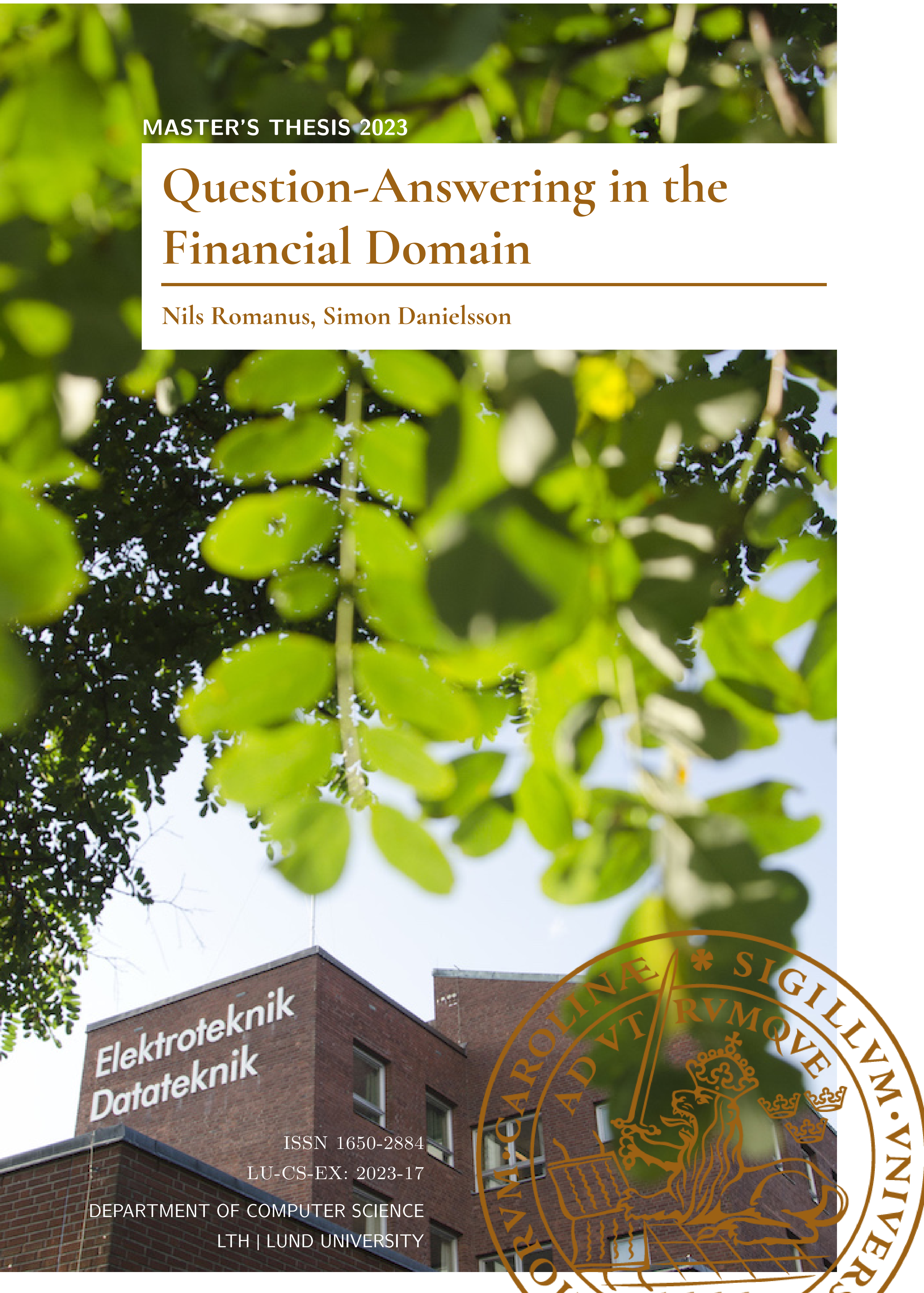
MASTER'S THESIS 2023

# Question-Answering in the Financial Domain

Nils Romanus, Simon Danielsson



ISSN 1650-2884
LU-CS-EX: 2023-17

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2023-17

# Question-Answering in the Financial Domain

Fråge-svarsystem i en bankmiljö

**Nils Romanus, Simon Danielsson**

# Question-Answering in the Financial Domain

## (An evaluation of Open-retrieval Question Answering Systems for Banking)

Nils Romanus
`ni5324ro-s@student.lu.se`

Simon Danielsson
`si7660da-s@student.lu.se`

June 12, 2023

Master's thesis work carried out at SEB.

Supervisor: Pierre Nugues, `pierre.nugues@cs.lth.se`

Examiner: Flavius Gruian, `flavius.gruian@cs.lth.se`

## Abstract

Question-answering systems have gained significant attention for providing specific information to users through natural language interfaces. While these systems are often optimized on benchmark datasets, their performances in real-life applications remain less known. This Master's thesis, in collaboration with the bank SEB, explores the feasibility of effective open-retrieval question-answering systems in banking. We evaluate both extractive and generative Readers, two augmentation techniques, and over 60 system configurations across four datasets.

Our study demonstrates these systems' viability for banking. They perform competitively on multiple benchmarks, with a 25-29% performance decrease in the banking context. We observe a weak correlation between lexical performance on benchmarks and banking-specific data, but a stronger correlation in semantic overlap. We show that this partly arises from limitations in evaluating generative models using lexical metrics, emphasizing the importance of task-specific systems and metrics aligned with the requirements and characteristics associated with the use case.

**Keywords**: NLP, Question-answering, Information Retrieval, Banking, Machine Reading Comprehension, Large Language Models

# Acknowledgements

We would like to express our sincere appreciation to our thesis supervisor, Prof. Pierre Nugues, for his valuable guidance and support throughout our research journey. We are grateful for his insightful feedback and dedication, which greatly contributed to the development and quality of this thesis.

We would also like to thank our supervisors at SEB, Amin Ahmadi, and Rahul Biswas, for their valuable industry knowledge and support. Their expertise and guidance were instrumental in conducting this research within the banking context.

We extend our gratitude to the team at SEB for providing a conducive research environment and their contributions during discussions and meetings.

# Contents

# Chapter 1

# Introduction

A *question-answering* (QA) system aims to provide specific information to users through a natural language interface. Traditional QA systems extract answers from documents provided by the user in response to a question (Chen and Yih, 2020). Another type of QA system is the *open-retrieval* question-answering (OpenQA) system, which searches for answers in a collection of documents rather than a single user-provided context (Zhu et al., 2021). Currently, large language models and other neural architectures are commonly employed to tackle these tasks (Floridi and Chiriatti, 2020; Izacard et al., 2022; Calijorne Soares and Parreiras, 2020). As a result, the field has become highly competitive, with state-of-the-art performance being updated every other month (Zhu et al., 2021).

The demand for enterprise-level adoption of QA systems is rapidly increasing (Gao et al., 2021). This trend can be seen in the success of various systems, such as Roller et al. (2020), Zeng et al. (2020), Bao et al. (2020), Chakravarti et al. (2020), Tang et al. (2020), and OpenAI's *ChatGPT* (Lund and Wang, 2023). However, while builders of QA systems often optimize their models for performance on benchmark datasets, their models' performance on real-life data remains less clear (Alanazi et al., 2021). Adopting new technologies can be challenging in an enterprise setting, especially when benchmark datasets provide a different setting and goal from the specific requirements of each industry. Consequently, industries may struggle to successfully implement these technologies.

In this work, conducted in collaboration with the Swedish bank *SEB*, we demonstrate the feasibility of constructing OpenQA systems for banking applications. Specifically, our systems achieve close to state-of-the-art performance on several benchmark datasets. Our systems generalize to the banking domain moderately well but exhibit a performance decrease of 25-29%. Moreover, we find that lexical metrics are insufficient for properly describing generative model performance. Rather, we observe that semantic metrics better correlate with analysts' assessments of answer correctness. This is partly due to that lexical metrics (1) do not account for semantic overlap and (2) overestimate performance on numerical questions. Furthermore, we find that good lexical benchmark performance does not generally translate well to domain-specific performance. In this case, semantic metrics on benchmarks better extrapolate to the banking setting. To foster further research and collaboration, we proudly release our code as open-source[1].

Our work highlights the significance of tailoring the configuration of a QA system to suit the specific application, as opposed to simply relying on widely used out-of-the-box components and metrics. Specifically, we find that relying solely on the current state-of-the-art is inadequate for fulfilling all the needs of a particular business. Instead, it is crucial to consider the application's characteristics when determining which components to use.

We anticipate that the effective implementation of QA systems in the industry will necessitate additional research into particular applications and their interaction with the system's components. In particular, we must gain a better understanding of the underlying mechanisms that determine why certain architectures perform poorly in certain domains, and also how to design appropriate application-specific evaluation metrics.

## 1.1 Objective

In this section, we outline the objective of the thesis work by presenting our main research questions and contributions. To provide a more clear understanding of the banking domain, we subsequently give three examples of banking use cases and formulate a set of assumptions that generalize these examples.

### 1.1.1 Research Questions

To investigate the performance and suitability of OpenQA systems in the banking domain, we answer the following research questions:

- Which OpenQA architectures exhibit superior performance on benchmark datasets and banking-specific data?
- Is there a correlation between the performance of OpenQA systems on benchmark datasets and their performance on banking-specific data?
- Do lexical evaluation metrics effectively evaluate the performance of generative models on banking data?
- What application-specific characteristics significantly impact the performance of QA systems in the banking domain?

### 1.1.2 Contribution

This thesis addresses the gap in the literature by investigating the real-life performance of OpenQA systems in the competitive field of question-answering. Specifically, it evaluates different architectures, explores the correlation between the benchmark and banking-specific performance, assesses

[1]`https://github.com/simondanielsson/SEB-OpenQA`

the effectiveness of lexical evaluation metrics, and identifies application-specific characteristics. By shedding light on the practical performance of these systems in the banking domain, this research makes a contribution to bridging the gap between benchmark performance and real-life applications. Both authors have contributed equally to both experimentation and writing, with specializations specified in Table D.1.

### 1.1.3 Banking use cases

In order to examine the feasibility of OpenQA systems in the banking domain, it is important to establish a clear definition of what is meant by the "banking domain". First, we provide three example use cases for OpenQA in banking. These examples provide a basis for what we consider to be our banking application. Then, we generalize these examples and construct a set of assumptions that implicitly define the OpenQA task in the banking setting.

**InvestmentQA.** The InvestmentQA use case focuses on providing precise and detailed answers to investment-related queries. It involves answering questions such as "What were the dividends of Example AB in the fourth quarter of 2023?" or "What is the historical performance of Company XYZ's stock?". By leveraging OpenQA systems, users can obtain accurate financial information and make informed investment decisions based on the extracted insights.

**ComplianceQA.** The ComplianceQA use case is designed to address internal regulatory and compliance inquiries. It involves answering questions related to legal and regulatory frameworks, policies, and procedures. For example, questions like "What are the documentation requirements for opening a business account?" or "What are the restrictions on international money transfers?" can be effectively addressed using an OpenQA system. By providing accurate and up-to-date compliance information, these systems assist banking professionals in ensuring adherence to industry standards and regulations.

**NewsletterQA.** In the NewsletterQA use case, a human compiles a newsletter from information extracted from a knowledge base using a QA system. For instance, the newsletter could consist of news about dividends, stock prices, or aquisitions – the QA system must have the flexibility to fetch information about any of these topics. In contrast to the two previous use cases, the answers from the QA system is not passed on directly to an end user. Instead, an employee asks questions to the QA system (depending on what type of newsletter is to be compiled), and then embeds the returned answers in the text of the newsletter. Since this is an application of human-in-the-loop AI, we can allow for returning multiple answers to a given question.

**Assumptions.** These use cases pose some broad requirements for the QA system. We phrase these as assumptions for our banking application, which will subsequently guide us in our decisions about which system components to use. These are summarized as follows.

1. **Open-retrieval.** There exists lots of documents with information one might want to ask questions about. We do not know *a priori* in which document one can potentially find the answer to any given question.

2. **Modality.** The documents contain unstructured or semi-structured text only.

3. **Temporality.** All documents are static and from a short period of time.
4. **Language.** All documents and questions are in English.
5. **Jargon.** The questions and documents often contain financial jargon and abbreviations.
6. **Factoid.** Questions are often concerned with finding concise answers: like named entities, numbers, yes/no answers, or short descriptions.
7. **Number of answers.** Most often we require one answer to a given question. However, in cases where a human is to assist the AI system, we can allow for multiple answers to be outputted by the system for later filtering by the human.

We do not make assumptions regarding deployment factors like latency, throughput, and hardware costs, as they are outside the scope of this thesis.

## 1.2 Related work

Since the 1960s lots of research has been devoted to building systems that can process queries in natural language from a user and return precise answers (Mishra and Jain, 2016). Both their architectures and use cases have changed quite significantly over the years. While they in the early days often functioned as front-ends for structured databases (Dwivedi and Singh, 2013), they are now capable of processing textual data on a semantic level and answering questions even without contact with an external knowledge base thanks to the development of large language models (LLMs) (Radford et al., 2019; Floridi and Chiriatti, 2020; OpenAI, 2023; Lund and Wang, 2023). Today's state-of-the-art question-answering systems exceed human performance on several benchmark datasets (Izacard et al., 2022; Hu et al., 2017; Floridi and Chiriatti, 2020).

**Early work.** The first question-answering systems were developed In 1959 (Simmons, 1965). According to Simmons (1965), these systems were not particularly useful in practical applications due to their inability to accurately process natural language. Instead, these systems could only answer very specific questions in particular formats, such as those related to family kinship (Lindsay, 1963). Numerous earlier systems focused on tasks such as syntactic parsing, part-of-speech tagging, and structural analysis of questions, which involved transforming the question into a data structure that could be utilized to formulate a response (Simmons, 1965). Subsequent research also incorporated techniques that enabled (limited) semantic analysis of the questions (Simmons, 1965) which is a crucial component for any natural language processing (NLP) system.

The field of QA gained traction in 1999 when it received its own track in the Text Retrieval Conference (TREC) (Prager, 2007). The work of Moldovan et al. (2000) was one of the earliest attempts to build question-answering systems that could condition on information from external knowledge bases by combining traditional information retrieval techniques with information extraction. While their system achieved state-of-the-art results at that time, it had several limitations due to the underdeveloped state of many fundamental NLP techniques, such as proper knowledge representation, textual reasoning, capturing of long-distance textual dependencies, semantic analysis, and document indexing (Moldovan et al., 2000). Consequently, the authors had to rely on heuristics, pattern-matching, and rule-based approaches, which lack the ability to generalize well to many possible demands that a QA system may face. Furthermore, during that period, many popular information retrieval (IR) systems (like TF-IDF and BM25), which are crucial components of OpenQA systems, were based on lexical document statistics. These systems are typically incapable of adequately capturing semantic relationships within text (Mishra and Jain, 2016; Dwivedi and Singh, 2013).

