clear nod to compute-optimal training). Here attention is focused on the quality of the data in order to maximize performance for smaller models. This presents an alternative approach for model improvement, in contrast to the more established method of scaling up models, which has led to the development of massive models with trillions of parameters.

By using high-quality data for training, Phi-3-mini (with 3.8 billion parameters) has shown to rival much larger models such as GPT-3.5 and Mixtral (which has 45 billion parameters). This is a consequence of a filtering process where heavy filtering was done on public and synthetic data in order to only use data that could maximize the reasoning capabilities of the models. A consequence of this heavy filtering is a trade-off with the amount of parametric

memory the model has access to, which limits its ability to answer factual questions. However, as the authors noted, this can be mitigated by the use of context driven augmentation, such as retriever systems in RAGs.

2.2.5 Llama 3

Similar to Phi-3, Llama 3 is a family of language models with varying parameter counts developed by Meta. They are built using a standard encoder-decoder Transformer architecture proposed in [Vaswani et al., 2017]. Llama 3.1 is a further development of Llama 3 and introduces multilingualism and supports longer context-lengths (up to 128k tokens from 8k) [Dubey et al., 2024]. In this work, Llama 3.1 will be used, and in the rest of the report, Llama 3.1 will be denoted as Llama 3. To achieve longer context length, input was gradually scaled from 8k to 128k in order for the model to adapt and get a balanced performance on both short and long contexts. Furthermore, to achieve compute-optimal training on a fixed compute budget, custom scaling laws were defined to determine the optimal number of training tokens and, subsequently also, the size of the model. In contrast to Phi-3, Llama 3 comes in much larger versions ranging from 8 billion parameters to 405 billion parameters. These models were built to serve as foundational models for other AI applications, such as chatting, reasoning, or code generation. Llama 3 is a further development of previous work done on the Meta Llama line-up, with more computing and data deployed for training and scaling the new models. Compared to the previous generation (Llama 2), which was trained on 1.8 trillion tokens, Llama 3 was trained on a significantly larger corpus of 15 trillion tokens together with more than 50 times more training compute deployed. This has resulted in an increase in performance which puts the largest Llama 3 model on par with GPT-4 and close to the state-of-the-art. The smallest model (Llama 3 8B) also outperforms other models with similar parameter counts, such as Gemma 2 9B and Mistral 7B, on several categories, such as general knowledge, coding, and math.

2.2.6 Zephyr

Zephyr is a 7 billion parameter language model built upon the Mistral 7B v0.1 model and fine-tuned by the H4 team at Hugging Face. It builds on the premise of training smaller LLMs entirely using distillation from larger LLMs. Three steps were used to train Zephyr: *distilled supervised fine-tuning* (**dSFT**), followed by *AI feedback* (**AIF**), and lastly *distilled direct preference optimization* (**dDPO**) [Tunstall et al., 2023]. This resulted in a model with state-of-the-art performance on MT-bench among other 7B parameter models, such as Xwin-LM v0.1 and MPT-Chat, and at the same time rivaling other models 10x larger than itself (Llama2-Chat, WizardLM v1.0, Guanaco, etc.) on the same chat-based benchmark.

Distilled Supervised Fine-Tuning (dSFT)

During a normal fine-tuning process, a baseline LLM is trained on inputs and outputs that are, in many cases, human-generated. This process can be costly as annotated datasets are expensive and time-consuming to create. Distilled supervised fine-tuning solves this issue by using a larger LLM (the teacher model) to generate inputs and outputs for our target model

(the student model) to train on. This way, knowledge from a larger model can be transferred (distilled) to a smaller model in a relatively inexpensive way [Tunstall et al., 2023].

AI Feedback (AIF)

Intention alignment is a term that has become more relevant lately due to the rise of generative language models. Although hard to quantify, the term revolves around how a language model should behave and align to the expectations of the end user, normally a human. These expectations can range from topics such as *helpfulness* (how helpful the response is to the user), *harmfulness* (the generated response should be safe and not cause any harm to the user), or *truthfulness* (the generated answer should be correct) [Ouyang et al., 2024]. To achieve intention alignment for language models, model outputs are often scored or ranked by a human annotator and then used to train a reward model. For Zephyr, however, the authors used a collection of pre-trained LLMs (Claude, Falcon, Llama, etc.) to generate outputs that were later scored by another LLM (e.g., GPT-4). This way, the authors avoided the time-consuming step of collecting human-annotated responses.

Distilled Direct Preference Optimization (dDPO)

The final step applies the scored output to the distilled fine-tuned student model (dSFT). Normally, this is done using *Reinforcement Learning from Human Feedback* (RLHF) and a reinforcement learning technique called *Proximal Policy Optimization* (PPO) [Ouyang et al., 2024]. This method is, however, complex, which is why the authors instead use *Direct Preference Optimization* (DPO) [Rafailov et al., 2023], which directly optimizes the target model, instead of a reward model. For Zephyr, they used the distilled student model (dSFT) as the target model, which is why they denote this step as *distilled* DPO.

2.2.7 Flan-T5

Flan-T5 is a series of encoder-decoder transformer-based language models released by Google in 2022, with parameter counts ranging from 80 million to the largest with 11 billion [Chung et al., 2024]. The models are instruction-tuned instances of the T5 model family where Flan stands for *Finetuning language models*. Flan-T5 builds upon the premise that model performance can improve by scaling up the number of instruction tasks together with the model size. The authors also explored chain-of-thought (CoT) instruction tuning, see Figure **??**, which involved instructing the model to explain the reasoning behind its answers. This can be done by providing one or multiple examples of how the answer should look like, using the same prompt. The authors found that CoT fine-tuning not only increased the reasoning capabilities of the models but also improved performance overall for all benchmarks (especially on zero-shot reasoning). Beyond T5, the authors also instruction tuned the much larger PaLM (8-540 billion parameters) and U-PaLM (540 billion parameters), but T5 displayed the largest performance boost compared to its non-tuned counterpart (in some cases showcasing double digit increase).

2.3 Retriever

In a RAG, the retriever is the first step of the system and it serves as the information gatherer for the generative reader model. The retriever is tasked with returning contexts with relevant information, which the reader can use to augment the answer. This step is crucial as the validity of the generated answer fully depends on the correctness of the retrieved contexts. There are mainly two types of retrievers used in question-answering and information retrieval systems today, *dense* and *sparse* retrievers [Izacard and Grave, 2020]. Sparse retrievers are usually based on the TF-IDF and BM25 algorithms and have been prevalent in question-answering and information retrieval systems for quite some time, while dense retrieval is a newer approach that has been emerging lately. Furthermore, more intricate, multi-step retrieval processes have been introduced as well such as hybrid retrieval, query expansion, and reranking [Wang et al., 2024].

2.3.1 Sparse Retrieval

Sparse retrieval builds upon bag-of-word methods, which represent text without regard to the order of tokens. These models tend to create large representations of text, often determined by the input corpus, where each dimension corresponds to a token. By default, this creates vectors with exceedingly high occurrences of zero values when trying to represent sentences that are part of a larger corpus. Sparse retrieval has shown to excel at term and keyword matching in information retrieval [Karpukhin et al., 2020].

TF-IDF

Term frequency-inverse document frequency (TF-IDF) is a popular encoding method widely used in information retrieval systems. It works by identifying the frequency of a term, usually a word, in a document and also in how many documents this term occurs [Ramos et al., 2003]. The weights for each term are then calculated using

$$w_{t,d} = f_{t,d} * log(\frac{|D|}{f_{t,D}}) \tag{2.1}$$

where $w_{t,d}$ is the weight for term t in document d, $f_{t,d}$ is term frequency of t in d, $|D|$ is the amount of documents in the corpus D, and $f_{t,D}$ is the document frequency of t in D. For common words across D, like articles and prepositions, $f_{t,D}$ will be high hence the inverse will be small. This serves as a penalty for very common terms found across all documents such as "the" or "a". Opposite to this are terms that are common in a document but have low document frequency. These cases will have weights that are larger and thus have higher "importance". When using TF-IDF for retrieval, all documents and subsequent queries are vectorized using the TF-IDF weight scoring. A search using cosine similarity is then conducted in order to retrieve the document(s) most similar to the query.

BM25

BM25 (*Best Matching 25* or *Okapi BM25*) is a further development on the TF-IDF algorithm and takes into account the document lengths of the corpus as well [Izacard and Grave, 2020]. There are numerous versions of BM25 but the original version (Okapi BM25) can be simplified and expressed with the equation

$$w_{t,d} = \frac{(k_1 + 1)f_{t,d}}{k_1((1 - b) + b\frac{dl}{avdl}) + f_{t,d}} * log(\frac{|D| - f_{t,D} + 0.5}{f_{t,D} + 0.5}) \tag{2.2}$$

where k_1 and b are constant parameters, dl the document length, $avdl$ the average document length of the corpus, and the rest is the same as in TF-IDF [Robertson et al., 2004]. Here, the document length is used as a normalizing factor to penalize longer documents as they, per default, will have a greater probability of having a higher term frequency for a certain term t.

2.3.2 Dense Retrieval

Dense retrieval, or *Dense Passage Retrieval* (DPR), is an alternative retrieval method that represents text using dense vector encoding [Karpukhin et al., 2020]. These dense vectors are generally generated using pre-trained embedding models, which take text as input and output embeddings with a dimension d. The same encoder model is then used to encode an input query, and consequently, a similarity search is done to find the document with the most similar embeddings. In contrast to sparse retrieval, which excels at matching exact words or tokens, dense retrieval can also capture the semantic similarity between two text chunks. This can be advantageous in cases where synonyms are used in the input query and document, e.g. "villain" and "bad guy", or when whole sentences have been formulated in a different way but the semantics are still the same.

GTE

GTE, an abbreviation for *general text embedding*, is a widely used embedding model developed by Alibaba and released in 2023 [Li et al., 2023]. The embedding model is a dual-encoder model initialized from a pre-trained BERT model and trained using a two-step approach (unsupervised contrastive pre-training followed by supervised fine-tuning). To ensure high generalization, the model was trained on a diverse set of data sources such as web pages, social media, scientific articles, and code repositories. Evaluating GTE on the MTEB benchmark showed that it outperformed OpenAI's *ada-002* model (which is assumed to be much larger) and also competed with 10x larger models as well. Subsequently, several enhanced models (denoted version 1.5) were released in 2024, supporting multiple languages and accepting substantially longer context lengths (8192 versus the previous 512) [Zhang et al., 2024b]. To support longer context lengths, version 1.5 makes use of *rotary positional embeddings* (RoPE) first proposed in [Su et al., 2024] instead of the widely used *absolute positional embeddings* introduced in [Vaswani et al., 2017]. The evaluation of GTE v1.5 showed that it matched the multilingual state-of-the-art *BGE-M3* and that the English version of the model (denoted gte-base/large-en-v1.5) achieved noticeably higher scores on the English MTEB subset compared to the multilingual model.

Nomic

Nomic-embed-text-v1.5 (hereby referenced to as *Nomic v1.5*) is an embedding model developed and trained by *Nomic AI*. It is an open source, reproducible, and 137 million parameter text embedding model supporting long context lengths up to 8192 tokens long [Nussbaum et al., 2024]. It was primarily developed due to the lack of open-source embedding models supporting long context lengths (> 512 tokens). The models that did support longer context lengths at the time were either closed source (like OpenAI's ada-002), not performative enough (jina-embedding-v2-base), or too large to be practical (E5-Mistral). Similar to GTE, Nomic is trained using a BERT model as a foundation and has a two-stage training process where the first stage involves an unsupervised contrastive pre-training while the second stage involves supervised fine-tuning. In addition, similarly to GTE 1.5, Nomic also uses rotary positional embeddings to support long context lengths. The goal of unsupervised contrastive training is to make the model good at selecting similar documents and, at the same time, separating them from irrelevant documents. Lastly, to boost performance on final downstream applications, a supervised step is employed, which uses human-annotated datasets such as MSMarco [Bajaj et al., 2016] and HotpotQA [Yang et al., 2018]. Subsequently, the evaluation of Nomic-embed-text-v1 showed it outperforming OpenAI's ada-002 and text-embedding-3-small on both long and short context benchmarks.

Jina

Jina-embeddings-v3 (hereby referenced to as *Jina v3*) is the latest text embedding model developed by *Jina AI* [Sturua et al., 2024]. It is a 570 million parameter model that supports context lengths up to 8192 tokens and is the successor of Jina Embeddings v2. Compared to its predecessor, the latest version has been optimized to support multiple languages and to have a high performance across several downstream tasks. Jina was trained using an XLM-RoBERTa model as base with several key modifications to it, such as added rotary positional embeddings and five pre-trained *LoRA* adapters. Low-Rank Adaptation (LoRA) adapters are trainable decomposed matrices [Hu et al., 2022], which can be used during fine-tuning to improve performance on downstream tasks. These adapters can be much smaller than the model itself, thus providing a compute-efficient way to improve performance. The LoRA adapters offered by Jina v3 are optional to use and can be chosen dynamically during run-time to provide task-specific performance boosts in four categories: *retrieval*, *separation*, *classification* and *text-matching*.

With the relevant LoRA adapter enabled, Jina exhibits state-of-the-art performance on both multilingual data and long-context retrieval. Furthermore, it ranks second highest amongst all embedding models under the *semantic textual similarity* (STS) category listed on the MTEB leaderboard [HuggingFace, 2024]. It also outperforms the latest embedding models from Cohere (Cohere-embed-multilingual-v3.0) and OpenAI (text-embedding-3-large) on English MTEB while at the same time being significantly smaller than its competitors (in both model and embedding size).

2.3.3 Vector Database

Vector databases are databases designed to store high-dimensional vectors [Han et al., 2023]. These vectors, sometimes referred to as embeddings, are, in many cases, representations of text, audio, images, and more. By using an embedding function (which could be a neural network), these inputs can be transformed into fixed-size vectors, which can subsequently be stored in a vector database. The reason why we would use a vector database over any other type of database comes down to *use-case* and *performance*. Many vector databases are designed and built to perform well in finding similar vectors based on either a distance or similarity score. This differentiates itself from many other databases, which usually return results based on some condition (could be SQL queries). Many vector databases are also designed to be scalable, meaning that they should be able to handle high quantities of vectors without compromising performance. This makes vector databases compelling to use in dense information retrieval systems, which rely heavily on embedding text into fixed-size dense vectors.

FAISS

FAISS is an open-source software library released by Meta [Douze et al., 2024] and built to enable efficient Approximate Nearest Neighbor Search (ANNS) algorithms on vectors and is therefore not defined as a vector database. The authors state this more directly: " *The scope of the library is intentionally limited to focus on ANNS algorithmic implementation*". Consequently, the FAISS package is often used as a building block for other DBMSs such as Pinecone and Zilliz [Douze et al., 2024]. FAISS is built using an *index*, which both stores the vectors and returns similar vectors based on a query vector and scoring function determined by the index used. An example is the *Flat Index*, which performs brute-force matching on all vectors, or the *Inverted File Index*, which instead performs clustering by grouping similar vectors together and only searches in the closest cluster.

2.3.4 Cosine Similarity

Cosine similarity is a scoring function used to capture the similarity between two vectors. It is commonly used in information retrieval systems [Xia et al., 2015], where similar vectors need to be retrieved conditioned on a query vector. Cosine similarity measures the angle between two vectors and can be formulated as:

$$\cos(\theta) = \frac{\sum_{i=1}^{d} x_i \times x'_i}{\sqrt{\sum_{i=1}^{d} x_i^2} \times \sqrt{\sum_{i=1}^{d} x_i'^2}} \tag{2.3}$$

where θ is the angle between vectors **x** and **x'**, and d the dimension of the two vectors. From Equation 2.3, we observe how cosine similarity can simply be computed as the normalized inner product between the two vectors.