**Advent of neural machine reading comprehension.** The field of machine reading comprehension (MRC), which is a crucial component of QA, has developed rapidly with the advancement of machine learning and deep learning (Chen and Yih, 2020). The *transformer* architecture from the paper *Attention is all you need* (Vaswani et al., 2017) introduced novel techniques for processing text data, leading to new state-of-the-art and even super-human results on various NLP tasks including machine translation (Stahlberg, 2020), text classification (Devlin et al., 2018), sentiment analysis (Gramer and Danielsson, 2023), and question answering (Devlin et al., 2018). The transformer architecture allowed for the creation of large language models: foundational models that have been trained on huge amounts of unlabeled text data (Radford et al., 2019) to capture probability distributions over text. Among the most famous examples of LLMs that changed the way NLP is conducted are *BERT* (Devlin et al., 2018), *GPT* (Radford and Narasimhan, 2018), *ELMo* (Peters et al., 2018), and BERT-like architectures like *RoBERTa* (Liu et al., 2019c), *DistilBERT* (Sanh et al., 2019), and *DeBERTa* (He et al., 2020). These LLMs can solve the MRC task in many ways: an early popular approach was to use transfer learning to fine-tune a composite model based on these LLMs on a specific task, for instance, question answering (Devlin et al., 2018).

**Neural open-domain question answering.** Chen et al. (2017) proposed a successful approach for answering open-ended questions using an external knowledge base by combining an IR component, the *Retriever*, with a neural MRC component, the *Reader*. The purpose of the Retriever is to fetch documents from a database that are relevant to the question. The purpose of the Reader is then to formulate an answer given these documents and the question. Yang et al. (2019) combined a sparse TF-IDF-like Retriever with a BERT-based Reader fine-tuned on the popular QA dataset *SQuAD*. The job of the Reader was then to extract the span of the document that is most likely to contain the answer. The authors obtain new state-of-the-art results on SQuAD using this two-component architecture. Lee et al. (2019) argue that sparse retrieval is suboptimal when the questions in the dataset are genuine and were constructed without knowledge of the context containing the answer. Instead, they propose a learnable neural Retriever that utilizes BERT to compare the similarity between a given question and documents in the knowledge base using a dot product in a document-query latent space. The Retriever and Reader are then jointly trained to fetch relevant documents and answer the question. Karpukhin et al. (2020), Reimers and Gurevych (2019) and Izacard et al. (2021), among others, propose novel neural architectures and pre-training schemes to improve these systems, again pushing the state-of-the-art performances on many benchmark datasets. Another key component that allows these kinds of systems to work efficiently is the development of *maximum inner-product search* (MIPS) (Abuzaid et al., 2017) and *approximate nearest neighbor* (ANN) (Keivani et al., 2017) algorithms, for instance, Facebook AI's *FAISS* library Mu et al. (2019).

**Augmenting extractive systems.** Lately, many techniques have been developed to augment these canonical Retriever-Reader pipelines, for instance by document re-ranking (Izacard et al., 2022), generation-augmented retrieval (Mao et al., 2021), and final span-score augmentation (Ma et al., 2022). These techniques aim to improve the performance of these relatively small extractive models that are designed to perform well on one or a few tasks, for instance question answering. This is in contrast to the development of general-purpose multi-task language models, such as OpenAI's *ChatGPT* (Lund and Wang, 2023).

.

**Generative open-domain question answering.** In recent years, the field of NLP has undergone a paradigm shift with the development of generative language models. These models, including OpenAI's *GPT*-suite (Radford and Narasimhan, 2018; Radford et al., 2019; Floridi and Chiriatti, 2020; OpenAI, 2023), Meta's *Atlas* (Izacard et al., 2022), *RAG* (Lewis et al., 2020), *LLaMA* (Touvron et al., 2023), and Stanford's *Alpaca* (Taori et al., 2023), are capable of accurately solving a wide range of NLP tasks using *in-context learning* (Ye et al., 2023). This is a technique for pre-trained decoder-based generative models to learn how to solve a particular task without fine-tuning or updating any model weights. Instead, these models can condition on instructions and examples provided at inference time to identify how to solve the task (Floridi and Chiriatti, 2020). These generative models generally have a much larger capacity than extractive BERT-based models, some having billions or hundreds of billions of parameters (Floridi and Chiriatti, 2020; Touvron et al., 2023; Taori et al., 2023). Thus, one requires adequate computational resources to run these models oneself.

**Applications in the industry.** Many companies have implemented open-domain question-answering technology in whole or in part. Some examples include Salesforce's (Zeng et al., 2020), Baidu's (Bao et al., 2020), and Facebook's (Shuster et al., 2020; Roller et al., 2020) systems. However, what sets each of these systems apart is that they are tailored to the unique requirements and pre-existing infrastructure of their respective companies. Zeng et al. (2020) built a natural language interface to relational databases powered by neural models to enable an end user to ask questions that can possibly be answered using data in a relational database. Shuster et al. (2020) builds a high-performing multi-turn chatbot that is fundamentally based on a Retriever and a generative Reader. Shuster et al. (2020) train their chatbot QA system on particularly curated data and show that it improves the final performance of the system. Bao et al. (2020) incorporate in their choice of system components their demand for multilingual compatibility. It is clear that the particular needs of a company's use case dictate the design of the final OpenQA system.

# Chapter 2

# Question Answering Systems

We begin by giving a brief overview of the concepts required to understand how to build and evaluate open-retrieval question-answering (OpenQA) systems. In Section 2.1, the OpenQA task is explicitly defined. Section 2.2 covers the taxonomy of OpenQA systems and some of the systems' basic properties. In Section 2.3, we introduce the most fundamental components of a canonical OpenQA system. Lastly, Section 2.4 introduces metrics for quantitatively evaluating both these components.

## 2.1 Task definition

Karpukhin et al. (2020) defines the task of an open-retrieval question-answering as follows. Given a factoid question and a knowledge base, present the answer if it exists in the knowledge base; otherwise, indicate that the question is unanswerable. Figure 2.1 shows how such a system should work. An OpenQA system achieves this objective by taking similar steps to what a human would do when solving such a task. Given a knowledge base and the question `What were the total assets of Example AB as of December 31, 2021?`, one would likely answer it by taking the following steps.

1. Search a knowledge base, such as a collection of annual reports or the web, for documents containing information on the company. This results in multiple (hopefully relevant) candidate documents that might contain the answer.

2. Assess whether the retrieved documents contain the answer using one's natural language understanding. This involves reading the documents and either identifying the specific section that contains the answer (i.e., *extracting* a span from the passage) or *generating* an answer based on the information within the documents.

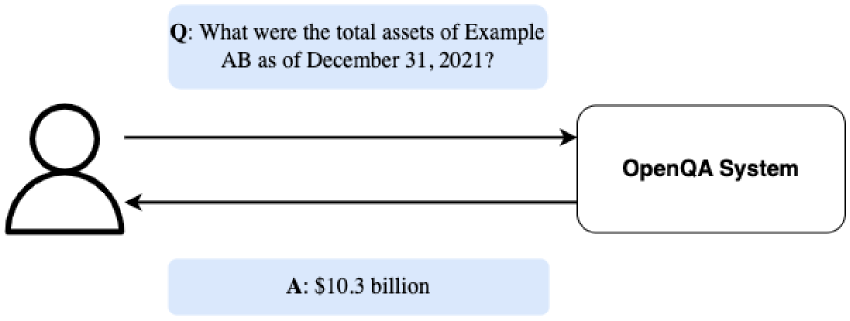

Q: What were the total assets of Example AB as of December 31, 2021?
OpenQA System
A: $10.3 billion

**Figure 2.1:** A typical interaction between an end user and an open-retrieval question answering system.

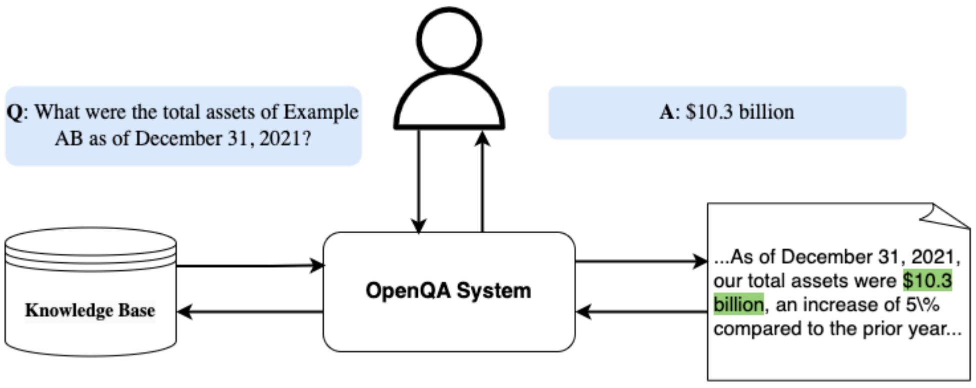

Q: What were the total assets of Example AB as of December 31, 2021?
A: $10.3 billion
Knowledge Base
OpenQA System
...As of December 31, 2021, our total assets were $10.3 billion, an increase of 5\% compared to the prior year...

**Figure 2.2:** An overview of the data flow in an open-retrieval question answering system.

3. Report the best possible answer back to the user.

A functioning OpenQA system should be able to complete the same steps. Figure 2.2 shows an example of the OpenQA process for finding the answer. Note that it is a design choice whether the final answer is to be (a) a span from one of the passages, or (b) a generated sentence based on the retrieved passages. This, among other things, can be encapsulated in the taxonomy of QA systems.

## 2.2 The taxonomy of QA systems

The landscape of question-answering systems is broad: there is a significant difference between systems such as ChatGPT (being able to answer questions in many languages in free-form, without a connection to an external database), Google Search's "Feature Snippet", and simple systems extracting a span of text from a user-supplied context. This breadth calls for a characterization of the typical properties of systems capable of answering questions. We describe two of these in the following.

**Closed vs open retrieval systems.** An *open-retrieval* (or *open-book*) QA system utilizes a vast array of external knowledge sources to provide answers to user queries (Ciosici et al., 2021). In contrast, a *closed-retrieval* (or *closed-book*) system lacks access to an external knowledge base and instead solely relies on the learned internal language representation stored within the model's weights (Roberts et al., 2020).

Open-book systems have an advantage in answering niche questions if the answer exists in their knowledge base, which can store vast amounts of specific information (Ciosici et al., 2021). However, these systems may encounter challenges if the answer does not exist in the knowledge base. In contrast, closed-book systems rely on limited knowledge acquired from their training data, making them valuable when external knowledge sources are not accessible (Roberts et al., 2020). OpenAI's ChatGPT, a closed-book system, is a well-known example that can answer a wide range of questions but may struggle with recent events that occurred after its training data was collected (Lund and Wang, 2023).

**Factoid versus non-factoid systems.** Question-answering systems can be classified based on the type of answers they provide to the user. *Factoid* (or *extractive*) systems give a text span from a context as the answer to a given question, allowing the source document to be traced back exactly (Pearce et al., 2021). To ensure answer uniqueness, the *minimum* span containing the answer should be provided (Pearce et al., 2021). Factoid questions typically involve inquiries about named entities, resulting in concise responses (Surdeanu et al., 2008). In contrast, *non-factoid* systems typically answer questions by synthesizing information across multiple documents, typically resulting in longer and more subjective answers (Surdeanu et al., 2008). Surdeanu et al. (2008) provide a typical example of non-factoid questions and answers: those originating from question-answering websites as *Yahoo! Answers* where the best answer is often generated from a combination of subsets of answers provided by several users in a single or across multiple discussion threads.

## 2.3 Components of OpenQA Systems

A system capable of answering questions in an open-retrieval fashion requires additional components compared to a closed-domain questions-answering system (Qu et al., 2020). In particular, it requires combining elements from information retrieval (IR) and machine reading comprehension (MRC) systems. Chen et al. (2017) describe how OpenQA systems usually consist of three main components:

1. A *document store* storing a large collection of documents. The documents are often stored as a vector representation in an efficient data structure, allowing for efficient document search.

2. A *Retriever* fetching $\text{top}_{k,\text{Retriever}} \geq 1$ documents that are relevant to the query. We aim to find the answer in these documents.

3. A *Reader* component that phrases $\text{top}_{k,\text{Reader}} \geq 1$ answers based on the relevant documents and the query. These answers might be spans from the retrieved passages, or generated sentences, depending on the type of system.

Figure 2.3 shows the interaction between the components. In the following, we go more in-depth into these three key modules. Section 2.3.1 introduces the document store, and Section 2.3.2 briefly covers the Retriever component. Lastly, Section 2.3.3 introduces the Reader component.

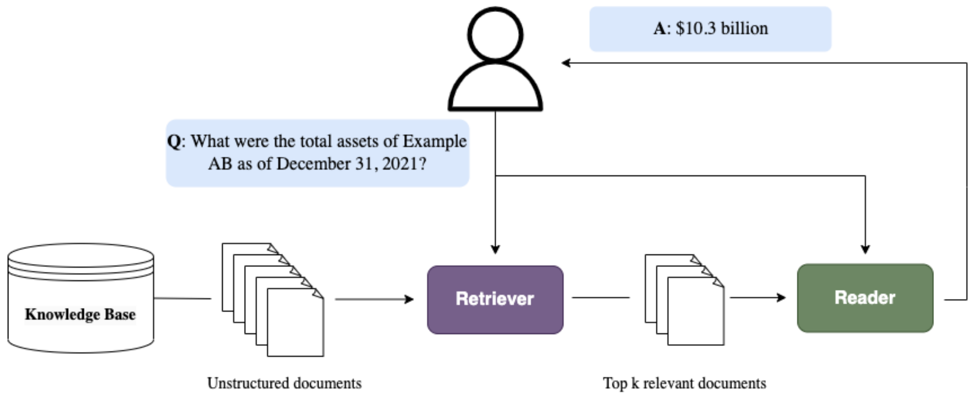


**Figure 2.3:** The data flow in a typical Retriever-Reader OpenQA pipeline.

## 2.3.1 Document store

The document store, or *knowledge base*, serves two purposes: It stores a collection of semi-structured documents and enables efficient document search (Kononenko et al., 2014). In essence, it functions like a document-oriented database, equipped with vector search algorithms that allow for search among the representations of the documents. The database stores document contents and metadata (for instance, IDs and vector representations of the documents) in some efficient data structure (Mu et al., 2019).

## 2.3.2 Retriever

The *Retriever* component fetches relevant documents to a given query. Due to the computational cost of running a Reader on many long texts (Devlin et al., 2018), explained in Chapter 4, a smaller set of $\text{top}_{k,\text{Retriever}}$ relevant documents is typically first fetched from the collection of documents in the knowledge base (Chen et al., 2017; Karpukhin et al., 2020). When using a large external knowledge base to answer a question, it makes sense to initially retrieve a small set of documents before actually reading them. This approach is similar to how humans would approach the task, as they would only read documents that they believe could contain the answer, rather than attempting to read every document in the knowledge base. The Retriever performs document retrieval by calculating the similarity between a given query and the documents in a vector space representation. It measures the similarity by computing metrics such as cosine similarity, which allows the Retriever to identify the most relevant documents based on their proximity to the query in the vector space. Section 4.3 provides an in-depth explanation of this process.

## 2.3.3 Reader

The *Reader* component in an OpenQA system extracts the answer from a (small) unordered set of documents. Essentially, the Reader tackles the challenge of reading comprehension. There are various methods to accomplish this task, including rule-based pattern matching, span extraction, and generative techniques. In this explanation, we will explore two approaches to solve this task: *extractive* and *generative* Readers. These two approaches create a distinction between different types of systems, as illustrated in Figure 2.4.

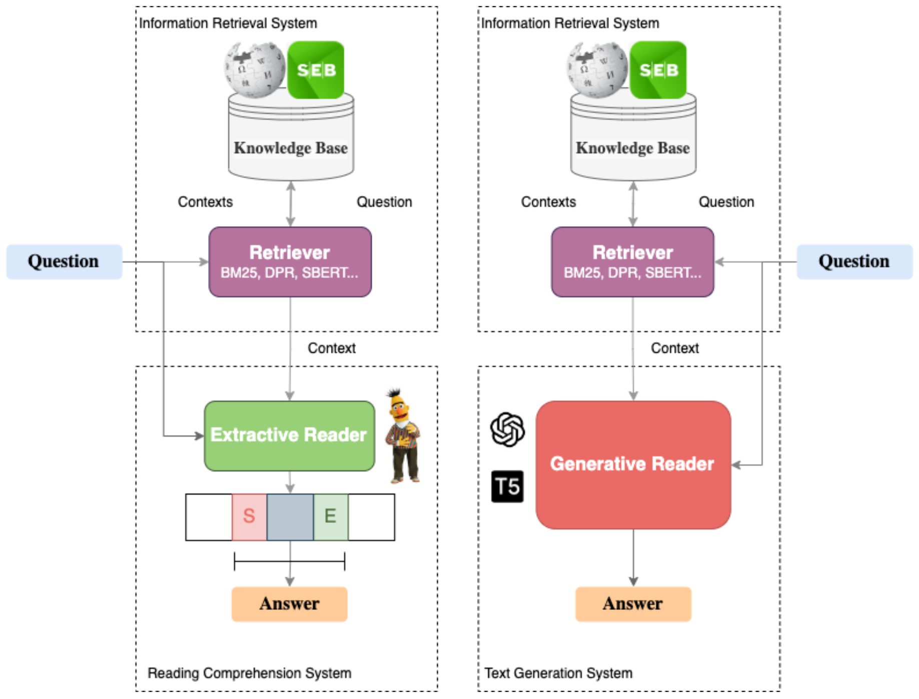


**Figure 2.4:** Two types of Retriever-Reader QA pipelines. Both use a Retriever to search for relevant documents, which are subsequently fed to either an extractive Reader (to the left), or a generative Reader (to the right).

**Extracting answers with an extractive Reader.** The *extractive* Reader, as described by Liu et al. (2019b), predicts the location of the answer within a given context. This is accomplished by identifying the smallest possible subsequence of the context that is likely to contain the answer, based on the given question. According to the authors, this approach enables more versatile machine reading comprehension (MRC) models compared to traditional models that are typically used for tasks such as multiple-choice or single-word answer questions. Extractive Readers that use span extraction can answer questions using multiple consecutive sentences and without relying on user-supplied answer choices.

The Reader can be viewed as a simple classifier, predicting the most probable start and end token of the answer in the context. The number of potential classes is equal to the max input length as allowed by the Reader model used. To tailor the answer extraction to the question, one commonly concatenates the question and the context, separated by a `[SEP]` token, and prepend a `[CLS]` token (Izacard et al., 2022). When an answer cannot be found, we predict the start and end token of the answer to coincide, both as the `[CLS]` token. We also consider an answer missing if the predicted end token precedes the predicted start token, or vice versa. Multiple answers can be the output of the Reader if one sorts the predicted spans probabilities with respect to decreasing sum of start and end probabilities (Devlin et al., 2018), and returns the best $\text{top}_{k,\text{Reader}} \geq 1$ spans. Figure 2.5 shows the answer span scoring using the example defined in Section 2.1. The output probabilities of the Reader serve as an inherent scoring mechanism for candidate spans. There are multiple scoring schemes available, however, the sum of start and end probabilities is a popular scoring choice. In Section 4.4.1, we will go into more detail on how these types of extractive Readers work.

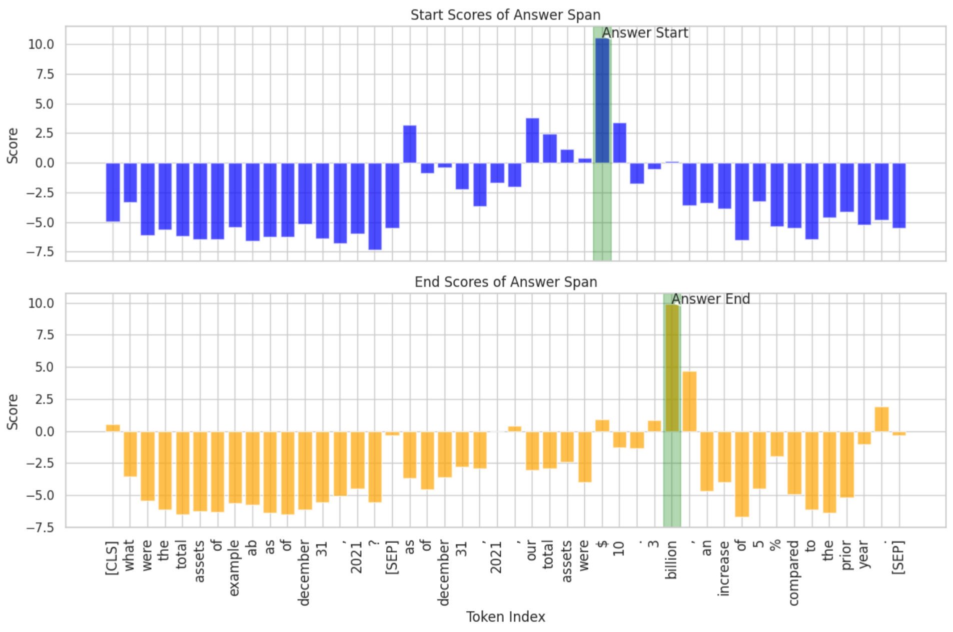


**Figure 2.5:** Start and end score (logit) distribution for the example question: `What were the total assets of Example AB as of December 31, 2021?`. This is the output of the extractive Reader. The top figure shows the logits for the predicted start token; the bottom figure for the end token. Answer spans are chosen in decreasing order with respect to the sum of start and end (normalized) logits.

**Generating answers using generative Reader.** Another approach to answering a question is generating the answer, given the query and some candidate passages. A *generative* Reader completes this task. Recently, lots of research has been conducted on generative machine learning models (see, for instance, Izacard et al. (2022), Lewis et al. (2020), and Gm et al. (2020)), many of which hold state-of-the-art results in many question-answering tasks. These models can aggregate information across multiple documents and generate answers that are not explicitly stated in any of the documents (Lewis et al., 2020). Generative models are generally split into two types: *encoder-decoder* (or seq-2-seq) models and *decoder* (or autoregressive) models. How these models work will be covered in more detail in Section 4.4.2.

## 2.4 Evaluation of OpenQA Systems

To design a functioning OpenQA system, one needs to be able to quantitatively measure the performance of different configurations. If the system contains multiple modules, it is also crucial to be able to separately evaluate each component in the pipeline to identify where problems or bottlenecks arise. We describe this evaluation in the following sections. Section 2.4.1 introduces some terminology related to ground-truth labels in the OpenQA setting. Then, Sections 2.4.3 and 2.4.2 outlines standard practices for evaluating the Retriever and Reader systems, respectively.

### 2.4.1 Terminology

In the evaluation of OpenQA systems, we begin by defining fundamental terms such as *gold answer*, *gold document*, and *answer scope*. These terms establish a shared understanding and framework for assessing system performance.

**Gold answers and documents.** To facilitate the evaluation of an OpenQA system, we first have to clearly state the desired behavior of the system for a specific dataset. When evaluating a Reader for a set of queries, we need predefined ground-truth answers for these queries, dictating what answers we want the Reader to return. Using the terminology from Farea et al. (2022), these answers are often called *gold answers*. For the Retriever, when given a query, we have to specify, ahead of time which document is the desired one for the Retriever to fetch. These are called *gold documents* or *gold passages*. Multiple gold documents, and answers, may exist (Farea et al., 2022). In our setting, the gold answer will always be a subsequence of the gold document, given that the gold answer exists. It is often the purpose of the training and evaluation sets to provide examples of tuples (`query`, `gold document(s)`, `gold answer(s)`), as we will see in Chapter 3.

**Answer scope.** The presence of gold answers across multiple documents means that it is sufficient for an OpenQA system to have the gold answer within the gold document. Consequently, even if the Retriever retrieves a non-gold document, the Reader may still be capable of extracting the gold answer from it, provided it exists within that document. It is up to the evaluator of the systems to determine the *scope* of evaluation: one might want to consider a predicted answer to be correct only if both (1) the answer predicted by the Reader matches the gold answer and (2) the document fetched by the Retriever matches the gold document. Another alternative is to loosen the restriction on the Retriever and consider a predicted answer span correct only if the predicted answer matches the gold answer (and thus remove point (2) above).

### 2.4.2 Evaluating the Retriever

The metrics for evaluating a Retriever in an OpenQA setting are similar to those traditionally used in machine learning. Here, they are derived by comparing the documents fetched by the Retriever and the gold document(s). The main difference between traditional metrics is that we now have to account for the fact that the Retriever may fetch multiple documents, as dictated by the $\text{top}_{k,\text{Retriever}}$ hyperparameter. In the following, we describe the so-called *recall@k* and *precision@k*. While several metrics exist concerned with evaluating the *order* in which retrieved documents are ranked (Järvelin and Kekäläinen, 2002), these are not interesting in the OpenQA setting as the Reader treats incoming documents as an unordered set.

**Recall@$k$.** Recall@$k$ specifies how well the system finds relevant documents in its knowledge base, given a specific question (Zuva and Zuva, 2012). It compares the number of relevant documents retrieved to the number of relevant documents that exist in the knowledge base. The way it differs from traditional recall is as follows. A Retriever that fetches more documents naturally has a higher likelihood of including relevant documents, and vice versa. Therefore, its recall is likely to increase if we allow more documents to be retrieved. It is thus reasonable to include $\text{top}_{k,\text{Retriever}}$ in the typical recall metric. Zuva and Zuva (2012) give us the formal definition

$$\text{Recall@}k = \frac{|\{\text{ relevant documents }\} \cap \{\text{ top}_{k,\text{Retriever}} \text{ retrieved documents }\}|}{|\{\text{ relevant documents }\}|}. \tag{2.1}$$

**Precision@$k$.** Precision@$k$ quantifies the systems' ability to present documents that are relevant (Zuva and Zuva, 2012). The metric tells you what fraction of the retrieved documents are indeed relevant. Again, this depends on how many documents you fetch: if 100 documents are retrieved, and you know only one relevant document exists, you cannot hope to get a precision better than 0.01. Formally, (Zuva and Zuva, 2012) define

$$\text{Precision@}k = \frac{|\{\text{ relevant documents }\} \cap \{\text{top}_{k,\text{Retriever}} \text{ retrieved documents }\}|}{\text{top}_{k,\text{Retriever}}}. \tag{2.2}$$

Although it is beneficial to keep both recall@k and precision@k high, Lampert (2004) claims that recall@$k$ is more interesting than precision@$k$ in the OpenQA setting. This priority is due to the following; to answer a question the most important factor is to have the relevant documents retrieved and available. This, of course, relies on the assumption that a downstream component is capable of in some way discerning between relevant and irrelevant documents, and not being overflooded by irrelevant information. Furthermore, for clarity, we wish to emphasize that a document is deemed relevant if it is a gold document: the names "relevant" and "gold" are thus interchangeable. In the remaining of the paper, recall@$k$ and precision@$k$ will be simply referred to as *recall* and *precision* as we always talk about retrieval with respect to the $\text{top}_{k,\text{Retriever}}$ hyperparameter.

### 2.4.3 Evaluating the Reader

There are several metrics for evaluating the Reader. Evaluation is normally done by comparing the inferred answers from the model to the gold answer in some way. Generally, they are divided into

| Question | Inferred Answer | Gold Answer | F1 | EM |
|---|---|---|---|---|
| "When did Beyonce start becoming popular?" | "1990s" | "In the late 1990s" | 0.4 | 0 |
| "In what city and state did Beyonce grow up?" | "Houston, Texas" | "Houston, Texas" | 1 | 1 |

**Table 2.1:** QA Evaluation metrics example

two categories: lexical-based and semantic-based similarity scores. The lexical-based metrics are *F1-score* and *exact match* (EM). Both metrics estimate the quality of a QA system by computing the lexical overlap between the inferred answer with the gold answer (Farea et al., 2022). Furthermore, a common semantic-based metric is the *semantic answer similarity* (SAS) (Chandrasekaran and Mago, 2021). These are briefly introduced in the following.

**Exact match.** The *exact match* (EM) score, as suggested by the name, is a binary metric that evaluates whether the output answer is identical to the gold answer, i.e. if there is an *exact match* between these. It takes the value of one given that the answers exactly match and zero otherwise (Farea et al., 2022). Table 2.1 shows an example of how the EM-score is computed.

**F1-score.** Compared to EM, the F1-score provides a more flexible estimate of the lexical overlap between an inferred answer and a gold answer. It evaluates the similarity between the inferred and gold answers by computing the harmonic mean between the precision and recall computed on a token level. In the Reader setting, a true positive (TP) refers to the tokens that are common in the inferred answer and the gold answer. A false positive (FP) would be a token existing in the inferred answer and not in the gold answer (Farea et al., 2022). In the Reader setting, precision can be defined as

$$\text{Precision} = \frac{|\{\text{ gold tokens }\} \cap \{\text{ predicted tokens }\}|}{|\{\text{ predicted tokens }\}|}, \tag{2.3}$$

and recall as

$$\text{Recall} = \frac{|\{\text{ gold tokens }\} \cap \{\text{ predicted tokens }\}|}{|\{\text{ gold tokens }\}|}. \tag{2.4}$$

This enables us to define the F1-score as the harmonic mean of the precision and recall; formally

$$\text{F1} = 2\frac{\text{Precision} \cdot \text{Recall}}{\text{Precision} + \text{Recall}}. \tag{2.5}$$

Table 2.1 shows examples of F1 scores for two samples from the SQuADv2 dataset (covered in Chapter 3).