2.4 SQuAD

The *Stanford Question Answering Dataset* (SQuAD) is a popular reading comprehension dataset consisting of more than 100,000 question/answer pairs released by Stanford University in 2016 [Rajpurkar et al., 2016]. It was created with the sole purpose of providing a large and, at the same time, high-quality dataset used for machine reading comprehension tasks. SQuAD was created by selecting high-quality Wikipedia articles (536 articles) and segmenting them into paragraphs shorter than 500 characters. This resulted in 23,215 paragraphs in total, which were subsequently divided into a training set (80%), validation set (10%), and test set (10%). A group of crowd workers was then tasked to generate up to 5 questions for each paragraph shown to them and subsequently highlight the answer to the question in the passage. In Table 2.2 an example paragraph about the University of Notre Dame and five subsequent question-answer pairs are presented. A new version of the dataset, denoted SQuAD v2.0, was released in 2018, that combines the previous dataset with 50,000 additional non-answerable questions[Rajpurkar et al., 2018].

Table 2.1: Example of five question-answer pairs for a chosen paragraph in SQuAD.

Context	Question	Answer
Architecturally, the school has a Catholic character. Atop the Main Building's gold dome is **a golden statue of the Virgin Mary**. Immediately in front of the Main Building and facing it, is **a copper statue of Christ** with arms upraised with the legend "Venite Ad Me Omnes". Next to **the Main Building** is the Basilica of the Sacred Heart. Immediately behind the basilica is the Grotto, **a Marian place of prayer and reflection**. It is a replica of the grotto at Lourdes, France where the Virgin Mary reputedly appeared to **Saint Bernadette Soubirous** in 1858. At the end of the main drive (and in a direct line that connects through 3 statues and the Gold Dome), is a simple, modern stone statue of Mary.	To whom did the Virgin Mary allegedly appear in 1858 in Lourdes France?	**Saint Bernadette Soubirous**
	What is in front of the Notre Dame Main Building?	**a copper statue of Christ**
	The Basilica of the Sacred heart at Notre Dame is beside to which structure?	**the Main Building**
	What is the Grotto at Notre Dame?	**a Marian place of prayer and reflection**
	What sits on top of the Main Building at Notre Dame?	**a golden statue of the Virgin Mary**

2.5 TechQA

TechQA, created by IBM and released in 2019, is a question-answering dataset specifically tailored for the technical support domain [Castelli et al., 2020]. It was created due to the lack of public question-answering datasets in the technical support domain and consists of 910 questions-answer pairs, of which 150 are non-answerable. Compared to SQuAD, where the questions were generated after seeing the context (which introduces observation bias), the questions in TechQA were formulated independent of context. TechQA was produced by crawling public technical forums hosted by IBM and falls into three main categories: *generic information requests*, instructions on *how to do something*, and lastly *how to resolve an issue/problem*. From the retrieved question-answer pairs, only those that had an "accepted answer" containing a link to an official IBM technote (an official document with technical information) were kept. This resulted in 15,918 questions. Furthermore, only questions shorter than 12 sentences were retained (due to other datasets, e.g., SQuAD, having much shorter questions), and further filtering was also done by human annotators using a set of requirements (e.g., if the answer was split up among multiple technotes). This resulted in 910 question-answer pairs (490 pairs were created additionally as part of a hidden test set). Similarly to SQuAD, the answers are extractions from the accompanying technote. This can be observed in Table 2.4, where an example of a question-answer pair and its technote is presented.

Table 2.3: An example question-answer pair and its technote, extracted from TechQA.

Technote	Question	Answer
IBM STREAMS 4.1.1.1 and 4.1.1.2 JOBS DO NOT INHERIT THE ENVIRONMENT VARIABLES SET IN .BASHRC, WHEN STREAMS IS RUN AS A SYSTEM SERVICE - United States FLASH (ALERT) ABSTRACT In Streams 4.1.1.1 and 4.1.1.2 Streams jobs may not pick up the user environment from the streams user's .bashrc. This behavior is different from earlier releases. With these versions, when Streams is run as a system service, application environment variables must be set with streamtool. CONTENT Problem Description When running Streams as a system service with Streams releases 4.1.1.1 or 4.1.1.2, toolkits and user applications that depend on the user environment may have various errors. For example the database toolkit might show the following error if it does not pick up the ODBCINI environment variable: * "An SQL operation failed. The SQL state is 08003, the SQL code is 0 and the SQL message is [unixODBC][Driver Manager]Connnection does not exist." * Problem Solution **To work around the issue, set environment variables that are needed by the application directly in the instance with: * * streamtool setproperty * -d <domain> -i <instance> –application-ev <VARIABLE NAME>=<VARIABLE VALUE>** * RELATED INFORMATION APAR IT18432 [https://www-01.ibm.com/support/entdocview.wss?uid=swg1IT18432	Have you found that after upgrade to Streams 4.1.1.1 or 4.1.1.2, that environment variables set in your .bashrc are no longer being set? For example ODBCINI is not set for the database toolkit and you get An SQL operation failed. The SQL state is 08003, the SQL code is 0 and the SQL message is [unixODBC][Driver Manager]Connnection does not exist.	**To work around the issue, set environment variables that are needed by the application directly in the instance with: * * streamtool setproperty * -d <domain> -i <instance> –application-ev <VARIABLE NAME>=<VARIABLE VALUE>**

2.6 Evaluation

One important step when choosing models and retrievers is evaluation. A set of predefined evaluation metrics which can effectively capture the accuracy and effectiveness of each solution is imperative for comparing and producing an optimal solution. In this section we present a set of evaluation metrics which we can use to evaluate the performance of the reader and retriever components.

2.6.1 Recall@k

Recall@k is an evaluation metric for measuring the performance of the retriever. It captures the fraction of relevant documents returned over all relevant documents available when the retriever is set to return k documents [Yu et al., 2024]. It is defined as:

$$Recall@k = \frac{|RD \cap Top_{kD}|}{|RD|} \tag{2.4}$$

where RD is the relevant documents and Top_{kD} is the retrieved top k documents from the retriever. Subsequently, in the case when there is only one relevant document we get a binary outcome (1 when the relevant document is in Top_{kD} and otherwise 0).

2.6.2 F1 Score

F1 Score is the harmonic mean between *precision* and *recall* [Chicco and Jurman, 2020]. Recall is defined as:

$$Recall = \frac{retrieved\ relevant\ instances}{all\ relevant\ instances} \tag{2.5}$$

and captures how well we classify/retrieve relevant instances with no regard to the number of classifications/retrievals that were done for that iteration. Thus, to maximize recall, retrieving all instances (with no regard for relevancy) would yield a perfect recall score of 1. Conversely, precision is defined as:

$$Precision = \frac{retrieved\ relevant\ instances}{all\ retrieved\ instances} \tag{2.6}$$

which captures how well we retrieve relevant instances with regard to the number of retrieved instances in total. To maximize precision, all retrieved instances should therefore be relevant which in that case would yield a score of 1. Subsequently, F1 score is defined as follows:

$$F1 = 2 * \frac{precision * recall}{precision + recall} \tag{2.7}$$

2.6.3 BLEU

Bilingual Evaluation Understudy (BLEU) is an evaluation metric originally used for assessing the quality of machine translations versus reference answers [Yu et al., 2024, Papineni et al., 2002]. This is done by taking the geometric mean of the n-gram word precisions of the generated text against the correct reference answer. Thus, the score is calculated using the following formula:

$$BLEU = BP * \exp\left(\sum_{n=1}^{N} w_n \log p_n\right) \tag{2.8}$$

where BP is a *brevity penalty*, w_n is the weight for n-gram n, and p_n is the precision for n-gram n. The brevity penalty is a penalty function used to penalize answers shorter than reference as otherwise high scores can be achieved using very short answers containing small subsets of the reference. Imagine the following example:

Predicted 1: `The cat is red.`
Predicted 2: `The cat.`
Reference: `The cat has a red undertone.`

For Predicted 1, this would yield $p_1 = 3/4$ ("The", "cat", "red"), $p_2 = 1/3$ ("Thecat"), and $p_3 = 0/2$. However, for Predicted 2, we would get $p_1 = 2/2$ ("The", "cat") and $p_2 = 1/1$ ("Thecat") which would result in a better score than Predicted 1, even though the first sentence seems to be "more correct" compared to the second sentence. This would be the case without a brevity penalty, which is why we apply it to the geometric mean.

2.6.4 BERTScore

One major drawback of an n-gram-based evaluation metric is the sensitivity to synonyms and paraphrasing. For example, imagine the sentence in the before section, but this time, the sentence has been rephrased to "*The feline is red*" instead of "*The cat is red*". BLEU score, which uses n-gram matching, lacks the ability to handle such cases, even though they are semantically similar. This is why BERTScore was introduced. BertScore leverages a transformer-based model to create embeddings for each token and then performs pairwise cosine similarity computations for each token pair [Zhang et al., 2020]. The maximum score for each token pair then determines the most "matching" tokens, whereby recall (denoted R_{BERT}) and precision (denoted P_{BERT}) is calculated. The F1 score is then calculated using Eq. 2.9.

$$F_{\text{BERT}} = 2 \cdot \frac{P_{\text{BERT}} \cdot R_{\text{BERT}}}{P_{\text{BERT}} + R_{\text{BERT}}} \tag{2.9}$$

In the rest of the paper, F_{BERT} will simply be referred to as *BERTScore*.

Chapter 3
Method

This section will focus on the implementation and experimentation of different RAG setups using both TechQA and SQuAD as evaluation datasets. It will explain some of the choices made for each experiment and the rationale behind the chosen methodology. Briefly summarized, three main experiments were conducted: (i) reader performance, (ii) retriever performance, and (iii) end-to-end performance.

3.1 Datasets

Three datasets were used during the evaluation: TechQA, SQuAD, and Fieldly. TechQA and SQuAD are pre-annotated datasets containing question-answer pairs and relevant context, while Fieldy only had relevant context. TechQA and Fieldly can both be categorized in the technical domain, while SQuAD, built on Wikipedia articles, is trivia-based. In our thesis, we choose to define *technical* as something complex, that requires highly specialized knowledge to comprehend. We further narrowed it down to encompass data in the IT and digital systems area, as most of the Fieldly documentation could be classified as such. Moreover, we define *trivia* as anything related to general knowledge and where the target audience is seen as the general public. An example of this could be Wikipedia articles, hence the choice of SQuAD.

3.1.1 TechQA

TechQA is a dataset focused on the technical support domain and offers a reasonable amount of pre-annotated question-answer pairs and accompanying contexts. It is also one of few datasets in this domain that we could find, as most well-known datasets we found were either literature or trivia-based. Due to the lack of pre-annotated question-answer pairs for the Fieldly data and the large costs associated with creating such data, it was determined that a proxy dataset (in the same domain) was to be used instead. By conducting experiments on the proxy dataset and producing evaluation metrics using reference answers, an

optimal configuration could then be determined. In our case, this proxy dataset was chosen to be TechQA. Following this, a qualitative evaluation, using a smaller set of question-answer pairs, can subsequently be performed on the Fieldly data to determine how well or not the results generalize. As mentioned in section 2.5, TechQA was created by crawling public IBM forums and finding both questions and received answers, together with accompanying technotes containing the answer. These were highly specific questions on IBM-related systems, which could be hard to comprehend without any previous knowledge. Moreover, the TechQA dataset is an extractive dataset, meaning that the reference answers had been extracted from a relevant context (technote) and is, therefore, a subsegment of a longer passage. In our evaluations, we chose to use all answerable questions in the development set for TechQA. This was due to the relatively small size of the dataset compared to SQuAD, which resulted in a total of 610 question-answer pairs.

3.1.2 SQuAD

SQuAD is a well-known reading comprehension dataset based on Wikipedia articles. We chose SQuAD due to its popularity (more than 9000 citations on *Google Scholar*) and it being a good match with our trivia definition. Due to the large size of SQuAD, we chose to only use a randomized subset of the development data for evaluation. This resulted in 3,000 question-answer pairs in total. Similarly to TechQA, SQuAD is also an extractive dataset, where the answers are extractions from relevant Wikipedia passages. However, the lengths of these extractions and, subsequently, the passages were not the same across the datasets. In Figure 3.1 the length distributions for question, answer, and context are presented for each dataset.

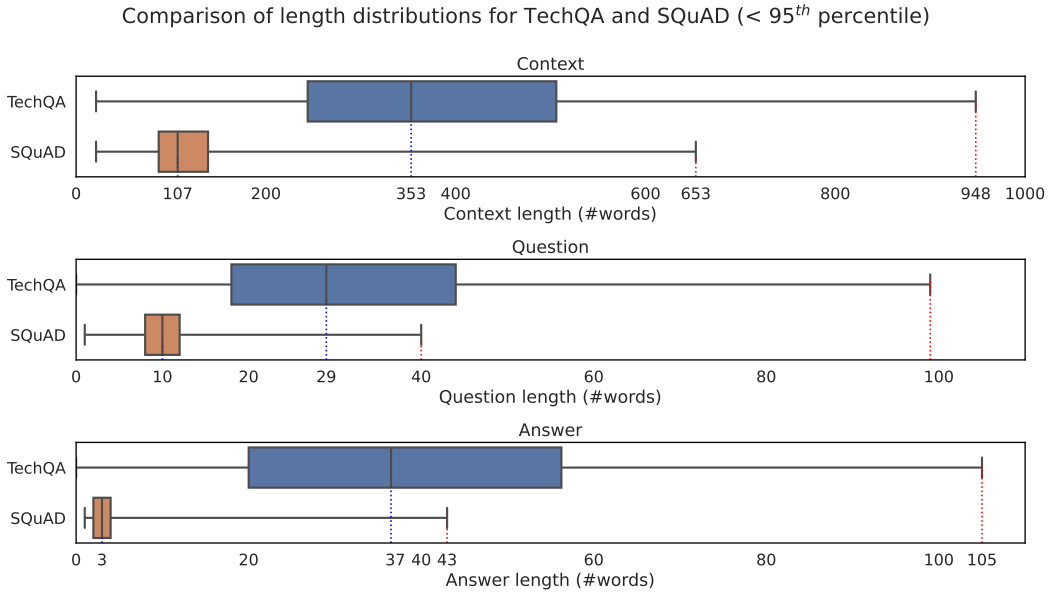


Figure 3.1: Box plots of context, question, and answer length distributions for SQuAD and TechQA (large extreme values > 95th percentile removed). Note the scale difference on the x-axis for context, which is almost 10 times larger.

From the figure we can observe that SQuAD generally features much shorter entries com-

pared to TechQA. This is especially true for answers, where SQuAD, in many cases, only anticipates answers consisting of a few words at most. Furthermore, the questions in SQuAD are, in many cases, longer than the answers, which differs from TechQA, where the opposite is true. These differences were taken into account during our experiments, especially when choosing evaluation metrics, and the potential impacts are further covered when discussing our results.

3.1.3 Fieldly

The Fieldly dataset consisted of data from two main sources: Jira tickets and internal self-service documentation in the form of PDFs. Jira is an internal workflow management tool used to track work done by individuals in a ticket-based manner. In our case, many of the tickets created at Fieldly revolved around improvements and product features implemented on the Fieldy service (which were mostly written in English). In total, 10,494 Jira tickets were collected from the Jira API and pre-processed into the format seen in 3.1.

```
{
    "resolution":  "unresolved",
    "created": "2024-12-31T13:41:15.388+0100",
    "description": "Align this button.",
    "summary": "Button",
    "display name": "John Doe",
    "project name": "Project X",
    "ticket key": "xxx-001",
}
```

Listing 3.1: Example of a Jira ticket and its structure.

For the self-service documentation, some were written in Swedish and to limit complexity, we chose to translate it to English. This was done by using the *DeepL* translation service, which we used due to its accessible API and high quality translations. Google Translate was used initially (which also has an API), but we quickly realized that some of the translations were subpar and misleading, e.g. "ROT avdrag" -> "ROOT deduction" instead of the desired "ROT deduction". Switching to DeepL solved many of these translation mishaps. A short summary of the Jira ticket lengths is also displayed in Table 3.1, where we can see that the average ticket length was 236 tokens.

Table 3.1: Summary of Jira ticket lengths, expressed in number of tokens.

Avg.	Median	Min.	Max.
236	165	1	13,086

3.1.4 Preprocessing

We chose to perform a standardized preprocessing on each dataset. This included removing all newline characters and any spacing more than two spaces wide. We also stripped any whitespace from the beginning and end of the context. Because TechQA and SQuAD are inherently English datasets, no translation (like the one done on Fieldly) had to be done for those datasets.