**Semantic answer similarity.** The previous metrics for QA models were based on lexical comparisons between the model output answer and the label. Predicted answers that have more words overlapping with the ground-truth answer is scored better. However, a simple lexical comparison might be overly pessimistic; answers that capture the same meaning as the true answer, but using different words, might still be an acceptable answer. As the name suggests, *semantic answer similarity* (SAS) is a metric that takes a different approach and instead evaluates the QA model output in terms of the semantic similarity between the predicted answer and the label, as proposed by Risch et al. (2021). The authors does this by fine-tuning a pre-trained cross-encoder architecture on a semantic textual similarity (STS) task (Agirre et al., 2013); the predicted and gold answers are concatenated and passed through a neural model, outputting a similarity score. Further details on how such a model is built is covered in Section 4.3.2. According to Risch et al. (2021), the SAS scores correlate better with human-judgment-based similarity scores than the aforementioned lexical evaluation metrics.

# Chapter 3

# Datasets

We use four different data sets to design and evaluate our OpenQA system. Three of these datasets are benchmark datasets common in the open-domain question-answering literature, and one is a private dataset curated by ourselves. In Section 3.1 we present *SEBQuAD*, an application-specific private question-answering dataset curated by SEB and ourselves. Section 3.2 introduces the Stanford Question Answering Dataset (SQuAD). In Section 3.3, we present Google's open-domain QA dataset, *Natural Questions*. Finally, in Section 3.4, we outline the *TriviaQA* dataset.

## 3.1 *SEBQuAD*

*SEB Question Answering Dataset* (SEBQuAD) is a small question-answering dataset used for evaluating an OpenQA system in a banking setting, curated by SEB and ourselves. The corpus consists of internal policy documents (e.g. the SEB code of conduct) and external documents (e.g. press releases and financial statements). We have annotated the dataset with a variety of questions and answers, covering topics related to banking, account management, investments, and regulatory compliance. Note that SEBQuAD contains proprietary data from SEB and thus cannot be shared publicly. However, we provide dataset statistics and a synthetic example of the sample format of the SEBQuAD dataset.

**Dataset Statistics.** The SEBQuAD dataset contains 15 long documents annotated with 111 question-answer pairs. Table 3.1 provides an overview of the number of documents, questions, and answers in the SEBQuAD dataset. Figure 3.2 shows the distribution of question types in the dataset, as determined by the questions' initial word.

**Sample Format.** SEBQuAD follows the SQuAD format, described in more detail in Section 3.2, where each document is associated with a set of questions and their respective answers explicitly

| | |
|---|---|
| # Documents | 15 |
| # Questions | 111 |
| # Answers | 111 |
| Average Document length | 3725 words |
| Average Question length | 11 words |
| Average Answer length | 6 words |

**Table 3.1:** SEBQuAD dataset statistics.

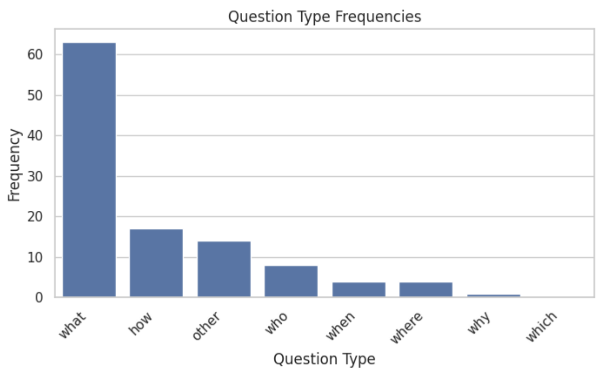


**Table 3.2:** Frequency of question types in the SEBQuAD Dataset

stated in the document (Rajpurkar et al., 2018). Many documents in the SEBQuAD dataset contain tabular data, which is difficult for classical QA systems to parse. While techniques exist to address this issue, as suggested by Jin et al. (2022), they are not within the scope of this project. Instead, we have removed all tabular data from these documents to produce a cleaner dataset. As SEBQuAD contains proprietary data from SEB, we are unable to share the actual dataset with the public. However, Table 3.3 shows some synthetic examples of the samples in the dataset. We believe that these examples are representative of the types of questions and answers in the dataset.

## 3.2 *SQuAD*

Rajpurkar et al. (2018) present the *Stanford Question Answering Dataset* (SQuAD), a ubiquitous question-answering dataset that serves as a common benchmark for QA systems. Each example consists of three components: a context paragraph, a question, and an answer (extracted from the context). The task is to extract the minimal span of the context containing the answer, given the questions and the context. A few different versions of SQuAD exist. Most notable is the second version, SQuADv2, which additionally contains unanswerable questions. In the rest of the paper, we will for brevity refer to SQuADv2 as *SQuAD*.

**Dataset statistics.** SQuAD consists of in total 130 319 training examples (out of which about one-third contain unanswerable questions), and 11 873 development examples (out of which about 50% contain unanswerable questions) (Rajpurkar et al., 2018). We will use the development set for evaluation. Table 3.4 shows dataset statistics for this subset of SQuAD. The dataset includes a variety of question types; Figure 3.5 shows the distribution of question types as determined by the questions' initial word.

**Sample Format.** SQuAD is widely regarded to be the MNIST of question answering and thus provides the standard format for most QA datasets. The SQuAD dataset is organized as a collection of JSON files, each containing a set of question-answer pairs and their associated contexts (Rajpurkar et al., 2018). Table 3.6 shows a sample from the SQuAD development set.

| Truncated Context | Question | Answer |
|---|---|---|
| ...As of December 31, 2021, our total assets were $**10.3 billion**, an increase of 5% compared to the prior year. | What were the total assets of the company as of December 31, 2021? | $10.3 billion |
| ...As part of our customer support policy, we strive to provide timely and effective resolution to any issues or concerns our customers may have. Our support team is available 24/7 via **phone, email, or live chat**. | What channels are available for customers to contact SEB's support team? | phone, email, or live chat. |
| ...SEB's travel expense policy outlines the guidelines and procedures for employees who incur expenses while on business trips. This includes **rules for transportation, lodging, meals, and incidentals**. | What does SEB's travel expense policy cover? | rules for transportation, lodging, meals, and incidentals. |

**Table 3.3:** Sample format of SEBQuAD dataset using synthetic, made-up examples.

| | |
|---|---|
| # Documents | 1204 |
| # Questions | 11873 |
| # Answers | 5928 |
| Average Document length | 128.8 words |
| Average Question length | 2.4 words |
| Average Answer length | 10.2 words |

**Table 3.4:** SQuAD dataset statistics.

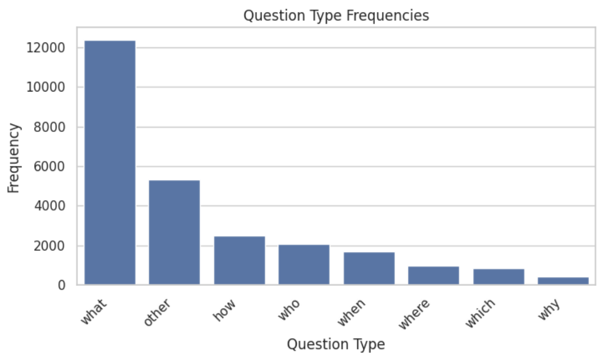


**Table 3.5:** Frequency of question types in the SQuAD Dataset

| **Truncated Context** | **Question** | **Answer** |
|---|---|---|
| ...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... | To whom did the Virgin Mary allegedly appear in 1858 in Lourdes France? | Saint Bernadette Soubirous |

**Table 3.6:** Sample format of SQuAD using examples from the dev set. The answer extracted from the (truncated) context is marked in bold font.

**SQuAD Open.** While SQuAD dataset was originally intended for closed-domain reading comprehension, we will use it in the open-domain setting. This is done by simply viewing every "context" entry as a document that is to be retrieved from our knowledge base. This modification is standard in the literature and the resulting data set is sometimes referred to as *SQuAD Open* (Izacard and Grave, 2020).

## 3.3 *Natural Questions*

The *Natural Questions* (NQ) dataset is a large open domain question-answering dataset, presented in 2019 by Google Research (Kwiatkowski et al., 2019). The authors define two different tasks for the dataset. The first task is to – given a question – extract a *short answer* from a passage from a Wikipedia article. The second task is to – given a question – extract a *long answer* from a Wikipedia article. At your disposal is the entirety of Wikipedia (particularly, a dump from December 20, 2018). The short answer task is very similar to the SQuAD Open problem proposed by (Izacard and Grave, 2020). The long answer task is more of a pure information retrieval problem where the system is to return an entire paragraph that contains the answer. In order to better fit the banking application we evaluate our openQA system on the short answer task.

**Dataset Statistics.** Natural questions consists of 307,373 training samples, 7,830 validation samples, and 7,842 test samples. 49% of the examples in the dataset have long answers only, and 36% have both long and short answers. The questions without answers are not discarded in the final dataset as the authors "consider the choice of whether or not to answer a question a core part of the question answering task" (Kwiatkowski et al., 2019, p. 457). We will evaluate our system on the Natural Questions validation set. Table 3.7 shows dataset statistics of the validation set. Figure 3.8 shows the distribution of question types in the validation set. Comparing Figure 3.8 with Figure 3.5, one may note that the `other` category is more prevalent in Natural Questions compared to SQuAD. NQ contains a larger proportion of questions that are phrased in a non-standard way compared to the SQuAD dataset. Table 3.10 shows an example of such a non-standard question – which is more of a statement rather than a question. These types of questions have emerged due

| | |
|---|---|
| # Documents | 108032 |
| # Questions | 7830 |
| # Answers | 2819 |
| Average Document length | 244 words |
| Average Question length | 9.3 words |
| Average Answer length | 4.2 words |

**Table 3.7:** Natural Questions dataset statistics.

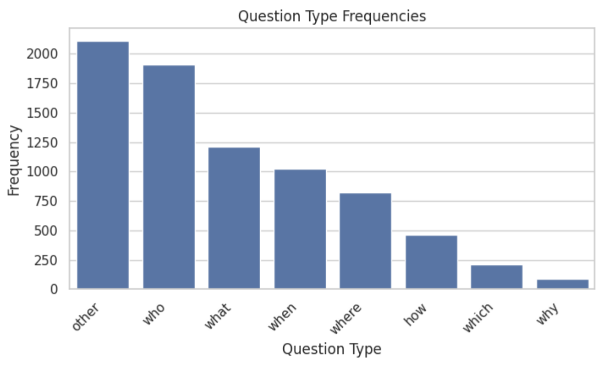


**Table 3.8:** Frequency of question types in the Natural Questions dataset.

to the fact that the questions in NQ were collected from real-world Google searches Kwiatkowski et al. (2019), as opposed to having been made up by an annotator as is the case for SQuAD. The authors suggest that this makes for a more challenging NLP task.

**Sample Format.** Kwiatkowski et al. (2019) describes the structure of each example in the dataset. Each example consists of a tuple (`question`, `Wikipedia page`, `long answer`, `short answer`). If the answer is not to be found in this gold document, both the long and short answer is `null`. Table 3.9 shows an example of a raw sample from NQ. The supplied Wikipedia article is to be considered the gold document. It is in raw HTML, but can be cleaned into pure natural language tokens using a conversion script[1] provided by the authors. This same script results in a SQuAD-formatted file that is cleaned of HTML tags and UTF-8 encoded symbols. Table 3.10 shows an example of such a cleaned sample.

## 3.4 *TriviaQA*

*TriviaQA* (Joshi et al., 2017) is an open-retrieval question answering dataset containing 95,000 question-answer pairs. According to the authors, the dataset comes in two forms *Wikipedia* and *Web*, depending on whether the contexts originate from Wikipedia or have been scraped from the internet. There are also *verified* subsets of each of the two forms for which all of the data points have been manually verified to correctly answer the question using the evidence provided (Joshi et al., 2017).

The authors show that the questions of the dataset require more complex strategies to be answered compared to similar datasets such as SQuAD. For instance the contexts from which the answers can be extracted are collected independently from the compilation of the questions, hoping to reduce bias in the style of questions being asked. Additionally, Joshi et al. (2017) claims that questions and answers have more lexical and syntactic variations between them than in previous datasets, requiring comprehension models that better understand semantic structure.

[1]`https://github.com/google/retrieval-qa-eval/blob/master/nq_to_squad.py`

| Example from Natural Questions | |
|---|---|
| **Question** | what do the 3 dots mean in math |
| **Context (trunc.)** | “Therefore sign - wikipedia <H1> Therefore sign </H1> Jump to : navigation, search <Table> <Tr> <Th_colspan="2"> ∴ </Th> </Tr> <Tr> <Td_colspan="2"> Therefore sign </Td> </Tr> <Tr> <Td_colspan="2"> <Table> <Tr> <Th_colspan="2"> Punctuation </Th>..." (+10 000 additional characters) |
| **Long Answer** | "<P> In logical argument and mathematical proof , the therefore sign ( ∴ ) is generally used before a logical consequence , such as the conclusion of a syllogism . The symbol consists of three dots placed in an upright triangle and is read therefore . It is encoded at U + 2234 ∴ therefore ( HTML & # 8756 ; &there4 ; ) . For common use in Microsoft Office hold the ALT key and type “ 8756 ” . While it is not generally used in formal writing , it is used in mathematics and shorthand . It is complementary to U + 2235 ∵ because ( HTML & # 8757 ; ) . </P>” |
| **Short answer** | “the therefore sign ( ∴ ) is generally used before a logical consequence , such as the conclusion of a syllogism” |

**Table 3.9:** An example from the Natural Questions dataset.

**Dataset Statistics.** The full TriviaQA dataset contains 95,000 question-answer pairs each with, on average, 6 evidence documents in which the answer can be found, resulting in a collection of about 650 000 documents. Due to hardware constraints, we only use a subset of 40,000 documents in our knowledge base. The verified Wikipedia and verified Web dataset contains only 297 and 322 labels, although of high quality compared to the rest of the dataset. Tables 3.11 and 3.13 shows an overview of the dataset statistics for the verified Wikipedia and the verified Web datasets, respectively. Similar to the previously mentioned dataset, both the verified Web and Wikipedia versions of TriviaQA are comprised of a variety of question types. Figures 3.12 and 3.14 shows distribution of question types for the Wikipedia and Web splits, respectively. TriviaQA also contain more questions that are phrased in a non-standard way compared to SQuAD. Moreover, TriviaQA have about 3 times as many questions (compared to SQuAD) that require the combination of information across multiple sentences to answer a particular question (Joshi et al., 2017).