3.2 Reader Evaluation

The first experiments were conducted on the reader component. This was done to both evaluate the performance of each model on the respective dataset and assess the practicality and inference characteristics of each language model. Due to the many different options of language models to choose from and limited hardware resources, an emphasis was put early on to narrow down the options by setting up constraints. The constraints set were:

1. The model should be hosted on the Hugging Face platform[1].
2. The model should have *reasonable* memory consumption.

Requirement 1: Hugging Face

Today, multiple platforms and interfaces exist from which language models can be accessed. There are models that are solely hosted in closed environments and accessed either through a UI or an API (e.g., OpenAI's GPT-4) or models hosted on open-source platforms (e.g., Hugging Face or Ollama). For this report, open-source models were preferred due to the limited transparency in the training and development processes of closed-source models, which could potentially hinder a clear understanding of the results produced by the experiments. Furthermore, many closed-source models operate under billing plans, which could quickly scale up in costs when performing large-scale experiments. Taking these factors into consideration, Hugging Face was chosen as the preferred platform to retrieve models from. Many models hosted on Hugging Face are open-source and free to use, making them compelling for academic purposes. Moreover, Hugging Face offers an easy-to-use programmatic interface. with many maintained adjacent python packages, and also an integration with *Langchain*, a popular framework for building LLM applications. Many models hosted on Hugging Face can also be downloaded and inferred locally, making it compelling to use from a privacy perspective, especially when working with confidential data.

Requirement 2: Memory Consumption

The second requirement was chosen for the sole purpose of practicality and hardware limitations. A large limitation on what model you can use lies in the size, or rather the number of parameters, of the model. In our case, we wanted the models to fit into the GPU memory (denoted VRAM), as this provides a time-effective and general way of speeding up inference.

[1] https://huggingface.co/models

A rough estimate of the memory consumption can be determined using $M = N * \frac{P}{8}$, where M is the memory consumed in bytes, N the number of model parameters, and P the precision of the weights in bits. Subsequently, a model with 1 billion parameters using 16-bit floating-point precision should thus consume roughly 2 GB of memory (when excluding non-accounted overhead). It is worth noting that the memory calculated is merely the model size and not the consumed memory during inference. From section 2.2.3 we observe how the space and time complexity of the attention mechanism strongly depends on the input sequence length n, and in the case of self-attention, results in a complexity of $O(n^2)$ (when considering the embedding dimension d, this becomes $O(n^2 d)$). This becomes evident for RAGs, which can receive multiple contexts as input (resulting in long input sequences). Consequently, with limited hardware and memory available for our experiments (see section 3.6), the condition "*reasonable* memory consumption" was therefore chosen to be interpreted as models with a maximum model size of roughly 4GB. With 4-bit quantization, this translates to a maximum parameter count of 8 billion parameters (when excluding additional memory overhead).

3.2.1 Choosing Reader Models

With the reader requirements now set, the next step involved choosing what models to examine. For this process, various criteria were considered when selecting models, including their performance, popularity, and practicality. The models chosen are presented in Table 3.2.

Table 3.2: The models that were chosen and their parameter counts.

Model	Nbr. of parameters	Hugging Face ID
Llama 3.1 8B Instruct	8 billion	meta-llama/Llama-3.1-8B-Instruct
Zephyr-7B-β	7 billion	HuggingFaceH4/zephyr-7b-beta
Phi-3-Mini-128k Instruct	3.8 billion	microsoft/Phi-3-mini-128k-instruct
Flan-T5 Large	738 million	google/flan-t5-large

Llama 3

Released in 2024, Llama 3 is among the newest and most well-known large language models. It is the latest iteration of the Meta Llama lineup and has quickly become a popular choice in LLM applications due to its general-purpose capabilities and wide range of model sizes to choose from. Due to its popularity and notable reputation [Adams et al., 2024, Zhang et al., 2024a, Ibrahim, 2024] we decided to assess the smallest model offered (with 8 billion parameters). It is also worth noting that the instruct version of the model was chosen (i.e., has been fine-tuned to follow instructions) due to the general characteristics of the tasks it would receive as a reader in a RAG.

Zephyr-7B

Zephyr-7B is another model that was chosen due to its popularity, but its performance and distinct training scheme also made it stand apart (see section 2.2.6). When it was released in 2023, it achieved state-of-the performance on the chat-based benchmark MT-bench among other 7B parameter models (Xwin-LM v0.1, MPT-Chat, etc.) and furthermore rivaling other models much larger than itself (i.e. Llama2-Chat 70B and WizardLM v1.0) [Tunstall et al., 2023]. Moreover, Zephyr was trained exclusively through language model distillation and alignment, eliminating the need for costly human annotation. This approach makes it particularly appealing from a training efficiency standpoint.

Phi-3

Phi-3 is another well-known language model which was released by Microsoft in 2024. It was selected mainly for its practicality and performance, as it was designed for inference on small devices while still delivering competitive performance when compared to much larger models [Abdin et al., 2024]. This is achieved by its distinct training scheme outlined in section 2.2.4, which focuses on high-quality data to maximize the reasoning capabilities of the model and not per say the factual knowledge base. This makes Phi-3 an appealing option as a reader due to the general characteristics of RAGs, which use context-driven augmentation to generate answers instead of its inherent knowledge store. Lastly, the selected version of Phi-3 supports very long sequence lengths (128,000 tokens), making it well-suited for handling large context-based prompts, a key advantage when working with RAGs.

Flan-T5

Lastly, Flan-T5 is a language model developed by Google and released in 2022. Although it has been available for a longer time and is relatively smaller than the other models, it was chosen for its impressive performance demonstrated in the previous thesis work by [Kapetanovic, Samil and Raja, Mohammad, 2024]. In the mentioned paper, Flan-T5 achieved superior performance on generative question-answering tasks compared to other models, which included BLOOMZ, Mistral 7B v0.1, and mT0. For that reason, Flan-T5 was chosen as a high benchmark for the other models to be compared with during the experimentation.

3.2.2 Experiments

The evaluation of each reader model was conducted in the following way:

1. Take a question-answer pair with its associated correct context.
2. Input the question and context into the reader model using a prompt and generate an answer.
3. Compare the generated answer with the reference answer and compute evaluation metrics.

4. Repeat for other question-answer pairs and the other dataset.

To minimize cost and memory consumption, all of the reader experiments (and end-to-end) were conducted using 4-bit quantization (with the *bitsandbytes* package). Moreover, since a small subset of the technotes in TechQA were significantly longer than the 95th percentile and memory consumption grow quadratically as a function of input length, we decided to truncate all contexts longer than 3,000 tokens. This was only done during reader evaluations and not end-to-end, mainly due to the large variations in inference time across the models. The truncation process was done by extracting a subsegment of the context (in total, 3,000 tokens) that included both the answer and equal portions of text to its left and right. The extraction was only done for TechQA as the contexts from SQuAD all fall under the 3,000 token limit. However, due to Flan-T5 having a maximum input sequence length of 512 tokens, a similar procedure was done for contexts in *both* datasets when using Flan-T5. This time, however, a much smaller segment (300 tokens) was extracted. The prompt that was used consisted of an instruction, question, and context. Depending on the model used, the final prompt differed due to different chat templates being offered by each model, but they were essentially the same. Below is the prompt used for Phi-3:

```
<|system|>
Answer the asked question using the provided contexts.
The answer should be concise and relevant to the contexts.<|end|>
<|user|>
Question: {question}.
Contexts: {context}<|end|>
<|assistant|>
```

Inference

In addition to the evaluation metrics mentioned, we also recorded the inference time for each model. This captures the usability of the models, as many real-world applications depend on fast inference for a good user experience. Additionally, we measured the model output length for each question to gain more insight into how comprehensive the answers were. By comparing the output lengths with reference, we also got a sense of how well the generated answers matched with the reference answer.

3.2.3 Evaluation

A range of metrics were used to evaluate the generated text. This includes Recall, Precision, F1, BERTScore, Exact Match, and BLEU. Recall, Precision, and F1 are defined in section 2.6.2, and in our case *a relevant instance* means a generated word that has a match in the reference answer. Subsequently, Recall and Precision are defined as:

$$Recall = \frac{|\,matching\ words\,|}{|\,reference\ words\,|} \quad \text{and} \quad Precision = \frac{|\,matching\ words\,|}{|\,predicted\ words\,|} \tag{3.1}$$

We used these bag-of-word metrics as they provide an overall view of how well the generated answer contains words found in the reference answer. Moreover, Recall and Precision complement each other well, as they both capture distinct aspects of what a good match

should be. For recall, the more reference words we manage to capture in our generated output, the better, while for precision, more emphasis is instead put on how well each predicted word matches the reference. However, a major drawback with the definitions of Recall and Precision is the need for exact matching on word-level. This is why we used BERTScore, which can handle synonyms and paraphrases with the same word-level, out-of-order matching as Recall and Precision.

We also used *Exact match*, denoted EM, which is defined as follows:

$$EM = \begin{cases} 1, & \text{reference} \subseteq \text{predicted}, \\ 0, & \text{reference} \nsubseteq \text{predicted} \end{cases} \tag{3.2}$$

Although not a reliable metric for long reference answers, especially together with generated predictions, it serves as a good metric for short reference answers (e.g. SQuAD). By checking if the reference answer is contained within the generated output, we get an overall view on the correctness of a generated answer and how well the reader finds the answer from a context. Lastly, we also used BLEU as a complement to the bag-of-words approaches mentioned above. By conducting n-gram matching, BLEU provides an option to capture the token order, thus providing a complementary evaluation aspect that the other metrics lack.

Reference

We also chose to include results presented by [Rajpurkar et al., 2016] on SQuAD, denoted *Logistic regression* and *Human*, to serve as a reference for our results. In the case of *Logistic regression*, a model was trained to correctly classify the correct answer span from a set of available candidate spans within a passage, while *Human* refers to human performance on the same task. Only F1 and EM scores were provided by the authors, and in this case, EM was defined as an identical match, differing from our own EM definition (see 3.2). It is worth noting that we did not find any similar and relevant reference results for TechQA, as it is a less-used dataset.

3.3 Retriever Evaluation

The next step involved evaluating the retriever component and some enhanced retrieval methods. Due to many embedding models being considerably smaller than readers, there was a larger selection of models to choose from. The same requirements for the reader were also used when selecting dense retrievers. This includes the requirement that the dense retriever should be hosted on the Hugging Face platform and moreover, should only consume a reasonable amount of memory. Even though many embedding models are smaller than reader models, there still exists some large embedding models with several billions of parameters (e.g. Nvidia's *NV-Embed-v2* with 7 billion parameters) which can consume a substantial amount of memory.

3.3.1 Choosing Retrievers

When choosing retrievers, different factors were looked upon such as benchmark performance, popularity, and relevance. For dense retrievers, the MTEB leaderboard [HuggingFace, 2024] was used to find high-performing retrievers, while for sparse retrievers, the two main encoding methods (TF-IDF and BM25) were explored. The chosen retrievers are presented in Table 3.4. The first retriever chosen was GTE v1.5, a versatile embedding model with reasonable size and developed with general purpose intent. It achieved very high performance on the MTEB benchmark when compared to other models with similar sizes. Compared to GTE v1.5, Jina v3 is a much larger embedding model (~4x), but offered better performance on MTEB. On the semantic textual similarity (STS) category, it achieved the second highest results on the leaderboard (Nov. 2024), while competing with the much larger models offered by OpenAI and Cohere on all MTEB categories.

Table 3.4: Retrievers used in our experiments.

Retriever	Nbr. of params.	Dim.	Hugging Face ID	Max. tokens
GTE v1.5	137 million	768	Alibaba-NLP/gte-base-en-v1.5	8192
Jina v3	572 million	1024	jinaai/jina-embeddings-v3	8192
Nomic v1.5	137 million	768	nomic-ai/nomic-embed-text-v1.5	8192
TF-IDF	~|vocab.|	|vocab.|	-	-
BM25	~|vocab.|	|vocab.|	-	-

Although Jina v3's strong performance on MTEB was likely influenced by the use of LoRA adapters, a strong baseline model still remains essential for achieving good results. Considering that the other retrievers did not have any LoRA adapters, and in order to make the evaluations more equal, we chose not to use the Jina v3 LoRA adapters during our experiments. The third dense model used in our experiments was Nomic v1.5, a model with the same size as GTE v1.5, and developed primarily with long context support in mind. Due to TechQA having longer contexts and questions than many other datasets, we decided to include Nomic v1.5 to assess its performance on that dataset. It should be noted that during our experiments we noticed how Nomic performed quite well on SQuAD compared to the other dense retrievers. A closer look at the technical report [Nussbaum et al., 2024] revealed a possible explanation for this, as the authors state that SQuAD constituted a small percentage (< 0.01%) of the total training mix used. Finally, TF-IDF and BM25 were chosen as our sparse retrievers, mainly due to their popularity and accessibility, as many implementations already exist in various packages and integrations (e.g., LangChain, Scikit-learn).

3.3.2 Experiments

The process of evaluating the retriever consisted of four steps:

1. Index all contexts into a vector database using a specified dense embedding model *or* in the case of sparse retrieval, encode the whole collection of documents at once.

2. Perform a search for a specific question, using encoding and cosine similarity, and save the top k documents retrieved.

3. Compare the retrieved k documents with the correct associated context for the question and compute Recall@k.

4. Iterate for all questions and retrievers.

First, all documents were indexed using an embedding model and stored in a chosen vector store. We used the LangChain implementation of a FAISS vector store together with an *IndexFlatIP* index. This enabled us to do brute-force matching using cosine similarity as scoring function. Indexation was performed on all documents in the SQuAD training dataset and the TechQA development set. This resulted in a total of 19,029 documents for SQuAD and 28,507 documents for TechQA (see Figure 3.5). However, before indexation, basic pre-processing was done (described in section 3.1.4) on passages from both datasets. For the sparse retrievers more heavy pre-processing was performed, such as lower-casing and only keeping alphanumeric characters. Furthermore, to handle the maximum input sequence limit of 8192 tokens, all documents longer than the specified token limit were split into sub-pieces using a greedy approach.

Table 3.5: Nbr. of indexed docs for each dataset.

Dataset	Nbr. of docs.
TechQA	28,507
SQuAD	19,029

Following the indexation, a search was conducted for each question in both datasets. The question was first encoded using the same embedding method as the indexation, and pairwise cosine similarity was then performed for every context to find the top k matching documents. The value of k was varied between 1 and 50 and Recall@k (see section 2.6.1) was subsequently calculated for each value of k.

Reference

Similarly to the reader evaluations, we also present retrieval results from similar studies on both TechQA and SQuAD as references. DPR_{TQA} is a retriever presented in [Yu et al., 2021] that uses a standard pre-trained, non-finetuned BERT bi-encoder as a dense retriever. The retriever was evaluated on TechQA and Recall@1 and Recall@5 were subsequently reported. We also introduce *RAG-end2end* [Siriwardhana et al., 2023], a model which similarly also utilizes a BERT bi-encoder as retriever. However, compared to DPR_{TQA}, *RAG-end2end* was finetuned specifically on the SQuAD dataset to evaluate the performance gains from such method. From these evaluations, the authors present Recall@5 on retrieval tasks performed on SQuAD.

3.3.3 Enhanced Retrieval Methods

To enhance recall performance in retrieval, more advanced multi-stage methods can be employed beyond simple cosine similarity. These methods are often additional steps, built on top of the vanilla solution, to either refine the results or improve the likelihood that we match with the correct document during the search. The methods we choose to evaluate in this report are:

1. Query expansion.
2. Hybrid retrieval.
3. Reranking.

Query Expansion

Query expansion is a collective name for methods that involve rewriting the query or reformulating it to increase the likelihood of matching with the correct document [Jagerman et al., 2023]. This can be done by either reformulating the entire query, adding keywords to the search string, or appending a hypothetical answer to the question and then perform a similarity search on the resulting string. The agent that is used to perform the expansion is often a language model. For our experiments, we chose to append a hypothetical answer to the search string by generating an answer using a language model. We choose to use Flan-T5-Large as the agent, due to its small size and faster inference. The prompt used was:

```
Give an example answer to following question.
Question:  {question}
```

Hybrid Retrieval

Another approach for improving retrieval performance is to combine multiple retrievers, often dense and sparse, to leverage the strengths of each method. As described in sections 2.3.1 and 2.3.2, sparse retrieval methods excel at keyword matching while dense retrievers are good at matching semantically similar sentences. By combining both retrievers, both aspects can be taken into consideration resulting in an overall recall improvement. When combining ranked retrieval results from multiple retrievers, there is a need to combine the rankings into a new consolidated and unified ranking. One way to achieve this is to use *Reciprocal Rank Fusion* (RRF). RRF is defined as:

$$RRF_{\text{score}}(d \in D) = \sum_{r \in R} \frac{1}{k + r(d)} \tag{3.3}$$

where D is the set of all retrieved documents, k a constant, R all the retrieval methods used, and $r(d)$ the rank of document d determined by the retriever r. For our experiments, we choose to combine the best-performing dense retriever with the best sparse retriever to assess the impact hybrid retrieval poses on performance.

Reranking

The final approach which can be used to improve retrieval is reranking. This method works by employing a reranker, often transformer-based, as a second stage retrieval step, using a cross-encoder style setup [Moreira et al., 2024]. This involves providing a query and a passage as input to the reranker, which then outputs a similarity score (see Figure 3.2).

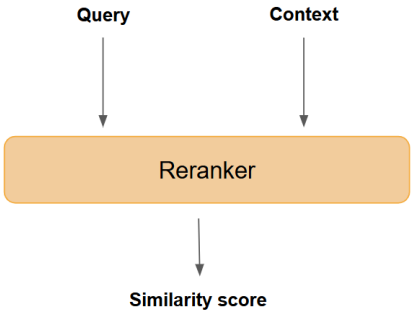


Figure 3.2: A cross-encoder style reranker.

By doing cross-encoder style comparisons instead of bi-encoder-based computations (vanilla retrievers), we can capture deeper semantic meanings between the query and context and hopefully return a more accurate similarity score. Reranking is usually done as a second step due to the increased costs associated with computing scores for each context, especially for a large indexed corpus. In Figure 3.3, this is illustrated, where we have a retriever with $k = 6$ and then a reranking step that finds the top 3 most similar documents according to the reranker. For our experiments, we chose **Jina Reranker v2** (Hugging Face id: jinaai/jina-reranker-v2-base-multilingual) as reranker.

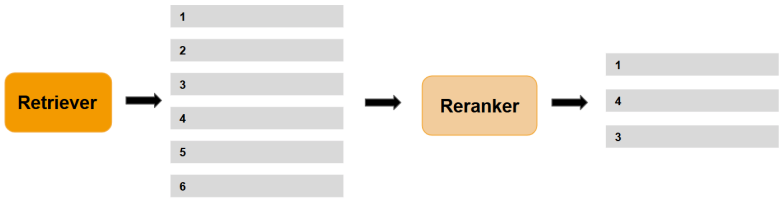


Figure 3.3: Illustration of the reranker as a second step.

3.4 End-to-End Evaluation

The final experiments were conducted using an end-to-end (E2E) setup. The experimental workflow can be described as follows:

1. Perform retrieval of k most relevant documents from a question-answer pair.
2. Insert the k most relevant documents together with the question into the reader.
3. Compute comparison metrics (such as F1 and BERTScore) for the generated answer and reference answer.
4. Iterate over all question-answer pairs and datasets.

When performing E2E testing, we chose to use setups consisting of high-performing components. After reviewing our results, which we will further disclose in our Results section, we decided to go with Phi-3 as the reader. This model exhibited good performance on both TechQA and SQuAD while having satisfactory inference times compared to the other models. Phi-3 was also less prone to generate excessively long or overly short answers, unlike some of our other models (i.e. Zephyr and Flan-T5). On the retriever side, we ended up using Jina v3 as the dense retriever due to its strong performance on both datasets for all k. We also chose to incorporate hybrid retrieval (with BM25) and reranking as these methods showed promising improvements to overall retrieval performance (will be further disclosed under Results). The chosen setups are shown in Table 3.6.

Table 3.6: E2E setups and their alias.

Alias	Retriever	Reader
Base	Jina v3	Phi-3
Hybrid	Jina v3 + BM25	Phi-3
Reranker	Jina v3 + Jina Reranker v2	Phi-3

The evaluation metrics used were the same as those applied when evaluating the reader component (i.e. F1, BERTScore, and others). However, in this case, the contexts provided to the model were retrieved by the retriever and, thus, not necessarily the correct one.

3.5 Fieldly

When building the Fieldly RAG prototype, we wanted to use the best configuration found from our evaluations on TechQA. Similarly to E2E-testing, we decided to use Phi-3 as reader and Jina v3 as dense retriever. However, to confirm that Jina v3 was indeed the best retriever, Nomic v1.5 was also used as an optional configuration on the RAG. Moreover, as the E2E evaluations did not produce any decisive or conclusive results, we decided to integrate hybrid retrieval (with BM25), normal dense retrieval, and reranking as configurable options for the RAG. This was done to assess the impact these methods have on a RAG from a human

user perspective. On the data side, to reduce complexity, each Jira ticket (see section 3.1.3) was indexed in its entirety, meaning that a few of the tickets were truncated to match the maximum sequence length (8192 tokens) of the embedding models used. Furthermore, to limit memory consumption, contexts from the much longer self-service documents were split into 100-token long sub-documents, with a 20-token overlap between each sub-document. Finally, we performed a qualitative assessment on the following configurations shown in Table 3.8.

Table 3.8: Fieldly setups.

Retriever	Reader
Nomic v1.5	Phi-3
Jina v3	Phi-3
Jina v3 + BM25	Phi-3
Jina v3 + Jina Reranker v2	Phi-3

The qualitative assessment was conducted by creating a small subset of questions on the Fieldly data. These questions were based on a mix of both specific and conceptual-type questions and were generated by manually looking at contexts beforehand. In total, up to 10 questions were used. Moreover, the retriever was set to only retrieve the top 3 most similar contexts. During the evaluation, we mainly checked the correctness of the answers, but we also directly looked at the retrieved contexts to see how relevant they were to the questions asked. In some cases, the data had multiple contexts that could be used to answer the question. However, some contained more relevant information than others. As a result, some answers, while correct, also included irrelevant information when relating to the question.

On top of the RAG back-end, a front-end UI was built for easy interaction with the application. This was done using *Flask*, a popular Python web application framework for building smaller applications. For the back-end, both LangChain and standard Hugging Face packages were used, as well as FAISS for vector storage.

3.6 Hardware and Software

The hardware used in our experiments was determined by the workload required for each experiment. Moreover, many of the experiments were conducted on Google Colab, which forced us to take cost into consideration. This is reflected by the large variety of GPUs used in our experiments.

Reader evaluation	NVIDIA T4 Tensor Core GPU.
Retriever evaluation	NVIDIA GeForce RTX 3050 Ti Laptop GPU
End-to-End (SQuAD and TechQA ($k = 1$))	NVIDIA L4 Tensor Core GPU
End-to-End (TechQA ($k = 3, 5$))	NVIDIA A100 Tensor Core GPU

Multiple packages were also used during our experiments and the most significant ones are displayed below.

Package/Data	**Version**
sentence-transformers	3.1.0
langchain	0.3.0
langchain-huggingface	0.1.0
langchain-community	0.3.0
transformers	4.44.2
torch	2.4.1
faiss-cpu	1.8.0.post1
bitsandbytes	0.44.1
SQuAD	Hugging Face ID: rajpurkar/squad
TechQA	Hugging Face ID: PrimeQA/TechQA

Chapter 4
Results

4.1 Reader Performance

The performance of our four reader models, evaluated on TechQA, are presented in Table 4.1. From the results we can distinguish large variations in ranking depending on which metric is used. For example, Zephyr had the best Recall score of all models while Flan-T5 had the best Precision. For the F1 score, Llama 3 was the best-performing model, closely followed by Phi-3. The opposite was true for BERTScore and BLEU, where Phi-3 was the better-performing model, with Llama 3 closely trailing.

Table 4.1: Reader performance on TechQA. Highest score marked in bold, while second highest with underline.

	TechQA					
Model	**Recall**	**Precision**	**F1**	**EM**	**BERTScore**	**BLEU**
Llama-3	<u>0.485</u>	0.265	**0.306**	<u>0.015</u>	<u>0.858</u>	<u>0.130</u>
Zephyr	**0.569**	0.210	0.287	<u>0.015</u>	0.852	0.114
Flan-T5	0.207	**0.382**	0.202	**0.061**	0.838	0.106
Phi-3	0.377	<u>0.302</u>	<u>0.302</u>	0.007	**0.865**	**0.165**

In Table 4.2 the reader results are presented for the SQuAD dataset. From the table, Flan-T5 can be observed to have the best-performing model of the four by a considerable margin. Here, the largest difference between Flan-T5 and the other models can be observed in Precision and BLEU scores, where the gap to the second-best performing model (Phi-3) is more than 60 percentage points. Excluding Flan-T5, the results indicate that Phi-3 had the best overall performance of the remaining models on SQuAD. Furthermore, when comparing our results with *Logistic regression* and *Human*, we note how these models/methodologies outperform all of our reader models, except Flan-T5, on F1. The EM scores for our reader

models do surpass the same score for *Logistic regression*, but this is most likely attributed to the different definition of EM used by the authors of [Rajpurkar et al., 2016], which is more restrictive than the one we used.

Table 4.2: Reader performance on SQuAD, including reference models. The best performance is marked in bold, while the second highest with underline.

	SQuAD					
Model	**Recall**	**Precision**	**F1**	**EM**	**BERTScore**	**BLEU**
Llama-3	0.543	0.166	0.229	0.702	0.872	0.080
Zephyr	<u>0.689</u>	0.055	0.095	<u>0.730</u>	0.830	0.016
Flan-T5	**0.862**	**0.878**	**0.849**	**0.773**	**0.969**	**0.676**
Phi-3	0.646	<u>0.242</u>	<u>0.307</u>	0.696	<u>0.876</u>	<u>0.093</u>
Logistic regression	-	-	0.510	0.400[1]	-	-
Human	-	-	0.905	0.803[2]	-	-

4.1.1 Inference

In Figure 4.1, the inference times are presented for each model on both datasets.

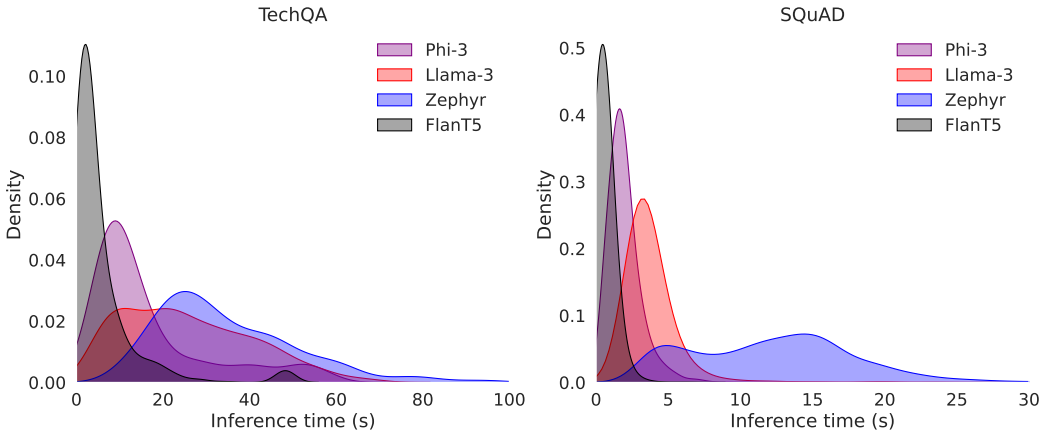


Figure 4.1: Distribution plot of inference time for our four generative models

From the plots, we observe how Flan-T5 overall has the lowest inference times of all four models, followed by Phi-3, Llama 3, and lastly, Zephyr. We can also see that SQuAD in general produces faster inference times than TechQA and that the difference between the fastest

[1] This score is based on SQuAD EM (requiring identical match), differing from our own EM definition (see section 3.2.3).

[2] Same as above.

model (Flan-T5) and slowest (Zephyr) is in some cases more than 10x.

Table 4.3: Summary of the average inference times in seconds (< 95th percentile)

Model	Avg. (TechQA)	Avg. (SQuAD)
Llama-3	23.8	3.4
Zephyr	32.3	11.5
Flan-T5	3.9	0.5
Phi-3	14.7	1.8

In Figures 4.2 and 4.3, distribution plots of the output lengths for each model are instead displayed for TechQA and SQuAD, respectively (together with their reference answers). In Figure 4.2 we see how Phi-3 produces answers with similar lengths compared to the TechQA reference. Both Llama 3 and Zephyr also produce, on average, longer answers than reference, while Flan-T5 produces much shorter answers than reference.

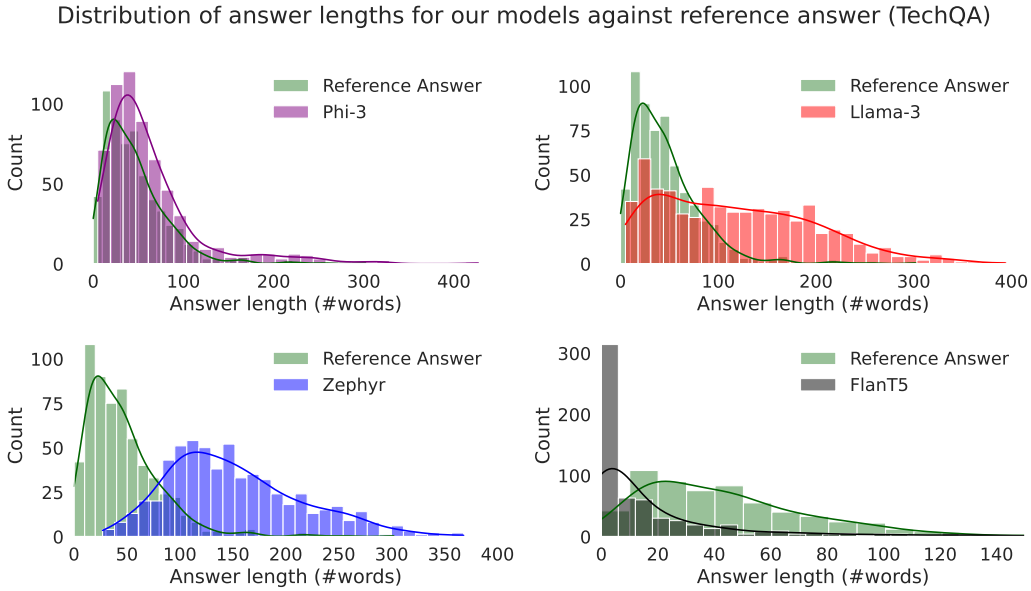


Figure 4.2: Distribution of answer lengths for our models versus reference answers, evaluated on TechQA.

In Figure 4.3 we can outline how SQuAD has much shorter expected reference answers than TechQA (< 5 words versus 20-30 words for TechQA). We also note how the Flan-T5 outputs match quite well with SQuAD reference answers, with an almost perfect match in distribution. Furthermore, the remaining models seem to produce answers that are longer than their reference. This is particularly true for Zephyr which in some cases produces answers 10x - 20x longer than reference for SquAD. In the case of Phi-3 and Llama 3, most of the outputs vary between 5-20 words, which is longer than reference but considerably shorter than Zephyr, which in some cases produces answers up to 100 words long.