**Sample format.** Joshi et al. (2017) provide a conversion script [2] that cleans and converts the TriviaQA files to the SQuAD-format (outlined in Section 3.2). Table 3.15 shows an example of such a cleaned data point, extracted from the verified Wikipedia subset.

**Question Types and Complexity.** TriviaQA include many types of questions which can be characterized in other ways than by their initial word Joshi et al. (2017). This contributes to making the questions more difficult to answer. These include fine-grained questions (“What **fragrant essential oil** is obtained from Damask Rose?”), coarse-grained questions (“**Who** won the Nobel Peace Prize in 2009?”), and comparisons (“What is the appropriate name of the **largest** type of frog?”). Table 3.16 shows two additional examples from TriviaQA where this complexity is manifested.

[2] `https://github.com/mandarjoshi90/triviaqa/blob/master/utils/convert_to_squad_format.py`

| **Truncated Context** | **Question** | **Answer** |
|---|---|---|
| ...The Chandrabhaga or Chenab (Vedic name Askni), the largest river (in terms of volume of water) is formed after the meeting of two streams namely, Chandra and Bhaga at **Tandi, in Lahaul**. It flows 122 kilometers (76 mi) and covers an area of 7,500 square kilometers (2,900 sq mi). in Himachal, before entering Kashmir. The Chandra passes through the barren tribal land... | chandra and bhaga river meets at the place | Tandi, in Lahaul |

**Table 3.10:** Sample format of the SQuAD-formatted version of NQ. The answer extracted from the (truncated) context is marked in bold font.

| | |
|---|---|
| # Documents | 40000 |
| # Questions | 297 |
| # Answers | 297 |
| Average Document length | 5845 words |
| Average Question length | 14.2 words |
| Average Answer length | 1.8 words |

**Table 3.11:** TriviaQA wikipedia verified dataset statistics.

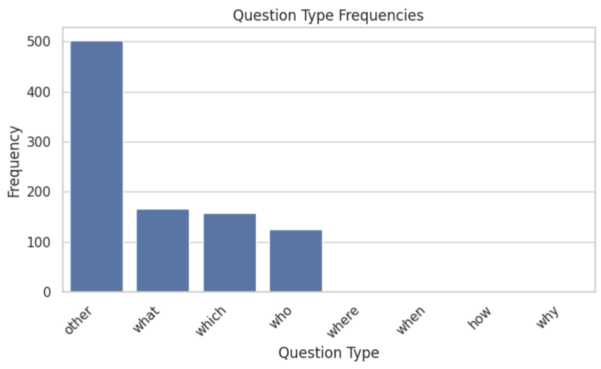


**Table 3.12:** Frequency of question types in the TriviaQA wikipedia verified dataset

| | |
|---|---|
| # Documents | 40000 |
| # Questions | 305 |
| # Answers | 305 |
| Average Document length | 3138 words |
| Average Question length | 13.3 words |
| Average Answer length | 1.8 words |

**Table 3.13:** TriviaQA web verified dataset statistics.

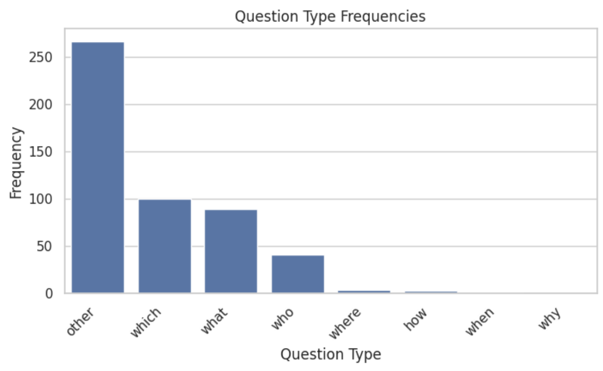


**Table 3.14:** Frequency of question types in the TriviaQA web verified dataset.

| **Truncated Context** | **Question** | **Answer** |
|---|---|---|
| ...Rachel Karen Green is a fictional character , one of the six main characters who appear in the American sitcom Friends . Portrayed by actress **Jennifer Aniston** , the character was created by show creators David Crane and Marta Kauffman , and appeared in each of the show ' s 236 episodes during its decade-long run , from its premiere on September 22 , 1994 to its finale on May 6 , 2004 ... | Who played Rachel Green in Friends? | Jennifer Aniston |

**Table 3.15:** Sample format of the SQuAD-formatted TriviaQA. The answer extracted from the (truncated) context is marked in bold font.

| **Lexical variation** | |
|---|---|
| Question | What is solid CO2 <u>commonly called</u>? |
| Answer | The frozen solid form of CO2, <u>known as</u> **dry ice** ... |
| **Answer in multiple sentences** | |
| Question | Who starred in and directed the 1993 film A Bronx Tale? |
| Answer | **Robert De Niro** To Make His Broadway Directorial Debut With A Bronx Tale: The Musical. The actor starred and directed the 1993 film. |

**Table 3.16:** Example of data points in TriviaQA, displaying two types of complexities contained in the question-answer pairs. Answers are in bold font; lexical variations are underlined.

# Chapter 4

# Architectures

In this chapter, we provide a technical, yet high-level, description of the components in an OpenQA system. We begin in Section 4.1 by giving some notes on recent advances in natural language processing (NLP) that are fundamental in an OpenQA system. Second, in Section 4.2, we outline implementations of the document store. Finally, in Sections 4.3 and 4.4, we describe the architectures of the Retriever and Reader, respectively.

## 4.1 Modern Natural Language Processing

We first provide a brief primer on a few key concepts in modern NLP in order to understand the reasoning behind the architectural decisions made in QA systems. Section 4.1.1 introduces the concept of mathematically representing text, to enable machines to process them. Many of the best QA systems have at least one component based on the *transformer* architecture (Zhu et al., 2021). Consequently, Section 4.1.2 provides a brief description of transformers. Finally, Section 4.1.3 outlines the concept of *transfer learning*, a practical implication of using transformer-based architectures.

### 4.1.1 Numerical Representations of Text

A computer can not read in the way that a human does and must instead rely on mathematical or numerical representations of text. There are multiple ways of mapping words to some mathematical object with the goal of enabling a computer to perform operations on them. Naively, we could represent a document as a collection (or a *bag*) of its words: a document thus becomes a point in a vector space where each axis represent a word in the vocabulary, and where the coordinates correspond to the number of times the words occur in the document (Harris, 1954). This yields a simple and sparse numerical representation of the text. One could improve this representation

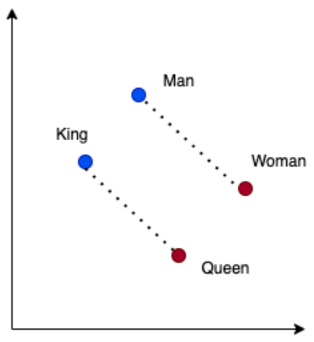


**Figure 4.1:** Simplified illustration of word embeddings

for instance by weighting the coordinates based on the importance of the words in the document using term frequency-inverse document frequency (TF-IDF) (Sparck Jones, 1972) or some other weighting scheme.

**Word embeddings.** A limitation of the bag-of-word representations is that they do not encapsulate the semantic relation between words. Nowadays it is more popular to represent words as low-dimensional dense vectors, often referred to as *word embeddings* (Pennington et al., 2014). These representations are often constructed in a way such that vector space operations between word vectors agree with the semantic relationships between the corresponding words (Pennington et al., 2014). Figure 4.1 shows an example of such meaning-preserving operations. Here, we have four words: `king`, `queen`, `man`, and `woman`, marked as points in a two-dimensional vector space. If these word embeddings are well-constructed we should be able to compute $\texttt{king} - \texttt{man} + \texttt{woman}$ and receive `queen` as a result. This simplified example shows how a vector space model of language is useful in enabling a computer to process natural language by capturing semantic relationships in vector arithmetic.

There are many ways of constructing text embeddings; some early successful ones are *word2vec* (Mikolov et al., 2013) and *GloVe* (Pennington et al., 2014) which utilize document statistics to construct meaningful vector representations. With the advent of the transformer architecture (Vaswani et al., 2017), the embeddings are nowadays often created by some large language model encoder – see for instance Devlin et al. (2018) or Peters et al. (2018). In essence, the construction and manipulation of text embeddings, through the use of machine learning techniques and heavy computational resources, is the key to modern NLP (Devlin et al., 2018). For instance, embeddings can be used as textual features for many downstream machine learning applications (Pennington et al., 2014).

**Language modeling.** The main goal of a language model is to learn the token distribution over text (Keskar et al., 2019). A token can be for instance a word, or a part of a word (Sennrich et al., 2015). Formally, consider a sequence of tokens $s = (t_1, t_2, ..., t_n)$ where each token $t_i$ comes from some set of symbols. The goal of our model is then to learn the joint probability distribution $p(s)$. The sequential nature of text allows for a decomposition of the join distribution into

$$p(s) = \prod_{i=1}^{n} p(t_i | t_{<i}), \tag{4.1}$$

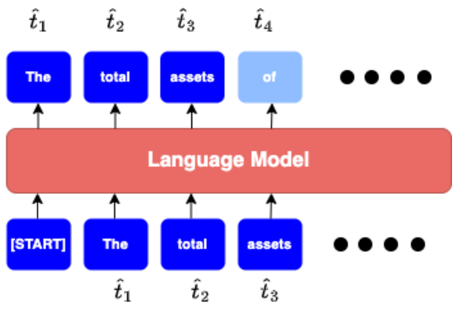


**Figure 4.2:** Schematic overview of text generation using language models. Tokens $\hat{t}_i$ are iteratively sampled from the model's learnt conditional distribution, and inserted back into the conditional for the sampling of the next token $\hat{t}_{i+1}$.

transforming the modeling problem into a problem of next-word prediction (Bengio et al., 2003). Modern models estimate $p(s)$ by some neural architecture with parameters $\theta$, outputting the distribution $p_\theta(t_i|t_{<i})$, trained by maximum likelihood estimation on some textual dataset (Keskar et al., 2019). Internally, these models utilize the aforementioned numerical representations of text to produce this estimate. Finally, one can generate text by iteratively sampling tokens $\hat{t}_i$ from the learnt distribution $p_\theta(t_i|t_{<i})$, inserting the sampled token into the conditional every iteration (Radford et al., 2019). Figure 4.2 shows a simplified example of text generation using language models.

### 4.1.2 The Transformer Architecture

The *transformer* is a deep learning architecture that uses an encoder-decoder structure to solve various NLP tasks. Here we provide a simplified overview of the architecture and the function of its components. This is by no means a comprehensive description of the transformer, and we urge the reader to refer to the original paper by Vaswani et al. (2017) for an in-depth explanation.

**Attention.** The main feature of the transformer is the attention mechanism. Attention allows the network to weigh the importance of different parts of an input sequence when making predictions. This is done by computing a learnable weighted sum of the word embeddings in the input sequence, outputting a modified sequence of word embeddings based on the surrounding words. Vaswani et al. (2017) proposes a scaled dot-product attention mechanism consisting of a multiplication of the so-called *query* ($Q$) and *key* ($K$) matrices, normalized by the square-root of $d_k$ (the dimension of $K$) followed by a softmax operation. This results in so-called *attention weights*, which are subsequently multiplied by the *value* ($V$) matrix producing the final output

$$\text{Attention}(Q, K, V) = \text{softmax}(\frac{QK^T}{\sqrt{d_k}})V. \tag{4.2}$$

Specifically, $Q$, $K$, and $V$ are learnable projection matrices for the input to the attention block. We call it *self-attention* when the query, key, and value matrices are derived from the same input sequence. Figure 4.3 illustrates the self-attention mechanism. The attention operation allows the model to capture the semantic relationships between words, even if they are distant in the sequence.

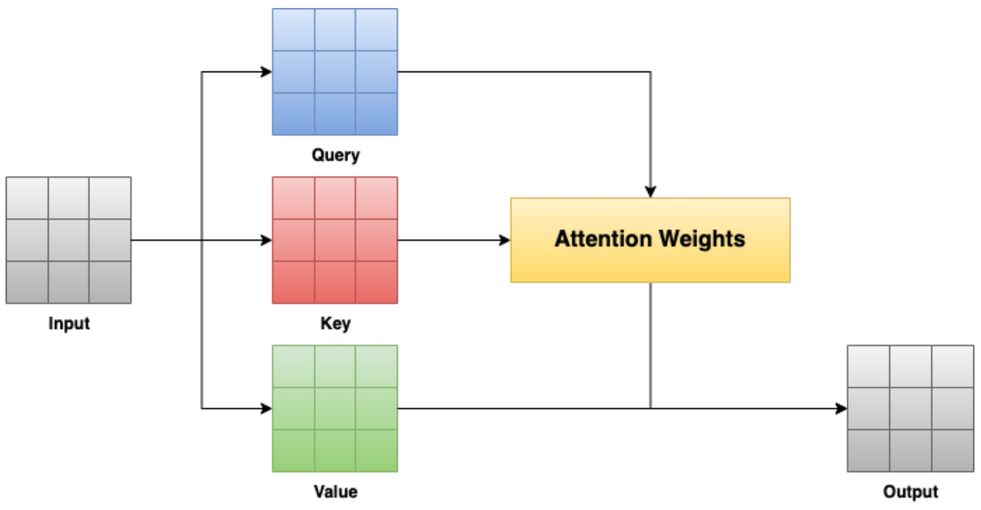


**Figure 4.3:** The self-attention operation proposed by Vaswani et al. (2017). The query, key, and value matrices are here all derived from the same input.

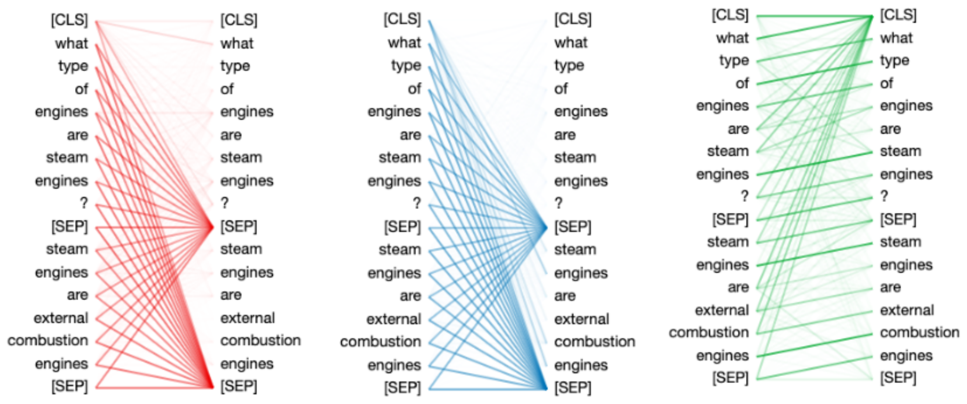


**Figure 4.4:** Graphical representation of the self-attention mechanism. The images represent the attention between tokens for three different attention heads in the final layer of a BERT(Devlin et al., 2018) encoder.