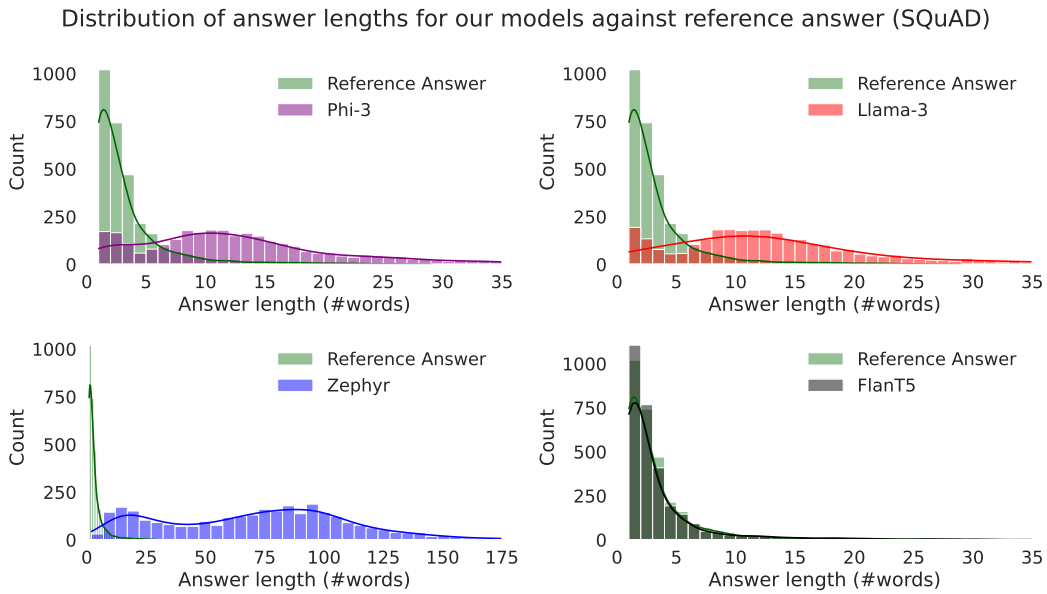


Figure 4.3: Distribution of answer lengths for our models versus reference answers, evaluated on SQuAD.

4.1.2 Model Output

In Table 4.4 we can observe an example question-answer pair from TechQA and the model outputs for that question. We note how the model output varies quite a bit for each model. For example, both Zephyr and Llama seem to produce the longest answers of the four, while Flan-T5 produces the shortest. Comparing the reference answer with the outputs also highlights how some of the models (Zephyr and Llama) generate more information than "required". In this case, the models list which packages that should be installed as well. In the same way, we also note how models can produce answers not on par with reference, which in this case would be Flan-T5. From the example output, Flan-T5 seems to produce only a partial and incomplete answer when comparing to reference.

Table 4.4: Model outputs for example questions extracted from TechQA.

TechQA	
Question: Trying to install DASH 3.1.3 on AIX but IIM won't start and instead creates a dump file with a GPF noted. We found a technical note with a list of the required libs for IIM on AIX and we have all of the latest versions of those libs. Why is the installer failing to start? **Answer**: To use the GUI, the gtk+ RPM and dependencies must be available on AIX. Install this set of RPMs on AIX:	
Model	**Output**
Phi-3	The installer is failing to start because the required gtk libraries are either not installed or not at the required versions on AIX. Ensure that all the latest versions of the listed libraries are installed and that the LIBPATH environment variable points to the installed library paths.
Llama-3	The issue is that the gtk libraries are not installed or not at the required versions on the AIX system. The error message indicates that the required libraries are not found in the java.library.path. To resolve this issue, you need to install the required gtk libraries on the AIX system. The recommended RPMs are: atk-1.12.3-2.aix5.2.ppc.rpm cairo-1.8.8-1.aix5.2.ppc.rpm expat-2.0.1-1.aix5.2.ppc.rpm fontconfig-2.4.2-1.aix5.2.ppc.rpm freetype2-2.3.9-1.aix5.2.ppc.rpm gettext-0.10.40-6.aix5.1.ppc.rpm glib2-2.12.4-2.aix5.2.ppc.rpm gtk2-2.10.6-4.aix5.2.ppc.rpm libjpeg-6b-6.aix5.1.ppc.rpm libpng-1.2.32-2.aix5.2.ppc.rpm libtiff-3.8.2-1.aix5.2.ppc.rpm pango-1.14.5-4.aix5.2.ppc.rpm pixman-0.12.0-3.aix5.2.ppc.rpm xcursor-1.1.7-3.aix5.2.ppc.rpm xft-2.1.6-5.aix5.1.ppc.rpm xrender-0.9.1-3.aix5.2.ppc.rpm zlib-1.2.3-3.aix5.1.ppc.rpm You can download the gtk2_bundle_v2.tar package containing all the RPMs and the install script from the IBM FTP site: ftp://ftp.software.ibm.com/aix/freeSoftware/aixtoolbox/ezinstall/ppc/ Follow the instructions in the technical note to install the required RPMs and ensure that the environment variable LIBPATH contains a path to the installed libraries.
Zephyr	To resolve the segmentation error caused by the missing gtk libraries on AIX, follow these steps: 1. Download the gtk2_bundle_v2.tar package from ftp://ftp.software.ibm.com/aix/freeSoftware/aixtoolbox/ezinstall/ppc/ and decompress it to a new directory. 2. Open the decompressed directory and run the install.sh script. This script will check if the required RPMs are installed at the compatible level. If they are not, it will prompt to install the compatible versions. 3. Verify that the environment variable LIBPATH contains a path to the installed libraries. For example, LIBPATH=/opt/freeware/lib/. The installed RPMs must be of the minimum versions listed above. The required RPMs are: atk-1.12.3-2.aix5.2.ppc.rpm cairo-1.8.8-1.aix5.2.ppc.rpm expat-2.0.1-1.aix5.2.ppc.rpm fontconfig-2.4.2-1.aix5.2.ppc.rpm freetype2-2.3.9-1.aix5.2.ppc.rpm gettext-0.10.40-6.aix5.1.ppc.rpm glib2-2.12.4-2.aix5.2.ppc.rpm gtk2-2.10.6-4.aix5.2.ppc.rpm libjpeg-6b-6.aix5.1.ppc.rpm libpng-1.2.32-2.aix5.2.ppc.rpm libtiff-3.8.2-1.aix5.2.ppc.rpm pango-1.14.5-4.aix5.2.ppc.rpm pixman-0.12.0-3.
Flan-T5	Install the following RPMs on AIX:

In Table 4.4, the model output from the SQuAD dataset is presented. Similarly to TechQA, Flan-T5 seems to produce the shortest answers to both questions, while Zephyr produces the longest. We also note how Flan-T5 seems to generate very similar answers to reference (in this case identical) and that the reference answers are concise and short by nature.

Figure 4.4: Model outputs on example questions extracted from SQuAD.

SQuAD		
Question/Answer	**Model**	**Output**
Question: When did Beyonce start becoming popular? **Answer**: late 1990s	Phi-3	Beyoncé started becoming popular in the **late 1990s** as the lead singer of the R&B girl-group Destiny's Child.
	Llama-3	According to the context, Beyoncé started becoming popular in the **late 1990s** as the lead singer of Destiny's Child, and her solo debut album "Dangerously in Love" in 2003 further established her as a solo artist worldwide.
	Zephyr	Beyoncé's solo career began in 2003 with the release of her debut album, Dangerously in Love. However, she had already achieved fame as the lead singer of Destiny's Child, a successful R&B girl group, in the **late 1990s**. So, while Beyoncé's solo popularity began in 2003, her overall fame and success as a musician started earlier with Destiny's Child.
	Flan-T5	**late 1990s**
Question: What are the most common foods in Estonia? **Answer**: black bread, pork, potatoes, and dairy products	Phi-3	**black bread, pork, potatoes, and dairy products** are the most common foods in Estonia.
	Llama-3	The most common foods in Estonia are: 1. **Black bread** 2. **Pork** 3. **Potatoes** 4. **Dairy products**
	Zephyr	The most common foods in Estonia are **black bread**, **pork**, **potatoes**, and **dairy products**, with fresh berries, herbs, and vegetables also popular during the summer and spring seasons. While international cuisine has influenced Estonian food, hunting and fishing remain popular hobbies, but are not as commonly consumed as they once were. Grilling outside is also a popular summer activity.
	Flan-T5	**black bread, pork, potatoes, and dairy products**

4.2 Retriever Performance

In Table 4.5, the Recall@k scores for $k = [1, 5, 10]$ are presented for each model on both datasets. From the results we note how Jina v3 has the best performance on TechQA (for all reported k), while Nomic v1.5 is the runner-up. However, the results differ for SQuAD, where we instead have Nomic with the best performance for $k = [5, 10]$, while BM25 has the highest Recall@1 score and second-best performance for the rest of k. Compared with our reference retrievers, we observe how all of our dense retrievers are able to surpass *RAG-end2end* on SQuAD, although the margin to GTE v1.5 is not as considerable. For DPR_{TQA}, we observe impressive performance compared to our sparse retrievers, showcasing that the non-finetuned BERT embeddings perform quite well compared to our sparse algorithms. Although not on par with our other dense retrievers, DPR_{TQA} still manage to obtain good Recall@k scores.

Table 4.5: Recall scores for our retrievers on both TechQA and SQuAD. The highest score is marked in bold, while the second highest is underlined.

	TechQA			SQuAD		
Retriever	**Recall@1**	**Recall@5**	**Recall@10**	**Recall@1**	**Recall@5**	**Recall@10**
GTE v1.5	0.373	0.572	0.643	0.572	0.792	0.850
Jina v3	**0.407**	**0.625**	**0.697**	0.592	0.808	0.870
Nomic v1.5	<u>0.400</u>	<u>0.608</u>	<u>0.667</u>	<u>0.658</u>	**0.845**	**0.890**
TF-IDF	0.238	0.397	0.502	0.457	0.692	0.780
BM25	0.372	0.520	0.564	**0.666**	<u>0.833</u>	<u>0.879</u>
DPR_{TQA}	0.263	0.475	-	-	-	-
RAG-end2end	-	-	-	-	0.758	-

These results in Table 4.5 can be further expanded upon by looking at every integer k in the range [1, 50]. This was done and the results are presented in Figures 4.5 and 4.6. In Figure 4.5, Recall@k for TechQA is presented for dense and sparse retrievers, respectively. Of the dense retrievers, we see how Jina has an overall better performance for all k, while GTE has the worst score for low k but increases for larger k until it reaches parity with Jina for $k >= 40$. Looking at the sparse retrievers, BM25 performs better than TF-IDF for all k, but worse than Jina.

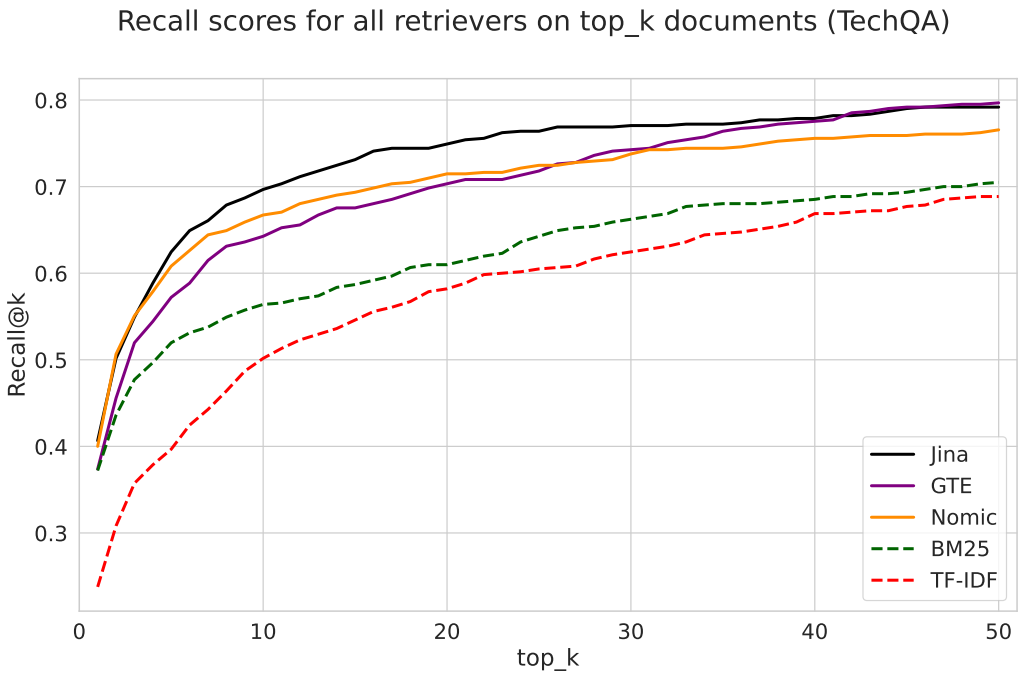


Figure 4.5: Recall@k for our retrievers on different k (TechQA).

In Figure 4.6 the same scores are presented, but for the SQuAD dataset. Here, Nomic seems to have the best performance of the dense retrievers while GTE has the worst. We also note that BM25 performs quite well on SQuAD, rivaling our dense retrievers and outperforming GTE. Moreover, similarly to TechQA, TF-IDF also exhibits the worst performance of our evaluated retrievers.

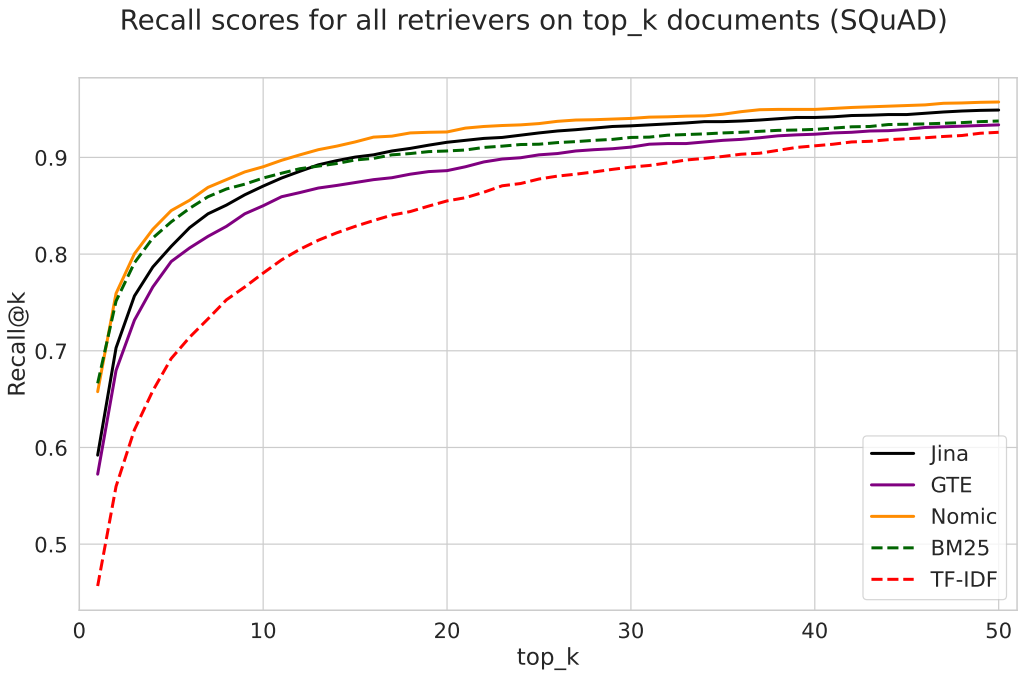


Figure 4.6: Recall@k for different values on k for SQuAD.

4.2.1 Hybrid Retrieval

In Figure 4.7 retrieval scores for BM25 and Jina are presented together with the results from their combined retriever (denoted Hybrid) for both TechQA and SquAD. We observe how Hybrid performs better than their individual components on SQuAD and partially better on TechQA (for $k >= 30$ and very small k). The recall improvement on SQuAD is noticeable, with an almost 10 percentage points improvement for low k.

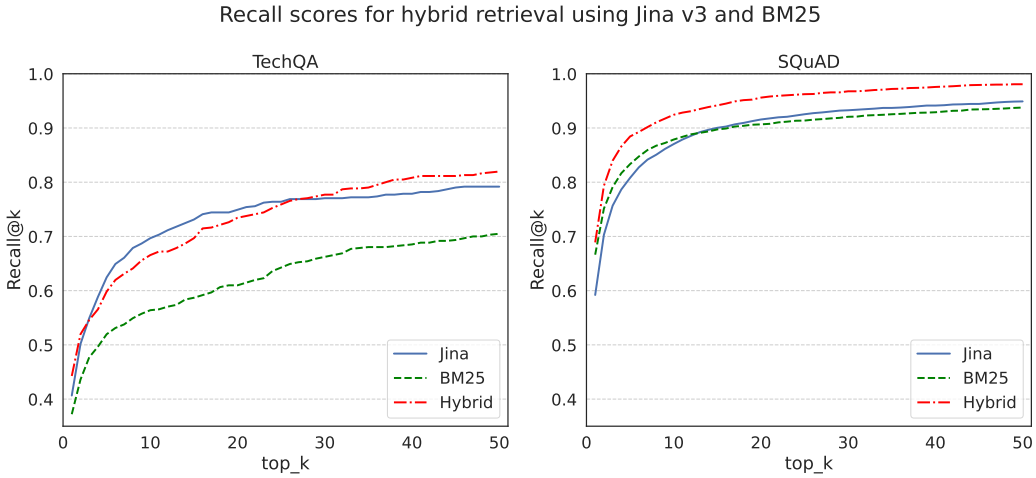


Figure 4.7: Recall@k for hybrid retrieval method using Jina and BM25.

4.2.2 Query Expansion

Query expansion is another way to increase recall and mitigate the effects of user bias on the input to the RAG. In Figure 4.8, the Recall@k scores for the Jina retriever and its query-expanded counterpart are presented. We can clearly observe how the recall doesn't improve in both datasets and, on the contrary, is worse than vanilla Jina on SQuAD. For TechQA, the recall difference between both methods is close to negligible.

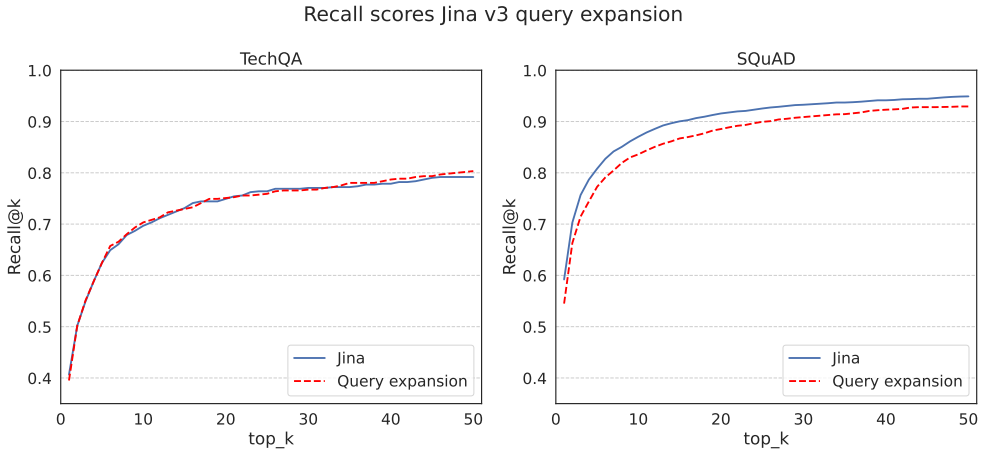


Figure 4.8: Query expansion using Jina embedding versus baseline.

4.2.3 Reranker

To improve recall, a reranker can be added as a second-stage retriever, providing a smaller, reshuffled, and more accurate subset of the first-stage retrieval results. In Figure 4.9 the recall scores for vanilla Jina and Jina + reranker are presented for both datasets. We observe how the reranker improves recall for both datasets, where the largest difference can be seen in SQuAD. Here an improvement of more than 15 percentage points can be seen when $k = 1$. For TechQA, the difference is smaller for low k but becomes more noticeable when $k >= 30$.

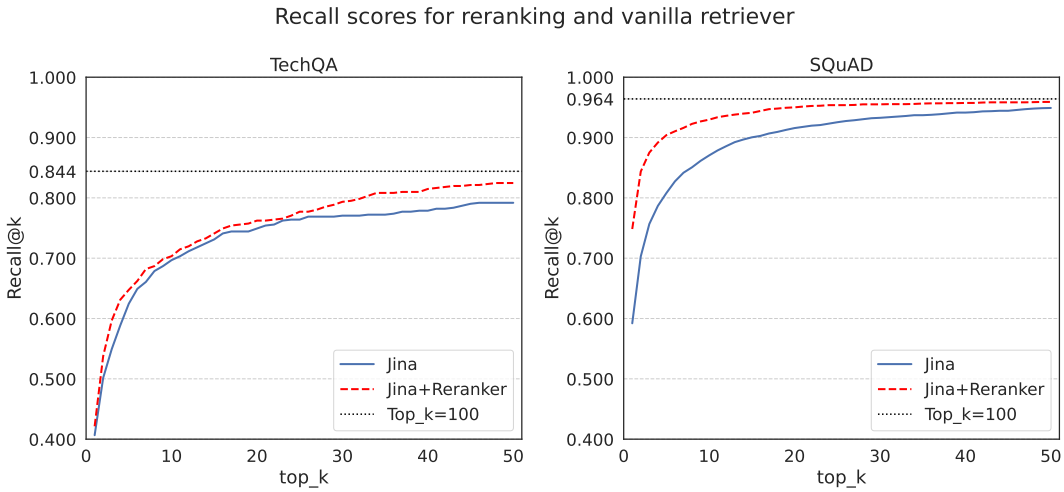


Figure 4.9: Recall scores for Jina retriever with and without reranker. The reranker was used on $\text{top}_k = 100$ retrieved documents from Jina.

4.3 End-to-End Testing

For E2E, the results were evaluated using the same metrics as for the reader, but in this case, the inputs were determined by the retriever. Three values on k were examined, $k = [1, 3, 5]$ and the results from TechQA are presented in Table 4.6. From the table, we observe how the results varied a lot depending on which metric was used. One thing that can be noted is how all scores (except recall) slightly decrease when k increases. Similarly, the E2E results for SQuAD are presented in Table 4.7. For SQuAD, Hybrid seems to have performed the best for $k = 3$ and $k = 5$, while reranking was the better performer for $k = 1$. The scores for SQuAD can also be seen as more coherent than TechQA, as the scores for each metric are more aligned with each other.

Table 4.6: End-to-End results for TechQA.

		TechQA					
top_k	**Setup**	**Recall**	**Precision**	**F1**	**EM**	**BERTScore**	**BLEU**
k=1	Base	0.275	**0.233**	0.225	0.0049	**0.849**	**0.119**
	Hybrid	**0.287**	0.228	**0.227**	**0.007**	0.848	0.114
	Reranker	0.282	0.230	0.225	0.005	0.848	0.082
k=3	Base	0.310	0.202	0.214	0.007	**0.841**	**0.080**
	Hybrid	**0.328**	0.193	0.212	**0.008**	0.839	0.079
	Reranker	0.315	**0.203**	**0.215**	0.005	**0.841**	**0.080**
k=5	Base	**0.314**	0.193	0.206	**0.008**	**0.841**	0.074
	Hybrid	0.312	**0.199**	**0.214**	0.005	0.840	**0.086**
	Reranker	0.313	0.190	0.206	**0.008**	0.838	0.082

Table 4.7: End-to-End results for SQuAD.

		SQuAD					
top_k	**Setup**	**Recall**	**Precision**	**F1**	**EM**	**BERTScore**	**BLEU**
k=1	Base	0.475	0.211	0.255	0.453	0.867	0.063
	Hybrid	0.520	0.237	0.284	0.509	0.873	0.073
	Reranker	**0.529**	**0.241**	**0.290**	**0.521**	**0.874**	**0.076**
k=3	Base	0.529	0.225	0.274	0.522	0.869	0.063
	Hybrid	**0.573**	**0.247**	**0.299**	**0.577**	**0.874**	**0.074**
	Reranker	0.552	0.239	0.288	0.546	0.872	0.068
k=5	Base	0.535	0.218	0.266	0.541	0.867	0.060
	Hybrid	**0.573**	0.219	**0.299**	**0.591**	**0.872**	**0.068**
	Reranker	0.550	**0.226**	0.276	0.558	0.869	0.063

4.4 Fieldly

In this section, we will present the results from our qualitative evaluations on the Fieldly RAG configurations (see section 3.5). First, an observation we made early on was that Nomic v1.5 performed considerably worse than Jina v3 embeddings. This manifested itself in Nomic struggling to retrieve relevant contexts and frequently retrieved passages irrelevant to the question. When evaluating the vanilla Jina v3 retriever, we did not experience the same issues, on the contrary, the recall actually improved significantly. Second, we noticed that reranking did not perform as well as expected, compared to normal dense retrieval, and that in many cases the top 3 retrieved contexts from the reranker coincided with the vanilla dense retriever (but with a shuffled order). Finally, we noticed that hybrid retrieval worked quite well in the sense that it was able to retrieve context that the vanilla dense retriever and reranker could not. Furthermore, a general observation made during these evaluations was that Phi-3 exhibited impressive capability to comprehend multiple contexts simultaneously and based on that, generate a coherent answer. This is best exemplified in Table 4.8, where example questions, contexts, and generated answers are displayed from our evaluations.

Table 4.8: Pseudonymized examples of questions, the retrieved passages, and generated answers from our RAG.

Question:
Who is the chief revenue officer?
Retrieved context:
`the customer success team at fieldly <name 1> chief revenue officer <phone number 1> <email 1> <name 2> head of customer success <phone number 2> <email 2> <name 3> customer success operations manager & interim team lead <phone number 3> <email 3>`
Generated answer:
`<name 1> is the Chief Revenue Officer at Fieldly.`
Question:
Does Fieldy support <service>?
Retrieved context:
`{resolution : Work has been completed on this issue. , created : <time> , description : <service> integration to complete the work flow between Fieldly and <service> , summary : <service> integration , display name : <name>, project name : <project> , ticket key : <key>}`
Generated answer:
`Yes, Fieldly supports <service> integration as indicated in the context where a resolution was completed to complete the work flow between Fieldly and <service>.`

In the first example, our retriever had to retrieve only one relevant context (the one displayed), as there was no other context containing the answer. Initially, using only a dense retriever, we were unable to retrieve this passage, and changing to a reranker did not help at all in this case. However, after incorporating a hybrid retriever using BM25, we successfully retrieved the relevant passage through keyword matching with the term "chief revenue officer". In the second example, to answer the question, the retriever had to either retrieve the

displayed Jira ticket or any passages containing information about the <service> integration. In this case, both the dense retriever and hybrid retriever were able to retrieve the relevant Jira ticket. Furthermore, as previously stated, the Phi-3 reader model showed impressive performance in understanding the retrieved contexts and the generation of a correct answer.

A front-end UI was also created for an easy interaction with the RAG. This final UI is displayed in Figure 4.10.

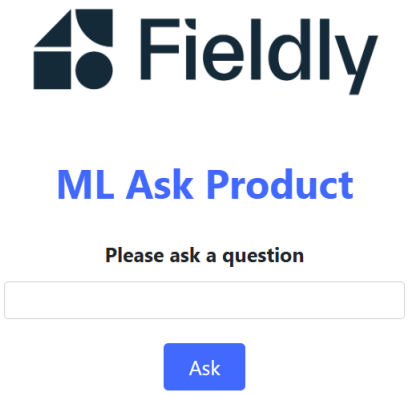


Figure 4.10: The UI created for the Fieldly RAG.

Chapter 5
Discussion

5.1 Impact of Datasets

The results from our evaluations can mainly be categorized into three categories: reader, retriever, and end-to-end. To clearly understand our results it is imperative to understand any impact the data posed on the outcome. First, an overall examination of the data reveals that TechQA is a much "larger" dataset in the sense that the questions, answers, and contexts are much longer than compared to SQuAD. This could create a dynamic that needs to be taken into consideration, especially when comparing results between both datasets. For example, while EM might seem to be a reliable metric for SQuAD, the opposite is true for TechQA. This is clearly observed in the reference answers from SQuAD and TechQA, where the former mostly has words comprised of a couple of words while TechQA sometimes has multiple sentences as answers. This makes having an exact match immensely harder for TechQA while being significantly easier for SQuAD. This is evident in the EM scores, which are much lower for TechQA, making this metric almost useless for evaluating TechQA. However, for SQuAD, EM seems to be more suitable as the generated sentences usually are longer than reference, making it more likely to contain an exact match. Therefore, if these nuances are not taken into consideration, it can be easy to end up with misleading conclusions. Another metric affected by shorter reference answers is recall. Recall only measures how many of the words in the reference answers we capture, thus, it should be easier to get a higher recall when reference answers are short. This is somewhat shown in Tables 4.1 and 4.2, where recall, in general, is higher on SQuAD compared to TechQA. Lastly, SQuAD, which is built on Wikipedia articles, constitutes of relatively high-quality text that has most likely been proofread multiple times, while TechQA, consisting of technotes and human-generated forum questions, should to a higher degree contain more linguistic errors.

5.2 Reader

The reader results from both TechQA and SQuAD highlight a couple of interesting observations:

1. Overall, Phi-3 demonstrated the best performance on TechQA.
2. Flan-T5 had overwhelmingly superior scores on SQuAD.
3. Overall, no major difference in reader performance between SQuAD and TechQA.

Phi-3 on TechQA

Phi-3 and Llama-3 both exhibited the strongest performance on TechQA. When only considering the evaluation metrics, we note how Phi-3 and Llama-3 had similar performance on the dataset. However, when also considering the produced answer lengths for each model, we observe how Phi-3 was more prone to produce answers aligned with reference compared to Llama-3. From Table 4.3, we also distinguish how Phi-3 has the second fastest inference of all the evaluated models. This creates an interesting hypothesis that inference time could be dependent on the size of the model, as Phi-3 was also the second smallest model and Flan-T5 (the smallest model) was the fastest. However, no definitive conclusions should be drawn from this as Llama-3, being the largest model, was significantly faster than Zephyr. Inference speed is most likely dependent on the training process and any model alignments made during training (e.g., how long the output should be) rather than the model size itself. Subsequently, in real-world applications, where user experience is a key factor when building RAGs, Phi-3 would have been preferred over the other slower models.

Flan-T5 on SQuAD

Flan-T5 exhibited an overall unmatched performance on SQuAD. This was a bit surprising at first, due to it being the worst performer on TechQA. However, although not explicitly stated by its creators, it is highly likely that Flan-T5 was fine-tuned on data closely related to SQuAD or that included SQuAD itself. This assumption is supported by the fact that the answer lengths generated by Flan-T5 are closely aligned with reference. Additionally, its high F1 score of 0.849 (almost rivaling human performance) and BERTScore of 0.969 further suggest that Flan-T5 was fine-tuned on a dataset closely resembling SQuAD or the dataset itself. This assumption is plausible, as SQuAD is a popular dataset within QA tasks, which could be attributed to its overall large size and diverse data. When excluding Flan-T5 of the SQuAD reader results, it becomes evident that Phi-3 also performed well on this dataset. Moreover, it is worth noting that Zephyr had the highest EM score on SQuAD, even better than Flan-T5. This can most likely be explained by Zephyr also producing the longest answers of all models, making it more probable to contain the reference answer. Moreover, our results indicate that Flan-T5 appears to struggle more with generating longer and more coherent answers, which is required by TechQA, whereas the other models perform well in this regard.

TechQA vs. SQuAD

Lastly, when comparing the reader results between SQuAD and TechQA and considering the impact their differences can have on results (see section 5.1), we can infer that many of our reader models exhibited comparable capabilities on both datasets. We especially note how similar the F1 scores are for Phi-3 on both TechQA and SQuAD (0.302 versus 0.307). This is not true for Zephyr, which is most likely rooted in the model being more prone to produce longer answers. Long outputs, coupled with short reference answers, can lead to declining precision, which will directly affect the F1 score. Furthermore, as we have already covered, Flan-T5 seems to have an unfair advantage on SQuAD, making any comparison between datasets unjust for this particular model. When examining the model outputs from Phi-3, Llama-3, and Zephyr, we also observe that, although TechQA is more technical and complex in nature, the larger models demonstrated impressive understanding and reasoning capabilities and could even generate highly technical and multi-step instructions based on the context. This was true for SQuAD as well.

5.3 Retriever

Moving on to the retriever, there were a couple of observations that stood out:

1. Jina v3 demonstrated the best performance on TechQA, while Nomic v1.5 performed the best on SQuAD.
2. Sparse retrieval performed much better on SQuAD than TechQA, and BM25 surpassed TF-IDF on both datasets.
3. Overall, the retrievers seem to perform better on SQuAD compared to TechQA.