The application of multiple attention mechanisms can thus produce new sets of *contextual* word embeddings that encapsulate the semantic relationship between the original words in the input sequence (Vaswani et al., 2017). In practice, one usually divides the attention mechanism into different so-called *heads*, enabling parallelism (Vaswani et al., 2017). Figure 4.4 shows a representation of the self-attention mechanism, for three such heads.

**Encoder.** In a transformer, the encoder module processes the input sequence and generates a sequence of new word embeddings that capture the relationship between words. This is done through the repeated use of identical sub-layers each comprised of a self-attention mechanism, a position-wise fully connected feed-forward network, and a residual skip connection (Vaswani et al., 2017). Layer normalization (Ba et al., 2016) is applied after each sub-layer. The left part of Figure 4.5 shows a simplified schematic overview of the encoder architecture.

The output of each layer in the encoder becomes a sequence of representations that tries to capture the meaning of each token in the input sequence. The output is thus usually referred to as *contex-*

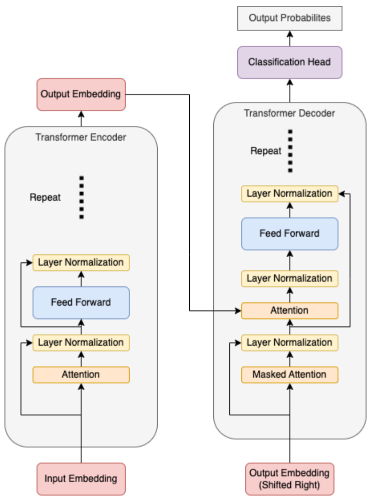


**Figure 4.5:** Simplified overview of the original transformer architecture as proposed by Vaswani et al. (2017). The left module represents the encoder, and the right module represents the decoder.

*tualized embeddings* since the output representation of each token will depend on the surrounding tokens in the context. This sequence of contextual embeddings can subsequently be fed to the decoder to generate text, or, as suggested by Devlin et al. (2018), sent to a different type of network head that is designed for other NLP tasks, e.g. span extraction or classification.

**Decoder.** Vaswani et al. (2017) describe how the decoder module generates an output sequence based on the encoded input sequence. The right module in Figure 4.5 shows the architecture of the decoder. The decoder is autoregressive, meaning that it predicts the next token $\hat{t}_i$ in a sequence given the previously generated tokens $\hat{t}_{<i}$ and the encoded input sequence $z$. It does this by iteratively estimating $p_\theta(t_i|\hat{t}_{<i}, z)$ and sampling from that learnt conditional probability distribution, for instance greedily or using beam search (Sutskever et al., 2014). Thus, to generate text, you do several forward passes through the decoder, generating a token $\hat{t}_i$ each time. Each generated token is then fed back into the decoder to be conditioned on in the next forward pass. In this sense it functions as a typical language model.

Similar to the encoder, the decoder consists of multiple layers of self-attention and attention, feed-forward networks, layer normalization, and residual connections followed by a standard classification head consisting of a linear layer and a softmax activation function (Vaswani et al., 2017). To condition on the encoded input $z$, the decoder attends on both the encoded input sequence *and* the generated output sequence (Vaswani et al., 2017); in Figure 4.5 this is the arrow from the encoder output to the decoder body. At each step in the sequence, the decoder solves a classification problem where the number of classes is equal to the size of its vocabulary. As with any classifier, the decoder can be trained by minimizing the negative log-likelihood of generating the correct next word at each step in the sequence. During training a so-called *casual mask* has to be applied to make the decoder only use information from previous tokens (Raffel et al., 2019).

### 4.1.3 Transfer Learning

Transformer-based models are generally large and expensive to train (Yao et al., 2021). Large models require large datasets and powerful hardware which imposes practical constraints. The *transfer learning* technique is a common remedy for this problem. The idea for encoder-based models is explained by Zhuang et al. (2019) as follows:

1. *Pre-train* a large model, often referred to as the *backbone* model, using a large amount of unlabeled text data.
2. Freeze some, or all, of the parameters of the backbone model, and attach a head that is suited for some downstream task, e.g question-answering.
3. *Fine-tune*, i.e train, the model with the head attached on the downstream task using a smaller, task-specific, labeled dataset.

Here, we only discuss the notion of transfer learning in the setting of encoder-based models. However, it is also possible to pre-train and fine-tune models that additionally consist of a decoder component, see for instance (Raffel et al., 2019).

**Pre-training.** The purpose of the pre-training step is to learn broad and general knowledge (Raffel et al., 2019) and create re-usable and fruitful representations of language to be used for downstream tasks (Devlin et al., 2018). These token representations are learned through modeling the joint distribution of tokens as in canonical language modeling, and can subsequently be used downstream for solving many NLP tasks (Devlin et al., 2018). For instance, *BERT* (Devlin et al., 2018) produces embeddings that serve as textual features to be used by machine learning models to solve tasks such as text classification or question answering. Although pre-training is an expensive operation (Yao et al., 2021), it only has to be done once.

Proper pre-training of models is crucial for its downstream performance (Raffel et al., 2019) and is typically done in a self-supervised fashion on large unlabeled corpora (Radford and Narasimhan, 2018; Radford et al., 2019; Devlin et al., 2018). There exists a vast space of pre-training schemes (Ramachandran et al., 2016; Devlin et al., 2018; Dong et al., 2019). While the first pre-training objectives were concerned with causal language modeling (predicting the next token) (Peters et al., 2018; Dai and Le, 2015), Devlin et al. (2018) proposed the objective to reconstruct intentionally corrupted sentences, usually referred to as *masked language modeling* (MLM) or *denoising*. The models are then trained by minimizing the negative log-likelihood of the masked tokens across the dataset. Another popular objective is to predict whether a sentence directly follows another given sentence, referred to as *next sentence prediction* (NSP) (Devlin et al., 2018). In the encoder setting, the predictions are derived from the softmax over the embeddings of the relevant tokens (i.e. the masked tokens for MLM, and `[CLS]` for NSP) (Devlin et al., 2018). A large model which has been pre-trained like this is usually referred to as a *large language model* (LLM) (Carlini et al., 2020).

**Fine-Tuning.** The goal of fine-tuning is making the language model perform well on a specific task (Devlin et al., 2018). According to Devlin et al. (2018), this is commonly done by attaching another layer to the pre-trained LLM, a so-called *head*, and train the head for the task in question. The fine-tuning is hence specific to the task at hand and often uses a smaller, labeled data set (Yao et al., 2021). Many models reach state-of-the-art performance by fine-tuning on task-specific datasets (like Zhang (2019), Izacard and Grave (2020), (Zaheer et al., 2020), among others).

## 4.2 Document store implementations

In many applications, there is a strong requirement on both the storage capacity and the retrieval speed of the document store (Kononenko et al., 2014). This has led to lots of research and numerous vendors providing the required tools to make this work efficiently (Kononenko et al., 2014). Some examples are *Elasticsearch* (Kuc and Rogozinski, 2013) and Facebook AI's *FAISS* (Johnson et al., 2017), that approach the problem of efficient document search and retrieval in slightly different ways. What is common between many approaches is that they represent documents as vectors, or embeddings, with different properties. As we will see in Section 4.3, this is tightly coupled with the purpose and output of the Retriever. Here, we will introduce two types of document store architectures: *sparse* and *dense* document stores, respectively.

**Sparse Document stores.** A simple way to represent a document is by considering the document as a collection of its words, and store the number of times each word occurs in the document in some vector (Sparck Jones, 1972). Here, each axis in the vector space corresponds to a word in the vocabulary. This results in a high-dimensional, *sparse* vector representation of the document, and hence the name of the document store. There are multiple ways to improve this representation, for instance by re-weighting the components of this sparse vector using a TF-IDF-like scheme (Sparck Jones, 1972). How these vector representations are generated in practice is covered in Section 4.3.1.

*Elasticsearch*, released in 2010, is a search engine based on sparse vector representations of documents. It has been widely adopted by large corporations such as Facebook and Netflix, among others (Kononenko et al., 2014). According to Kuc and Rogozinski (2013), Elasticsearch allows for efficient document retrieval through an inverse index lookup and is scalable to enormous data collections. As the underlying vector representations are not generated by transformer-based encoding techniques (which require fine-tuning), Elasticsearch is readily applicable to most document collections.

**Dense document stores.** In contrast to sparse document stores, *dense* document stores persist low-dimensional, dense vector representations of the documents. A key motivation for the development of dense document stores is to allow for semantic search: documents that are semantically similar to a question (or another document) should be returned by the system, instead of only returning documents with high lexical overlap (Mu et al., 2019). How these embeddings are constructed and how semantic similarity is captured through cosine is the key topic of Section 4.3. There exist many implementations of dense document stores. Among these are *FAISS*, an open-source library for storage of and efficient similarity search on dense vectors (Johnson et al., 2017).

**Efficient dense vector search algorithms.** The search for relevant documents given a query is usually conducted by means of some *nearest neighbor algorithm* (Reimers and Gurevych, 2019; Johnson et al., 2017). In simple terms, this is done by finding documents in the proximity of the query in a low-dimensional vector space. Proximity is often measured in terms of dot-product or cosine (Reimers and Gurevych, 2019). When the number of documents in the knowledge base grows, exact nearest neighbor computations may be infeasible. In those cases, one usually employs *approximate nearest neighbor search* (Arya et al., 1998) where computational feasibility is gained at the cost of decreased accuracy. Popular algorithms include *HNSW* (Malkov and Yashunin, 2018), a graph-based similarity search algorithm with logarithmic complexity. FAISS also allows for optimizing for memory by quantizing (Jégou et al., 2011) the embeddings. See Appendix A for an intuitive explanation of document similarity search with accompanied examples.

## 4.3 Retriever Architectures

The responsibility of the Retriever is tightly coupled with that of the document store. Its job is to generate useful vector representations of the query and documents (Karpukhin et al., 2020), respectively, which downstream will be used for query-document similarity search in the document store. Therefore, the Retriever is simply an encoder with an additional hyperparameter $\text{top}_{k,\text{Retriever}}$ dictating how many of the most relevant documents should be passed on to the next component in the pipeline. There exist two main types of Retriever architectures: *sparse* and *dense* Retrievers, described in Sections 4.3.1 and 4.3.2 respectively.

### 4.3.1 Sparse Retrievers

Document retrieval systems have traditionally utilized statistical information about the lexical contents of documents for finding relevant documents to a query (Wilkinson, 1994). For instance, Wilkinson (1994) employs a TF-IDF-like bag-of-words approach to model documents as sparse vectors, which can subsequently be stored in a sparse document store.

A *sparse Retriever* is a module that produces these representations; it can be viewed as an encoder that determines the entries of the vector representation using some handcrafted function of term and document frequencies. Note that this representation does not account for the order of the words in the document (Thakur et al., 2021). Neither does it account for the semantics of the document – documents are encoded and compared to each other on a lexical basis. However, sparse Retrievers generally do not require to be trained on large labeled datasets (Izacard et al., 2021) and can thus be readily used, for instance in conjunction with a sparse document store like Elastisearch.

**BM25.** A popular sparse retrieval model is *BM25* (Robertson and Walker, 1994), which has obtained state-of-the-art results on many retrieval tasks (Svore and Burges, 2009). Although BM25 has existed since the 1990s and is outperformed by modern retrieval techniques on several tasks (Izacard et al., 2021), Ma et al. (2022) have shown that BM25-based question-answering systems are capable of achieving comparable performance results to several modern Retriever architectures on SQuAD Open. BM25 takes a probabilistic approach to document modeling and also accounts for term and document frequencies, and document lengths (Svore and Burges, 2009). While there exist several other sparse Retrievers leveraging the transformer architecture (e.g. *DeepCT* (Dai and Callan, 2020), *SPARTA* (Zhao et al., 2021)), we will not cover or experiment with them in this thesis.

### 4.3.2 Dense Retrievers

Dense Retrievers generate the low-dimensional dense representations of documents and questions to be used downstream by a dense document store for similarity search (Reimers and Gurevych, 2019). The main difficulty here is to encode the questions and documents such that a cosine or dot-product between them actually represents semantic similarity. This is accomplished by dense

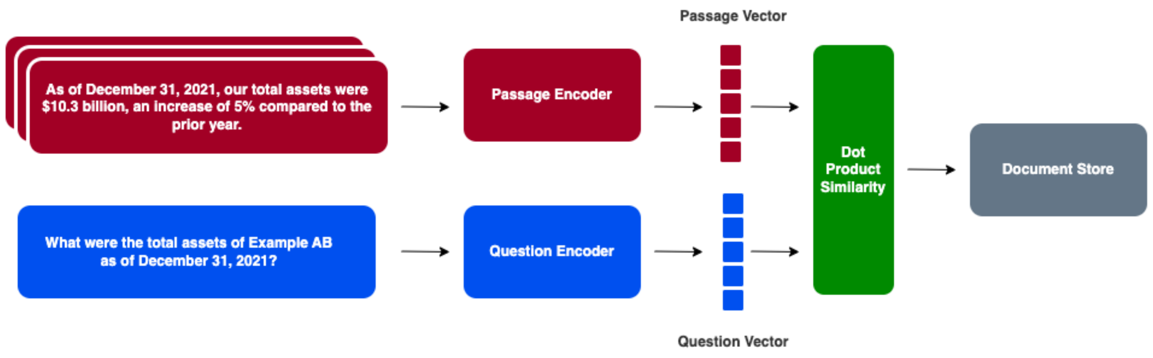

Passage Vector
As of December 31, 2021, our total assets were $10.3 billion, an increase of 5% compared to the prior year.
Passage Encoder
Dot Product Similarity
Document Store
What were the total assets of Example AB as of December 31, 2021?
Question Encoder
Question Vector

**Figure 4.6:** High-level data flow of a bi-encoder dense Retriever at inference. Passages and questions are encoded separately (by separate encoders), and similarity is computed using dot-product.

Retrievers, based on the transformer architecture, through clever training schemes (Karpukhin et al., 2020).

**Dense Retrievers at inference time.** Karpukhin et al. (2020) describe the following three key steps performed by the Retriever at inference time. Here, we assume that the documents have been encoded by the Retriever into vectors at some previous point in time.

1. A question is passed from the user to the Retriever. The Retriever encodes the questions into an embedding vector.
2. The Retriever passes the question embedding to the document store, which uses either an inverted index lookup or one of the algorithms described in Section 4.2 for vector similarity search to find the $\text{top}_{k,\text{Retriever}}$ most similar documents to the question.
3. The Retriever passes these relevant documents to the next component of the pipeline.

Figure 4.6 shows a schematic overview of steps 1. and 2. At what point in time the documents are encoded differs between Retriever architectures: it can be done both at inference time and offline (prior to inference).

**Sentence-Transformers (SBERT).** Reimers and Gurevych (2019) were among the first to introduce a state-of-the-art dense Retriever architecture with their retrieval model *SBERT*. The authors introduced the idea of using a *bi-encoder* architecture – having two separate encoders for the questions and query, respectively – to allow for semantic search. This idea contrasts the previously traditional way of computing document similarity: concatenate the query and the document and pass it through a BERT-based classifier – fine-tuned on a semantic similarity task – predicting the similarity between them. Such an architecture is instead referred to as a *cross-encoder* architecture (Reimers and Gurevych, 2019). Although cosine does not offer an equally expressive quantifier of similarity compared to one based on a cross-encoder network, it allows for much cheaper retrieval as the documents can be embedded offline (i.e. prior to inference). Also, it does not require a forward pass through the BERT-based network (Reimers and Gurevych, 2019).

The original bi-encoder SBERT Retriever generates document or question embeddings using a fine-tuned BERT-encoder (Reimers and Gurevych, 2019). The embeddings are obtained using some pooling operation on the output layer of the encoder; many SBERT-like models simply use the representation of the `[CLS]` token. Reimers and Gurevych (2019) then fine-tune the encoder on 1M labeled sentence pairs from for instance the Stanford Natural Language Inference (SNLI) Corpus using a *contrastive* loss function (Chen et al., 2020) that rewards similar documents that are close to each other (in some metric) and penalizes dissimilar documents that are close.

The NLP community has contributed with vast numbers of SBERT-like Retrievers that are not restricted to be based on BERT-encoders, and that are fine-tuned on different datasets depending on the use case (see the SBERT website[1]). For instance, `all-MiniLM-L12-v2` is based on the (distilled) MiniLM architecture (Wang et al., 2020b), and is fine-tuned using self-supervised contrastive learning (Izacard et al., 2021) on about 1B sentence pairs.

**Dense Passage Retrieval.** Another popular dense Retriever which achieved state-of-the-art results at its release (Mao et al., 2021) is *Dense Passage Retrieval* (DPR), as proposed by Karpukhin et al. (2020). Just like SBERT, it uses bi-encoder architecture but this time using two different encoders for the question and documents, respectively. The encoders are pre-trained BERT encoders that have been jointly fine-tuned in a supervised fashion on four OpenQA datasets, including Natural Questions and TriviaQA. The fine-tuning of DPR is done to capture semantic similarity through dot-product, and it employs the following idea: first a so-called positive document $d^+$ and several negative documents $d_j^-$ are collected for every question $q$ in the dataset. The positive documents contain the answer, while the negative documents do not contain the answers (and are generated in a few different ways, see Karpukhin et al. (2020)). Then, the model is trained by minimizing the negative log-likelihood of the similarity between the positive document and the question, i.e. using the loss

$$\mathcal{L} = -\log \frac{e^{\mathrm{sim}(q,d^+)}}{e^{\mathrm{sim}(q,d^+)} + \sum_j e^{\mathrm{sim}(q,d_j^-)}} \tag{4.3}$$

Minimizing this across the dataset modifies the encoder to map positive documents and the query closer to each other while pushing negative documents further away from the query vector. The usage of a bi-encoder architecture enables offline encoding of the documents as the encoding is performed independently of the question being asked (Karpukhin et al., 2020). Furthermore, while DPR better captures semantic patterns in the text compared to its sparse counterparts, dense Retrievers can suffer from loss of information as each document is compressed into a single vector of fixed size (Luan et al., 2021). Also note that as DPR uses a large language model backend, as are all other dense Retrievers we will cover in this paper, and thus require sufficient computational resources to run efficiently (Mao et al., 2021).

**Contriever.** *Contriever* was presented by Meta AI Research's Izacard et al. (2021) as a response to dense Retrievers that are trained in a supervised fashion and that fail to generalize well to datasets that do not have labels to fine-tune on. Particularly, they claim that sparse Retrievers like BM25 generalize better than many dense Retrievers in the zero-shot setting, without training samples. Izacard et al. (2021) train Contriever using contrastive learning (Chen et al., 2020) – using a loss which rewards similar documents that are close in the vector space and dissimilar documents that are far apart. In contrast to DPR, Contriever trains on positive and negative documents that have

[1]`https://www.sbert.net/index.html`

been collected in a self-supervised fashion, particularly through the *inverse Cloze task* (ICT) proposed by Lee et al. (2019). In ICT, one randomly samples a span of a document and uses this span as the query and the complement of the span as the positive document. The authors sample the negative documents in several different ways, but most importantly independent of the positive document.

The authors show that their model outperforms BM25 on 11 out of 15 information retrieval benchmark datasets. Moreover, the current leader (as of May 2023) of the TriviaQA benchmark fine-tunes a Contriever and use as their retrieval component (Izacard et al., 2022). The Contriever was published in two forms: the pre-trained `Contriever`, and the `Contriever MS MARCO`: a Contriever fine-tuned on Microsoft's vast dataset collection *MS MARCO* (Nguyen et al., 2016). The latter attains slightly better performance than on several benchmark datasets (Izacard et al., 2021).

## 4.4 Reader Architectures

As mentioned in Chapter 2, there is a dichotomy between Reader architecture types. In Sections 4.4.1 and 4.4.2, we cover the basics of *extractive* and *generative* Reader architectures, respectively.

### 4.4.1 Extractive Readers

*Extractive* Readers, also known as *encoder-based* Readers, answer questions by extracting a span from a given document. They are constructed by combining a pre-trained transformer-based encoder with a classification head, as proposed by Devlin et al. (2018). The pre-trained encoder plays a crucial role in capturing the semantic connections within the question and the context, enabling the extraction of the answer. The Reader's output consists of probabilities assigned to each token in the context, indicating the likelihood of it being the start or end token of the span.

**Extractive Readers at inference time.** Figure 4.7 displays the flow of information in the extractive Reader at inference time, as described by Devlin et al. (2018). First, the question and context are concatenated and fed to the pre-trained encoder network as a sequence of tokens $(t_{\texttt{[CLS]}}, t_1, ..., t_{\texttt{[SEP]}}, ..., t_n)$. The encoder then produces a sequence of contextualized embeddings $(h_{\texttt{[CLS]}}, h_1, ..., h_{\texttt{[SEP]}}, ..., h_n)$ from the input tokens. These embeddings are then fed through a classification head, producing the two desired conditional probability distributions $\text{softmax}(\mathbf{hW}_s)$ and $\text{softmax}(\mathbf{hW}_e)$ from which an answer span can be derived. Here, $\mathbf{h} \in \mathbb{R}^{n+2,H}$ is the matrix of token embeddings, where $H$ is the size of the encoder latent space. Thus, this requires two additional learnt weight vectors compared to a pure BERT encoder: $\mathbf{W}_s, \mathbf{W}_e \in \mathbb{R}^H$ for start and end token distributions, respectively (Devlin et al., 2018).

**Training an extractive Reader.** There are two steps for training an extractive Reader. First, one needs a pre-trained encoder. One could either use a published model, such as BERT or RoBERTa (Liu et al., 2019c), or proceed to pre-train it yourself using for instance masked-language modeling (Devlin et al., 2018). Second, we need to fine-tune the weights $\mathbf{W}_s, \mathbf{W}_e$ of the classification head. As this is simply a classifier, we first collect a large set of labels (`question`, `document`, `start_token_index`, `end_token_index`) and train it using the sum of negative log-likelihoods of the correct start and end tokens (Devlin et al., 2018). While it is possible to use more clever

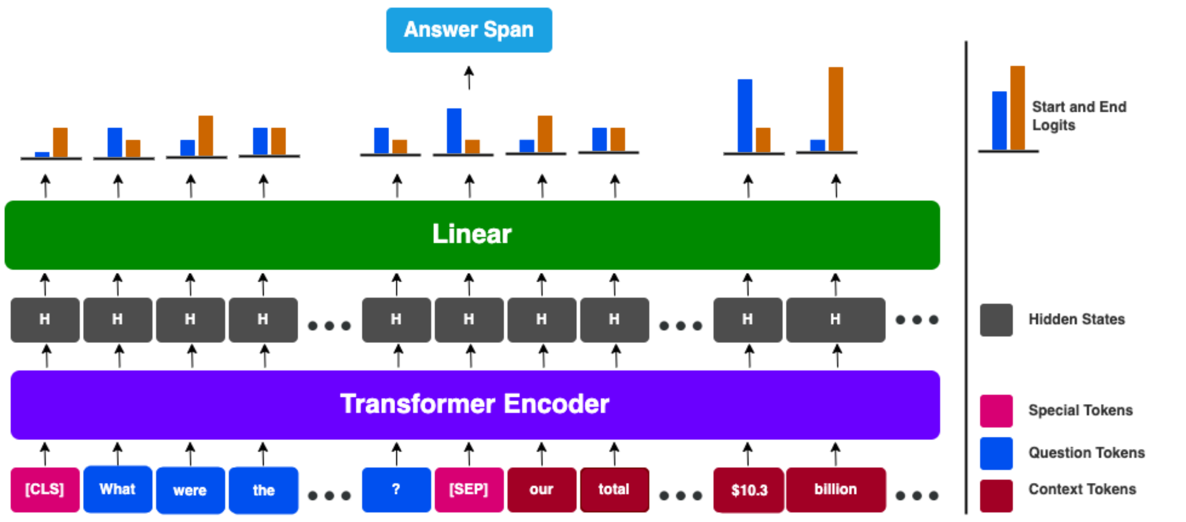


**Figure 4.7:** High-level architecture of an extractive Reader using transformer encoder backbone at inference time. The question and document is concatenated and inputted to the Reader, and the output is the (unnormalized) probabilities of each token being the start and token token, respectively.

fine-tuning schemes to obtain better performance of the resulting Reader (see, for instance, Zhang (2019)), this is the basic idea.

**The issue with long texts.** One issue with self-attention-based transformer encoder networks is their ability to handle long texts (Zaheer et al., 2020). The self-attention mechanism has a quadratic time complexity with respect to input sequence length (Zaheer et al., 2020) due to the need for pairwise comparison between all tokens in the sequence. This results in a disproportionate computational cost for longer sequences and can lead to slow execution times and memory issues (Devlin et al., 2018). In order to remedy this, one can, for instance, either make use of *chunking* to split documents into smaller, potentially overlapping subsequences, or use a Reader that uses an LLM back-bone with an alternative attention mechanism with more beneficial complexity (like Beltagy et al. (2020) or Zaheer et al. (2020)).

**Popular model types.** There exists numerous instances of extractive Readers made available by the NLP community on HuggingFace. These mostly differ in terms of the LLM backbone used and on which question-answering dataset the head has been fine-tuned. Many are based on BERT-like architectures such as BERT (Devlin et al., 2018), RoBERTa (Liu et al., 2019c), or multilingual counterparts (Conneau et al., 2019); or knowledge distilled (Hinton et al., 2015) versions such as DistilBERT (Sanh et al., 2019) or MiniLM (Wang et al., 2020b). Lately, many highly successful participants of the TriviaQA and Natural Questions leaderboards use LLM back-bones that employ alternative attention mechanisms to better handle long text sequences, for instance, DeBERTa (He et al., 2020), BigBird (Zaheer et al., 2020), PoolingFormer (Zhang et al., 2021), and ClusterFormer (Wang et al., 2020a). Among the most popular datasets to fine-tune are SQuADv2, Natural Questions, and TriviaQA (Zhu et al., 2021).

## 4.4.2 Generative Readers

In contrast to extractive Readers, *generative Readers* generate answers using a decoder component and by conditioning on the input sequence. Generative Readers can be divided into two architectural subtypes: *encoder-decoder* models and pure *decoder* models (Wolf et al., 2020). In the following, we briefly introduce the concept of each of them. For clarity, we will refer to the specific architecture when discussing generative models.

**Encoder-decoder models.** An encoder-decoder generative Reader does not extract an answer span from a given piece of text but instead generates an answer in a sequence-to-sequence (seq2seq) (Raffel et al., 2019) manner (Mao et al., 2021). Many NLP tasks can be viewed as seq2seq modeling tasks (Yin and Wan, 2022); open-retrieval question answering can be considered a seq2seq task as we wish to generate an answer (a sequence of tokens) given the concatenation of a question and documents (Raffel et al., 2019). While generative models are capable of answering questions without a provided document – and instead only learn in their parameters, see for instance Roberts et al. (2020) or Brown et al. (2020) – we only cover their usage in conjunction with a retrieved context. The key components of an encoder-decoder Reader are (unsurprisingly) the encoder and decoder of a large language model, potentially fine-tuned on certain tasks.

Encoder-decoder Readers are autoregressive: they generate an answer to a question by iteratively predicting the next token in a sequence while conditioning on the encoded representation of the input text (the question and documents) and previously generated tokens. As such, it consists of some version of the full transformer architecture proposed by Vaswani et al. (2017).

**Encoder-decoder models at inference time.** Raffel et al. (2019) describe the formal rationale behind generating an answer given a question and a set of documents using an encoder-decoder Reader. Figure 4.8 shows how answers are generated in an encoder-decoder-based Reader. It is similar to how the original transformer architecture generates text (as described in Section 4.1.2), i.e. by iteratively estimating and sampling from $p(t_i|\hat{t}_{<i}, z)$. The main adaptation in the OpenQA setting is that the encoded input now consists of the encoded concatenation of the question $q$ and the documents $d_i$, i.e. $z = \text{Encoder} \circ \text{concat}(q, d_1, ..., d_{\text{topk,Retriever}})$. The concatenation of the question and documents is sometimes referred to as the *prompt*. The same $z$ is then fed into the network in each forward pass. While there exist more sophisticated approaches for generating answers using retrieved evidence (for instance *Retrieval-Augmented Generation* (Lewis et al., 2020) or *Fusion-in-Decoder* (Izacard and Grave, 2020)), this is among the simplest methods.