Dense Retrievers

It was not surprising that the Jina v3 embedding model outperformed the other dense retrievers on TechQA, as it also has the highest MTEB scores among all of our evaluated models. This could be attributed to Jina being both a larger model, enabling it to capture deeper semantic information, as well as having a larger embedding dimension, making it better at storing more information per vector. It is worth noting that GTE v1.5 performed worse for smaller k but improved for $k > 35$ (even matching Jina for large k), while the opposite was true for the Nomic retriever. This indicates that GTE is probably better than Nomic at retrieving hard-to-match cases, i.e., cases where the question and passage differ significantly semantically. Moving on to the retrieval performances on SQUAD, the results seem to be more homogeneous than TechQA. We observe that Nomic v1.5 outperforms the other dense retrievers across all k, exceeding GTE's performance by more than eight percentage points for $k = 1$. This is probably attributed to SQuAD being a part of Nomic's training mix, although, as we mentioned in section 3.3.1, it only constituted a fraction (<0.01%) of the total mix used.

Sparse Retrievers

The next best retriever on SQuAD was the sparse retriever BM25. This was surprising, as our sparse retrievers performed much worse on TechQA, when comparing Recall@k scores. One possible explanation for this discrepancy could be the indexed passages in SQuAD being more distinct in nature, i.e., they contain text or topics that differentiate themselves from the other passages. This way, the sparse retriever might be able to effectively keyword match topics mentioned in both question and passage (e.g. Notre Dame) better than TechQA, where passage topics might overlap with each other, making it harder to differentiate the correct one. This assumption is reasonable, as SQuAD passages consist of distinct Wikipedia articles covering diverse topics, whereas the technotes in TechQA predominantly focus on information related to IBM systems. Another explanation could be that the questions on SQuAD were generated *after* seeing the context (see section 2.4), thereby introducing bias when formulating the question. This was not the case for TechQA, where the questions were asked *before* knowing the answer or seeing the technote. This way, there is a lower chance that the question will contain terms used in the context, making keyword matching much harder. Consequently, BM25 (or TF-IDF) might have a natural advantage on SQuAD over dense retrievers, due to the higher possibility that the question can contain keywords used in the context. Our results also indicate that BM25, in general, seems to outperform TF-IDF, on both datasets and for all k. This makes sense as BM25 was developed as an improvement of TF-IDF, taking into consideration the document length as well when computing the encoding.

TechQA vs. SQuAD

When comparing raw Recall@k scores between TechQA and SQuAD, our results do point to an overall better performance on SQuAD compared to the other. There could be many reasons for this. The fact that SQuAD passages are shorter should, in theory, make dense retrieval easier as less information is needed to be captured in the fixed-length embeddings. Also, due to SQuAD's popularity, we should take into consideration that many dense embedding models offered today may have gotten exposed to SQuAD at some point during their training process. This creates an unfair advantage for SQuAD compared to the lesser-known TechQA. However, even disregarding our dense retrievers, the sparse retrievers also performed better on SQuAD when compared to TechQA. Possible explanations for this have already been covered in the previous section, but it is still an interesting observation. In addition to what has been mentioned, maybe also the heavy alphanumeric filtering on both datasets when performing sparse retrieval affected TechQA more than SQuAD. Due to its technical domain, TechQA does inherently contain more special characters than SQuAD (e.g. see Table 2.4). As a result, during filtering, we may lose more information that is needed to effectively match keywords, compared to SQuAD. Conclusively, the consensus of our results does indicate that retrieval on SQuAD, although maybe not because of the domain, exhibits overall better performance than compared to TechQA. To determine more in-depth the impact the domain has on retrieval, more research is needed.

5.3.1 Enhanced Retrieval Methods

Hybrid

As we have shown, choosing good retrievers is not the only way to improve recall performance. Although having a good baseline retriever as foundation is important, adding additional steps to the retrieval process could improve performance equally as much. This is true for hybrid retrieval on SQuAD where a significant improvement can be seen when combining Jina with BM25 retriever. For TechQA, however, the resulting hybrid performance differs. Judging from Figure 4.7, we note that for $5 < k < 30$, the performance is actually worse than Jina, and for $30 < k < 50$, the hybrid retriever performs better than each individual component. This discrepancy is somewhat unexpected and difficult to interpret. However, it could be attributed to the significant performance gap displayed between BM25 and Jina on TechQA, which reaches up to 15 percentage points for $10 < k < 20$. When this gap narrows for larger k, the hybrid retriever then starts outperforming. Consequently, one could assume that if both retrievers were to have similar performance (i.e., like SQuAD), then hybrid retrieval would result in an overall improvement. However, this can not be concluded as more experiments are needed to confirm this hypothesis.

Reranking

Another demonstrated method for improving retrieval performance is reranking. Adding a reranker made a significant difference for SQuAD, which was surprising at first due to the large noticeable improvement. From prior discussions, whenever such impressive performance was observed, a recurring thought was that the model might have already encountered the data during its training. Although we could not find any details about the data Jina Reranker v2 was trained on or if SQuAD was involved during this process, it is nonetheless difficult to overlook that the performance gain on SQuAD seems almost too good to be true. This is further solidified by the much smaller gain on TechQA (which is a less well-known dataset). Conclusively, reranking did improve Recall@k and thus should be considered when building retrieval-augmented generative systems.

Query Expansion

Lastly, not all enhanced retrieval methods demonstrated improved retrieval performance. An example of that is query expansion, which performed worse than expected, even worse than without using it for SQuAD. In theory, this method is attractive due to its ability to enhance the search string, thus increasing the probability of matching with the correct passage. However, in practice, it appears to be harder to implement in a way that effectively improves performance. Factors such as which language model to select as agent and how to transform the search string are most likely meaningful parameters to consider when performing query expansion. The types of transformations that can be done are also dictated by the language model chosen, as the size and design of the model determine the actions it can perform. In our case, the combination of Flan-T5 as agent and appending hypothetical answers to the questions did not improve retrieval performance. However, it cannot be ruled out that an alternative combination of language model and transformation method could result in a different outcome.

5.4 End-to-End

We chose to conduct end-to-end testing on different combinations of high-performing components and methods revealed by the previous experiments. Looking at Table 4.6, shows inconclusive scores across many of our configurations on TechQA. For example, Hybrid had the highest F1 score for $k = [1, 5]$ while Reranker had the highest for $k = 3$. Moreover, while Hybrid had the highest F1 score on $k = 5$, Base exhibited the highest BERTScore for the same k. Furthermore, the difference between scores across our setups was also, in many cases, marginal (e.g., 0.225 versus 0.227 for Base versus Hybrid on TechQA, $k = 1$), making the results even harder to interpret. Should Hybrid really be categorized as the better configuration if the score is just 2 milli-percentage points higher? Probably not.

The results just mentioned highlight the difficulties associated with evaluating and ranking RAG configurations based on bag-of-word or n-gram-based metrics (F1, BERTScore, BLEU), etc.). It also shows the limitations of the metrics as they lack the capability to capture the nuances of an answer, such as quality, detailedness, and coherence. In such cases, human evaluation may be a better option, although it is inherently more time-consuming. Another potential option is *LLM-as-judge*, an empiric evaluation process where one or more LLM(s), based on a generated and reference answer, outputs a similarity score. Finally, by looking at Table 4.7, we notice how the scores on SQuAD seem to be more consistent across all setups and k. For example, we get a consensus from all metrics that Reranker is the best-performing setup for $k = 1$, while Hybrid is the best for $k = 3$. The scores are also more decisive, meaning that the difference between values is larger (e.g., 0.290 vs. 0.255 for Reranker vs. Base when $k = 3$). The noticeable margins can most likely be explained by the large differences in performance on retrieval. For example, in Figure 4.9, we showed that Reranking on SQuAD improved Recall@k significantly versus a vanilla-dense retriever. Therefore, what we see is potentially the manifestations of the Recall@k improvements on E2E results.

5.5 Fieldly

From our qualitative evaluations of the Fieldly RAG, a couple of interesting observations emerged:

1. Sparse retrieval serves as a good complement to dense retrievers.
2. The TechQA results generalize well to the Fieldly data.

First, we noticed that adding a sparse retriever (i.e. using hybrid retrieval) greatly improved our performance on some questions. This is best illustrated by the first question in Table 4.8, where a context containing "chief revenue officer" had to be extracted. In our evaluations, the sparse retriever excelled in such a situation while the dense retriever struggled a lot more. This points to the fact that dense retrievers, even though very powerful, can perform poorly on incoherent data and situations where keywords are more important than semantics. Finally, we observed that the configurations that performed best on TechQA also did well on the Fieldly data. This was emphasized by the superior performance of Jina v3, compared to Nomic v1.5, which aligns with the results observed on TechQA. Moreover, while

we did not test other reader models, Phi-3 produced well-balanced answers with appropriate sentence lengths, consistent with previous findings.

5.6 Ethical Considerations

RAGs are systems which can potentially be used to increase productivity and efficiency during information gathering. By leveraging an external knowledge bank, LLMs can be given the ability to answer some very task specific questions while outputing an easy to understand language. However, with the rise of this technology, the need to address ethical considerations also grows.

5.6.1 Bias

One aspect to keep in mind when building RAGs is bias. The application relies heavily on the data it augments on, therefore, ensuring no bias in the data itself is important. If the data contains more information about a certain gender or topic, the output in those areas will subsequently be better and more detailed. Thus, to ensure fairness in the answers, the developer should strive for balanced and nuanced data from a broad range of sources. It is also worth keeping in mind that, depending on the training process, there might exist inherent bias within AI models. This could manifest itself in skewed rankings from the dense retriever or biased responses from the language model. Therefore, it is important to consider those aspects when building RAGs and select models accordingly.

5.6.2 Data Privacy

Another important aspect to consider is data privacy. Storing and managing information in an external knowledge base carries the risk of including sensitive and confidential data, potentially creating both legal and ethical issues. The language model used can also not differentiate between sensitive from non-sensitive data, meaning that it will output whatever it receives. It is therefore important to review and consider what data can be included in the knowledge bank to not violate any ethical or regulatory frameworks (e.g. GDPR).

Chapter 6
Conclusion

During our work we explored the optimal configurations and mechanics behind building a retrieval-augmented generative system for technical documentation. Experiments were conducted on both sparse and dense retrievers like BM25 and Jina v3 while also exploring some alternative enhanced retrieval methods such as hybrid retrieval, reranking, and query expansion. Moreover, different generative language models were assessed, such as Phi-3, Llama-8, Zephyr, and Flan-T5. By using two separate datasets in different domains, TechQA and SQuAD, we were able to capture the nuances and the impact domain-different data can have on RAG performance. We found that Jina-Embeddings-v3 excels on TechQA ($\mathit{Recall}@5 = 0.625$), while BM25 performs best on SQuAD ($\mathit{Recall}@5 = 0.833$). Furthermore, Phi-3 Mini-128K-Instruct emerged as the optimal reader for TechQA, achieving a BERTScore of 0.865. Finally, by distilling findings from TechQA, an optimal RAG was built for Fieldly AB, which used hybrid retrieval (Jina v3 and BM25) combined with Phi-3 as the reader.

We now try to answer our initial research questions:

1. Which retrieval methods work best for retrieval-augmented generation in the technical domain?
2. Which generative models are the most accurate, efficient, and user-satisfactory for retrieval-augmented generation in the technical domain?
3. What are the performance differences for a RAG handling data in the technical domain compared to other domains such as trivia and popular science?

Question 1

We conclude that Jina v3, combined with BM25, works best on technical data. This is supported by Jina v3 surpassing all other dense retrievers on Recall@k on TechQA, while BM25

was the best-performing sparse retriever. Combining the two retrievers (i.e., hybrid retrieval) showed outperformance for small k ($k < 3$) on TechQA, and qualitative assessment on the Fieldly RAG confirmed this, where $k = 3$.

Question 2

We found that Phi-3 delivered impressive performance on technical data, especially given its compact size compared to larger models such as Llama-3 and Zephyr. Phi-3 exhibited the best performance on BERTScore for TechQA (0.865), while having the second fastest inference times. It also produced answers closely aligned with reference. These factors highlight that Phi-3 is not only accurate and efficient in its answers, but also user satisfactory. Furthermore, qualitative assessment of the Fieldly data, using human judgment, also supports this conclusion.

Question 3

We conclude that domain differences did not have a noteworthy impact on reader performance. Our readers exhibited impressive capabilities on both the technical and trivia datasets. The differences that we did observe (on EM and Recall) were caused more by dataset differences, such as reference answer lengths, rather than the domain itself. However, for retrievers, we did observe an overall better performance on SQuAD compared to TechQA. We attribute this to a couple of potential reasons, such as SQuAD being a popular dataset for training, the development process of SQuAD (where questions were generated after seeing the context), and SQuAD having shorter passages. Therefore, the better retrieval on SQuAD, when compared to TechQA, should not be solely attributed to the domain itself, as not all trivia datasets are expected to yield similar results as SQuAD. Consequently, further research is needed for a more definitive conclusion on this question.

6.1 Evaluation

Finally, we also wish to highlight the difficulty and limitations of using bag-of-words and n-gram-based metrics like F1 and BLEU. Although they can provide a good indication of the correctness of an answer, these metrics lack the ability to capture broader aspects, such as the detailedness and coherency of an answer. Moreover, other metrics, like EM, are heavily affected by the length ratio between the generated answer and reference answer and should be used carefully depending on the dataset. When reference answers are short and generated output long, EM might serve as a reliable indication of the accuracy of an answer. Conclusively, although time-consuming, human evaluation is still preferred whenever possible.

6.2 Future Work

Building RAGs and evaluating different configurations can be an exhaustive process, mainly due to the many components and combinations that can be tuned in this process. In our work, we introduce five different retrievers and four reader models. Furthermore, we evaluate three enhanced retrieval methods: Hybrid retrieval, Reranking, and Query expansion. However, as stated in section 3, delimitations had to be put forth due to project constraints and the many choices available. Therefore, we list some potential topics which we think are interesting for future research.

- **More domain research.** We evaluated SQuAD versus TechQA, but, as mentioned in our conclusions, more research has to be done in this area to give a definitive answer. Preferably, two datasets with more similar characteristics should be used. This might be difficult, due to the limited number of public datasets available, but it should not be rejected as more datasets might become available in the future.
- **Expand query expansion**. We did not find any performance boosts from using query expansion, but, as we mentioned in our Discussion, there are more transformation methods available (instead of appending a hypothetical answer). Moreover, you could also explore other agents to use (in our thesis we chose Flan-T5).
- **Similarity search**. In our thesis, we used cosine similarity as a scoring function for retrievers. In the future, more scoring functions could be explored. This might include K-Nearest-Neighbor or some Tree-based search.
- **Sparse retrievers**. We used BM25 and TF-IDF, but more intricate sparse retrievers can be tried and evaluated upon.
- **LLM-as-judge**. In our thesis, we discussed the challenges of using bag-of-words and n-gram-based metrics for evaluating generative output. As an alternative, future work could explore leveraging a collection of language models to assess similarity between segments and generate a score based on perceived similarity.

References

[Abdin et al., 2024] Abdin, M., Aneja, J., Awadalla, H., Awadallah, A., Awan, A. A., Bach, N., Bahree, A., Bakhtiari, A., Bao, J., Behl, H., et al. (2024). Phi-3 technical report: A highly capable language model locally on your phone. *arXiv preprint arXiv:2404.14219*.

[Adams et al., 2024] Adams, L. C., Truhn, D., Busch, F., Dorfner, F., Nawabi, J., Makowski, M. R., and Bressem, K. K. (2024). Llama 3 challenges proprietary state-of-the-art large language models in radiology board–style examination questions. *Radiology*, 312(2):e241191. PMID: 39136566.

[Bahdanau et al., 2014] Bahdanau, D., Cho, K., and Bengio, Y. (2014). Neural machine translation by jointly learning to align and translate. *CoRR*, abs/1409.0473.

[Bajaj et al., 2016] Bajaj, P., Campos, D., Craswell, N., Deng, L., Gao, J., Liu, X., Majumder, R., McNamara, A., Mitra, B., Nguyen, T., et al. (2016). Ms marco: A human generated machine reading comprehension dataset. *arXiv preprint arXiv:1611.09268*.

[Castelli et al., 2020] Castelli, V., Chakravarti, R., Dana, S., Ferritto, A., Florian, R., Franz, M., Garg, D., Khandelwal, D., McCarley, S., McCawley, M., Nasr, M., Pan, L., Pendus, C., Pitrelli, J., Pujar, S., Roukos, S., Sakrajda, A., Sil, A., Uceda-Sosa, R., Ward, T., and Zhang, R. (2020). The TechQA dataset. In Jurafsky, D., Chai, J., Schluter, N., and Tetreault, J., editors, *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, pages 1269–1278, Online. Association for Computational Linguistics.

[Chang et al., 2024] Chang, Y., Wang, X., Wang, J., Wu, Y., Yang, L., Zhu, K., Chen, H., Yi, X., Wang, C., Wang, Y., Ye, W., Zhang, Y., Chang, Y., Yu, P. S., Yang, Q., and Xie, X. (2024). A survey on evaluation of large language models. *ACM Trans. Intell. Syst. Technol.*, 15(3).

[Chicco and Jurman, 2020] Chicco, D. and Jurman, G. (2020). The advantages of the matthews correlation coefficient (mcc) over f1 score and accuracy in binary classification evaluation. *BMC Genomics*, 21.

[Chung et al., 2024] Chung, H. W., Hou, L., Longpre, S., Zoph, B., Tay, Y., Fedus, W., Li, Y., Wang, X., Dehghani, M., Brahma, S., Webson, A., Gu, S. S., Dai, Z., Suzgun, M., Chen, X.,

Chowdhery, A., Castro-Ros, A., Pellat, M., Robinson, K., Valter, D., Narang, S., Mishra, G., Yu, A., Zhao, V., Huang, Y., Dai, A., Yu, H., Petrov, S., Chi, E. H., Dean, J., Devlin, J., Roberts, A., Zhou, D., Le, Q. V., and Wei, J. (2024). Scaling instruction-finetuned language models. *Journal of Machine Learning Research*, 25(70):1–53.

[Douze et al., 2024] Douze, M., Guzhva, A., Deng, C., Johnson, J., Szilvasy, G., Mazaré, P.-E., Lomeli, M., Hosseini, L., and Jégou, H. (2024). The faiss library. *arXiv preprint arXiv:2401.08281*.

[Dubey et al., 2024] Dubey, A., Jauhri, A., et al. (2024). The Llama 3 Herd of Models. *arXiv e-prints*, page arXiv:2407.21783.

[Han et al., 2023] Han, Y., Liu, C., and Wang, P. (2023). A comprehensive survey on vector database: Storage and retrieval technique, challenge. *arXiv preprint arXiv:2310.11703*.

[Hoffmann et al., 2022] Hoffmann, J., Borgeaud, S., Mensch, A., Buchatskaya, E., Cai, T., Rutherford, E., de las Casas, D., Hendricks, L. A., Welbl, J., Clark, A., Hennigan, T., Noland, E., Millican, K., van den Driessche, G., Damoc, B., Guy, A., Osindero, S., Simonyan, K., Elsen, E., Vinyals, O., Rae, J. W., and Sifre, L. (2022). An empirical analysis of compute-optimal large language model training. In Oh, A. H., Agarwal, A., Belgrave, D., and Cho, K., editors, *Advances in Neural Information Processing Systems*.

[Hu et al., 2022] Hu, E. J., yelong shen, Wallis, P., Allen-Zhu, Z., Li, Y., Wang, S., Wang, L., and Chen, W. (2022). LoRA: Low-rank adaptation of large language models. In *International Conference on Learning Representations*.

[HuggingFace, 2024] HuggingFace (2024). Mteb leaderboard. `https://huggingface.co/spaces/mteb/leaderboard` [Acccessed: 01-11-2024].

[Ibrahim, 2024] Ibrahim, M. (2024). Fine-grained language-based reliability detection in spanish new with fine-tuned llama-3 model. In *IberLEF@SEPLN*.

[Izacard and Grave, 2020] Izacard, G. and Grave, E. (2020). Distilling knowledge from reader to retriever for question answering. *arXiv preprint arXiv:2012.04584*.

[Jagerman et al., 2023] Jagerman, R., Zhuang, H., Qin, Z., Wang, X., and Bendersky, M. (2023). Query expansion by prompting large language models. *arXiv preprint arXiv:2305.03653*.

[Kapetanovic, Samil and Raja, Mohammad, 2024] Kapetanovic, Samil and Raja, Mohammad (2024). Enhancing System Documentation Accessibility through Question-Answering. Student Paper.

[Kaplan et al., 2020] Kaplan, J., McCandlish, S., Henighan, T., Brown, T. B., Chess, B., Child, R., Gray, S., Radford, A., Wu, J., and Amodei, D. (2020). Scaling laws for neural language models. *arXiv preprint arXiv:2001.08361*.

[Karpukhin et al., 2020] Karpukhin, V., Oguz, B., Min, S., Lewis, P., Wu, L., Edunov, S., Chen, D., and Yih, W.-t. (2020). Dense passage retrieval for open-domain question answering. In Webber, B., Cohn, T., He, Y., and Liu, Y., editors, *Proceedings of the 2020*

Conference on Empirical Methods in Natural Language Processing (EMNLP), pages 6769–6781, Online. Association for Computational Linguistics.

[Lewis et al., 2020] Lewis, P., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W.-t., Rocktäschel, T., Riedel, S., and Kiela, D. (2020). Retrieval-augmented generation for knowledge-intensive nlp tasks. In *Proceedings of the 34th International Conference on Neural Information Processing Systems*, NIPS '20, Red Hook, NY, USA. Curran Associates Inc.

[Li et al., 2023] Li, Z., Zhang, X., Zhang, Y., Long, D., Xie, P., and Zhang, M. (2023). Towards general text embeddings with multi-stage contrastive learning. *arXiv preprint arXiv:2308.03281*.

[Moreira et al., 2024] Moreira, G. d. S. P., Ak, R., Schifferer, B., Xu, M., Osmulski, R., and Oldridge, E. (2024). Enhancing q&a text retrieval with ranking models: Benchmarking, fine-tuning and deploying rerankers for rag. *arXiv preprint arXiv:2409.07691*.

[Nussbaum et al., 2024] Nussbaum, Z., Morris, J. X., Duderstadt, B., and Mulyar, A. (2024). Nomic embed: Training a reproducible long context text embedder. *arXiv preprint arXiv:2402.01613*.

[Ouyang et al., 2024] Ouyang, L., Wu, J., Jiang, X., Almeida, D., Wainwright, C. L., Mishkin, P., Zhang, C., Agarwal, S., Slama, K., Ray, A., Schulman, J., Hilton, J., Kelton, F., Miller, L., Simens, M., Askell, A., Welinder, P., Christiano, P., Leike, J., and Lowe, R. (2024). Training language models to follow instructions with human feedback. In *Proceedings of the 36th International Conference on Neural Information Processing Systems*, NIPS '22, Red Hook, NY, USA. Curran Associates Inc.

[Papineni et al., 2002] Papineni, K., Roukos, S., Ward, T., and Zhu, W.-J. (2002). Bleu: a method for automatic evaluation of machine translation. In *Proceedings of the 40th Annual Meeting on Association for Computational Linguistics*, ACL '02, page 311–318, USA. Association for Computational Linguistics.

[Rafailov et al., 2023] Rafailov, R., Sharma, A., Mitchell, E., Ermon, S., Manning, C. D., and Finn, C. (2023). Direct preference optimization: your language model is secretly a reward model. In *Proceedings of the 37th International Conference on Neural Information Processing Systems*, NIPS '23, Red Hook, NY, USA. Curran Associates Inc.

[Rajpurkar et al., 2018] Rajpurkar, P., Jia, R., and Liang, P. (2018). Know what you don't know: Unanswerable questions for SQuAD. In Gurevych, I. and Miyao, Y., editors, *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 2: Short Papers)*, pages 784–789, Melbourne, Australia. Association for Computational Linguistics.

[Rajpurkar et al., 2016] Rajpurkar, P., Zhang, J., Lopyrev, K., and Liang, P. (2016). SQuAD: 100,000+ questions for machine comprehension of text. In Su, J., Duh, K., and Carreras, X., editors, *Proceedings of the 2016 Conference on Empirical Methods in Natural Language Processing*, pages 2383–2392, Austin, Texas. Association for Computational Linguistics.

[Ramos et al., 2003] Ramos, J. et al. (2003). Using tf-idf to determine word relevance in document queries. In *Proceedings of the first instructional conference on machine learning*, volume 242, pages 29–48. Citeseer.

[Robertson et al., 2004] Robertson, S., Zaragoza, H., and Taylor, M. (2004). Simple bm25 extension to multiple weighted fields. In *Proceedings of the Thirteenth ACM International Conference on Information and Knowledge Management*, CIKM '04, page 42–49, New York, NY, USA. Association for Computing Machinery.

[Schmidt, 2019] Schmidt, R. M. (2019). Recurrent neural networks (rnns): A gentle introduction and overview. *arXiv preprint arXiv:1912.05911*.

[Siriwardhana et al., 2023] Siriwardhana, S., Weerasekera, R., Wen, E., Kaluarachchi, T., Rana, R., and Nanayakkara, S. (2023). Improving the domain adaptation of retrieval augmented generation (RAG) models for open domain question answering. *Transactions of the Association for Computational Linguistics*, 11:1–17.

[Sturua et al., 2024] Sturua, S., Mohr, I., Kalim Akram, M., Günther, M., Wang, B., Krimmel, M., Wang, F., Mastrapas, G., Koukounas, A., Wang, N., and Xiao, H. (2024). jina-embeddings-v3: Multilingual Embeddings With Task LoRA. *arXiv e-prints*, page arXiv:2409.10173.

[Su et al., 2024] Su, J., Ahmed, M., Lu, Y., Pan, S., Bo, W., and Liu, Y. (2024). Roformer: Enhanced transformer with rotary position embedding. *Neurocomputing*, 568:127063.

[Tunstall et al., 2023] Tunstall, L., Beeching, E., Lambert, N., Rajani, N., Rasul, K., Belkada, Y., Huang, S., von Werra, L., Fourrier, C., Habib, N., et al. (2023). Zephyr: Direct distillation of lm alignment. *arXiv preprint arXiv:2310.16944*.

[Vaswani et al., 2017] Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. In *Proceedings of the 31st International Conference on Neural Information Processing Systems*, NIPS'17, page 6000–6010, Red Hook, NY, USA. Curran Associates Inc.

[Wang et al., 2024] Wang, X., Wang, Z., Gao, X., Zhang, F., Wu, Y., Xu, Z., Shi, T., Wang, Z., Li, S., Qian, Q., Yin, R., Lv, C., Zheng, X., and Huang, X. (2024). Searching for best practices in retrieval-augmented generation. In Al-Onaizan, Y., Bansal, M., and Chen, Y.-N., editors, *Proceedings of the 2024 Conference on Empirical Methods in Natural Language Processing*, pages 17716–17736, Miami, Florida, USA. Association for Computational Linguistics.

[Xia et al., 2015] Xia, P., Zhang, L., and Li, F. (2015). Learning similarity with cosine similarity ensemble. *Information Sciences*, 307:39–52.

[Yang et al., 2018] Yang, Z., Qi, P., Zhang, S., Bengio, Y., Cohen, W., Salakhutdinov, R., and Manning, C. D. (2018). HotpotQA: A dataset for diverse, explainable multi-hop question answering. In Riloff, E., Chiang, D., Hockenmaier, J., and Tsujii, J., editors, *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 2369–2380, Brussels, Belgium. Association for Computational Linguistics.

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

[Yu et al., 2021] Yu, W., Wu, L., Deng, Y., Zeng, Q., Mahindru, R., Guven, S., and Jiang, M. (2021). Technical question answering across tasks and domains. In Kim, Y.-b., Li, Y., and Rambow, O., editors, *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies: Industry Papers*, pages 178–186, Online. Association for Computational Linguistics.

[Zhang et al., 2024a] Zhang, J., Mai, Z., Xu, Z., and Xiao, Z. (2024a). Is llama 3 good at identifying emotion? a comprehensive study. In *Proceedings of the 2024 7th International Conference on Machine Learning and Machine Intelligence (MLMI)*, MLMI '24, page 128–132, New York, NY, USA. Association for Computing Machinery.

[Zhang et al., 2023] Zhang, Q., Chen, S., Xu, D., Cao, Q., Chen, X., Cohn, T., and Fang, M. (2023). A survey for efficient open domain question answering. In Rogers, A., Boyd-Graber, J., and Okazaki, N., editors, *Proceedings of the 61st Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 14447–14465, Toronto, Canada. Association for Computational Linguistics.

[Zhang et al., 2020] Zhang, T., Kishore, V., Wu, F., Weinberger, K. Q., and Artzi, Y. (2020). Bertscore: Evaluating text generation with bert.

[Zhang et al., 2024b] Zhang, X., Zhang, Y., Long, D., Xie, W., Dai, Z., Tang, J., Lin, H., Yang, B., Xie, P., Huang, F., Zhang, M., Li, W., and Zhang, M. (2024b). mgte: Generalized long-context text representation and reranking models for multilingual text retrieval.

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2025-02-19

EXAMENSARBETE Retrieval-Augmented Generation for Technical Question Answering
STUDENT Victor Tiet
HANDLEDARE Marcus Klang (LTH)
EXAMINATOR Jacek Malec (LTH)

Besvarning av tekniska frågor med hjälp av språkmodeller

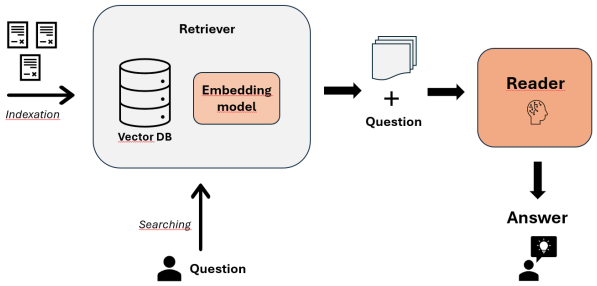
POPULÄRVETENSKAPLIG SAMMANFATTNING **Victor Tiet**

Många företag har idag samlat stora mängder intern dokumentation om sina tekniska system. Tänk om man kunde skapa en intern frågeassistent som, med tillgång till denna dokumentation, kan svara på anställdas frågor på ett automatisk sätt? Detta examensarbete kommer att utforska just den möjligheten.

I takt med att stora språkmodeller har blivit allt populärare har även tjänster kopplade till dem ökat. Ett exempel på ett populärt frågesvar system idag är ChatGPT. I många fall så har dem underliggande modellerna tränats på stora mängder offentlig data, vilket har gett dem breda kunskaper. Men hur får man en språkmodell att svara på frågor som handlar om specifik information? På information som kanske inte finns tillgängligt offentligt eller till och med innehåller känsliga företagshemligheter som inte kan delas med?

Många företag hanterar stora mängder intern data i form av dokumentation om sina tekniska system och produkter. För anställda kan det ibland vara både svårt och tidskrävande att hitta rätt information. Tänk om man kunde skapa en intern frågeassistent som automatiskt hämtar relevant information och sedan genererar ett svar?

I mitt examensarbete utforskar jag just detta. Vi visar att detta kan uppnås genom att implementera en *retriever*, som hämtar relevant information och sedan skickar den vidare till språkmodellen (här kallad *reader*). Enkelt uttryckt: om språkmodellen är en student som behöver information för att skriva en uppsats, fungerar retrievern som bibliotekarien som hjälper till att hitta rätt material.



Vårt arbete undersöker hur man bygger den optimala RAG-modellen för teknisk dokumentation genom att testa både olika språkmodeller och retrievers. Vi undersöker även hur datans domän kan påverka prestanda. För att göra detta genomför vi experiment på både teknisk dokumentation och populärvetenskapliga texter. Bland de modeller vi testar kommer vi fram till att Phi-3 är den bästa språkmodellen för den tekniska domänen, medan Jina-Embeddings-v3 är den bäst presterande retrievern inom samma område. Vi identifierar också vissa prestandaskillnader mellan domänerna, men drar slutsatsen att ytterligare forskning krävs för att ge ett tydligare svar på domänens inverkan på prestandan.