**Training an encoder-decoder model.** The training of an encoder-decoder Reader can be divided into a pre-training step and a fine-tuning step (Raffel et al., 2019), just as encoder-based transfer learning described in Section 4.1.3. The main difference in this setting is that the encoder and decoder is now jointly pre-trained, meaning that the model is trained using the decoder output instead of the encoder output. Similarly to encoder pre-training, the objective is usually to generate the original tokens given an intentionally corrupted sentence. For instance, Raffel et al. (2019) uses objectives similar to BERT-like denoising, and Lewis et al. (2019) uses several objectives including token shuffling (Liu et al., 2019a) and document rotation. The model is then trained using the cross entropy between the generated output and the uncorrupted sentence (Raffel et al., 2019). To allow for contexts or instructions to be fed to the model, it can be pre-trained using

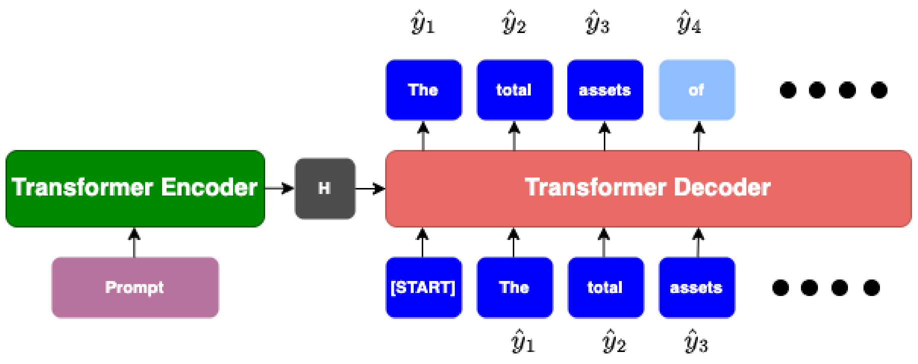


**Figure 4.8:** High-level view of the iterative text generation in an encoder-decoder-based Reader. The prompt may be a concatenation of the question and context(s) and can also contain an instruction. The latent representation of the prompt is fed into the decoder every iteration of the token generation. Here, the generated tokens are labeled $\hat{y}_i$.

*prefixes* (Raffel et al., 2019). In that case, the input sequence is first split in two parts. The first part (the prefix) is then considered fully visible (not causally masked) and fed to the encoder, and the second part is fed to the decoder for masked language modeling as usual.

Fine-tuning an encoder-decoder model on a supervised task is quite similar to that of an encoder model, as described by Raffel et al. (2019). The main difference lies in the fact that fine-tuning simply a head is not sufficient; in this case, we must fine-tune the model to generate the task-specific output using the decoder. The authors describe two approaches: fine-tuning only certain parts of the underlying transformer stacks (Houlsby et al., 2019), or gradually unfreezing (Howard and Ruder, 2018) more and more layers of the encoder and decoder in parallel. As usual, the model can be fine-tuned using maximum likelihood on the correct output sequence.

**Popular encoder-decoder models.** The space of seq2seq models is vast: Google's *T5* (Raffel et al., 2019) advances the state-of-the-art on many benchmark datasets at its release; Chung et al. (2022) additionally fine-tunes T5 (*inter alia*) of various sizes on 1.8k tasks, creating *FLAN-T5* which outperforms T5 on many tasks; Facebook's *BART* (Lewis et al., 2019) achieves state-of-the-art results at its release on many tasks, including question answering. Fortunately, many of these models are available through HuggingFace.

**Decoder models.** Similarly to encoder-decoder Readers, pure *decoder Readers* generate answers instead of extracting them. The main difference is that decoder models do not condition the encoder output to determine its output token distribution. Among the main motivations for not using an encoder is (1) the difficulty in effectively transferring learned input representations for usage in vast amounts of tasks simultaneously, and (2) which task best learns these task-agnostic embeddings (Radford and Narasimhan, 2018). Thus, the goal of many decoder models is to better generalize to various heterogenous textual tasks using a single model (Floridi and Chiriatti, 2020), without requiring access to large labeled datasets to fine-tune on whenever a new task is to be solved. Many task-agnostic decoder-based models have been shown to outperform task-specific architectures (like many encoder-based models) on a variety of NLP tasks (Radford and Narasimhan, 2018;

Radford et al., 2019), and decoder-based *ChatGPT* has become a huge commercial success (Lund and Wang, 2023). Among the most popular decoder models are OpenAI's suite of *GPT* models (Radford and Narasimhan, 2018; Radford et al., 2019; Floridi and Chiriatti, 2020; OpenAI, 2023), Meta's *LLaMA* (Touvron et al., 2023) and *Alpaca* (Taori et al., 2023), and Salesforces' *CTRL* (Keskar et al., 2019).

**Decoder models at inference time.** Similarly to encoder-decoder models, decoder models generate tokens $\hat{t}_i$ by iteratively sampling from a learnt $p_\theta(t_i|\hat{t}_{<i}, q, d_1, ..., d_{\text{topk,Retriever}})$. However, now the question and document are now fed directly to the decoder rather than through the encoder's latent representation $z$ (Radford and Narasimhan, 2018). Figure 4.9 shows how a decoder-based Reader generates an answer. By directly conditioning on the context through the decoder input, decoder models are capable of "learning" how to solve a task without explicitly updating its weights (Xie et al., 2021) but instead directly through the latent information in the decoder's hidden states. These hidden states are essentially the key, value, and query matrices of the decoder, which are determined by the decoder input (for instance the prompt and previously generated tokens) and which themselves contribute to the final output of the decoder. In this way, information from the prompt is passed on through these matrices to determine the next token.

Why and how models learn more specifically from context is an active area of research, and Xie et al. (2021) provide a framework for viewing this type of instruction learning from the point of view of (implicit) Bayesian inference. In essence, the authors claim that the conditioning on context helps the model properly identify the task by modifying the way the output distribution is produced. This property is inherently learnt during the pre-training process of the models (Xie et al., 2021).

**In-context learning.** The ability for decoder models to learn by conditioning on long input sequences is often referred to as *in-context learning* (Floridi and Chiriatti, 2020). In contrast to the traditional way to teach a model a new task – by fine-tuning (updating model weights) on some labeled dataset – the in-context learning happens without any parameter updates (Lazaridou et al., 2022). Some decoder models reach state-of-the-art results on certain NLP tasks by using this technique of including some particular information in the prompt (Floridi and Chiriatti, 2020; Lazaridou et al., 2022). Particularly, the information included in the prompt is typically (among other things) (1) explicit task instructions and (2) examples of the task to be solved (Ye et al., 2023; Radford et al., 2019; Floridi and Chiriatti, 2020). This technique is usually referred to as *zero-shot*, *one-shot*, and *few-shot learning*, depending on how many examples are provided in the prompt at inference time. In the few-shot setting, the model is provided both an instruction of the task, such as `"Given a context, please answer the question"`, and several examples of the task (essentially a label from a QA dataset like SQuAD) (Floridi and Chiriatti, 2020). In the zero-shot and one-shot settings, zero or one example are provided in the input, respectively, along with the instruction (Floridi and Chiriatti, 2020). We emphasize again that no parameters are updated using these examples – they are simply conditioned on at inference time. Lots of research is today conducted on how to efficiently formalize and format these prompts (Ye et al., 2023; White et al., 2023; Floridi and Chiriatti, 2020).

A limiting factor to how many examples can be provided at inference time is the maximal token length the model accepts as input. GPT-3.5 allows for 4097 tokens (Floridi and Chiriatti, 2020), while FLAN-T5 only accepts 512 tokens (Chung et al., 2022).

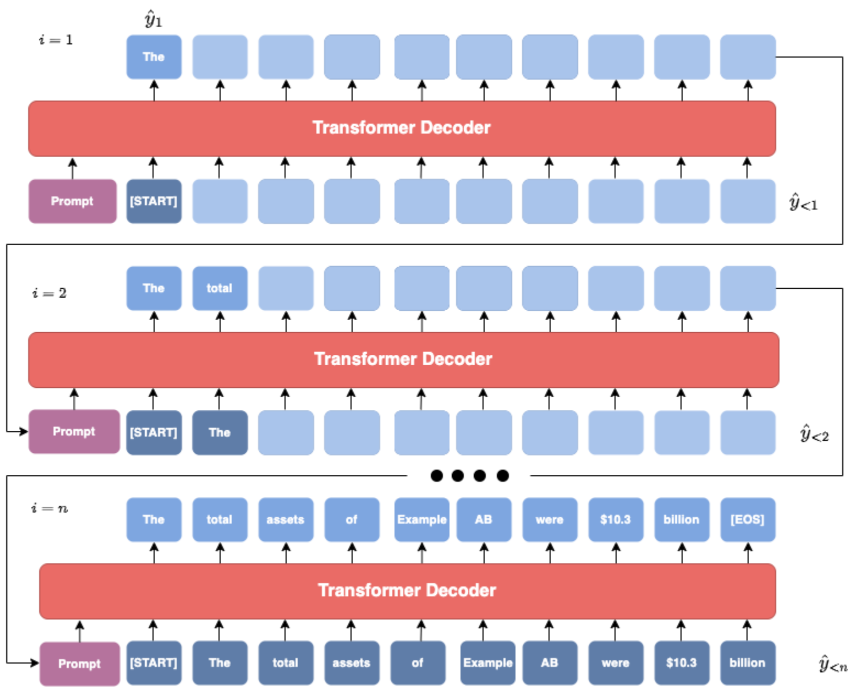


**Figure 4.9:** High-level view of the iterative text generation in a decoder-based Reader. The prompt may be a concatenation of the question and context(s) and can also contain an instruction. Here, the generated tokens are labeled $\hat{y}_i$.

# Chapter 5

# Method

Many design decisions must be made when developing an open-retrieval question-answering system for a specific application. These decisions include selecting the components for the pipeline, determining the underlying architectures of these components, choosing the appropriate training and evaluation data, and optimizing hyperparameters. Moreover, it is crucial to consider the requirements imposed by the final use case. In the subsequent sections, we outline our approach to constructing and assessing OpenQA systems for our banking application.

**Experiment setup.** We run the following three high-level experiments in order to find a performant OpenQA system for our application.

1. **Finding best components and hyperparameters.** Evaluate the performance of canonical Retriever-Reader QA systems on three benchmark datasets. The purpose is to get insights into how the choice of components, their underlying architectures, and hyperparameters affect the system's performance on popular benchmark datasets.
2. **Augmenting the pipelines.** Evaluate a more sophisticated QA system, additionally using a ranking component and an augmented scoring mechanisms for determining the final answers to a given query. The purpose is to investigate whether performance can be improved by adding additional simple components.
3. **Banking-specific evaluation.** Evaluate the best systems on the banking data. The purpose is to uncover the performance of our systems in the banking domain, and particularly its relationship to the performance on benchmark data.

Figure 5.1 shows an overview of our experimentation workflow. In the following three sections we describe in further detail the steps carried out in each of these experiments. In the first experiment, we try a large set of Retriever and Reader models on three benchmark datasets. In the subsequent experiments, we evaluate the best-performing models on a single dataset; first on the most difficult

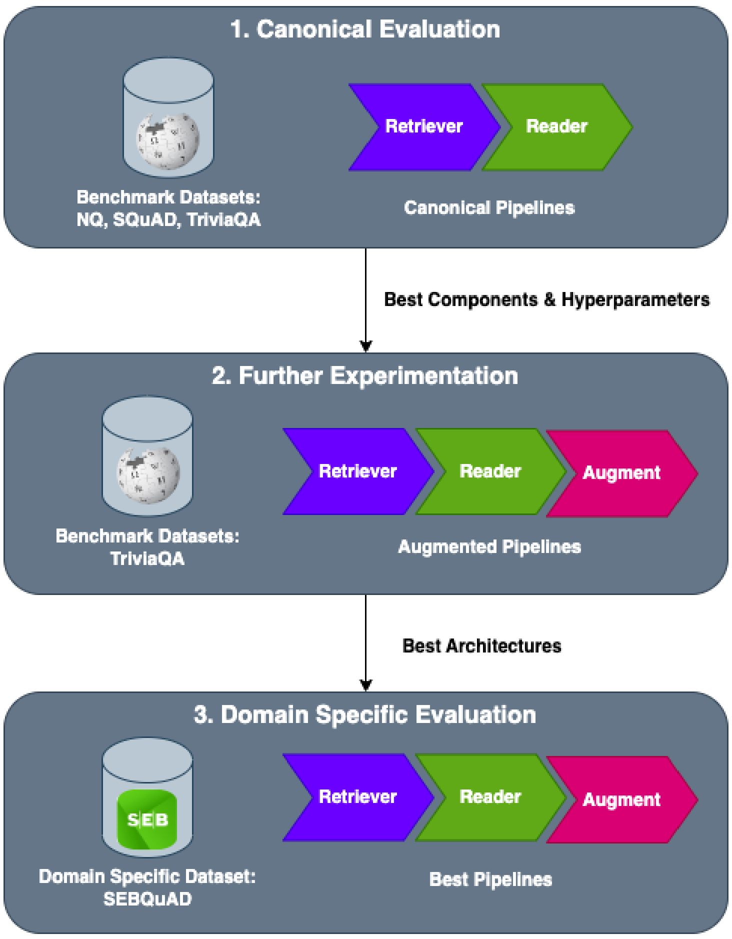

1. Canonical Evaluation
Benchmark Datasets: NQ, SQuAD, TriviaQA
Retriever
Reader
Canonical Pipelines
Best Components & Hyperparameters
2. Further Experimentation
Benchmark Datasets: TriviaQA
Retriever
Reader
Augment
Augmented Pipelines
Best Architectures
3. Domain Specific Evaluation
SEB
Domain Specific Dataset: SEBQuAD
Retriever
Reader
Augment
Best Pipelines

**Figure 5.1:** Our experimentation workflow. We use the benchmark datasets in order to find suitable configurations for our application-specific system which we finally test using the SEBQuAD dataset

benchmark dataset, and then on the banking data. Lastly, Section 5.4 covers preprocessing and implementation details that hold for all of our experiments.

## 5.1 Evaluating Retriever-Reader systems

To start, we evaluate typical Retriever-Reader pipelines on benchmark datasets. Figure 5.2 shows the data flow in such a pipeline. The evaluation is carried out in two steps. In Section 5.1.1, we find pipeline hyperparameters that will be utilized across all configurations. In Section 5.1.2 we describe which pipeline components that undergo evaluation.

### 5.1.1 Finding optimal pipeline hyperparameters

Ideally, hyperparameters should be tuned individually for each system configuration. Due to time constraints, we instead find a single set of hyperparameters that will be used for all our systems. Particularly, we find the optimal number of documents fetched by the Retriever, $\text{top}_{k,\text{Retriever}}$, and the number of answers returned by the Reader, $\text{top}_{k,\text{Reader}}$.

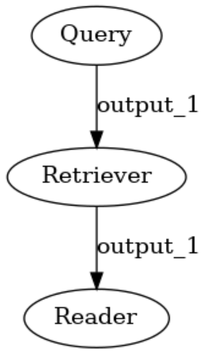


**Figure 5.2:** Canonical Retriever-Reader pipelines for open-retrieval question answering. The final answers are the output of the Reader, which in itself is fed relevant documents by the Retriever.

**Retriever models**

| Model name | Training type | Trained on |
|---|---|---|
| **DPR** | Supervised | Natural Questions, TriviaQA, SQuAD, ... |
| **BM25** | - | - |
| **SBERT** (`all-MiniLM-L12-v2`) | Self-supervised | 1B+ sentence pairs, from Natural Questions, TriviaQA, SQuADv2 among others |
| **Contriever MS MARCO** | Self-supervised | Wikipedia, CCNet, fine-tuned on MS MARCO |

**Table 5.1:** Retriever models evaluated in the Retriever-Reader system, how they were originally trained and on which datasets. Note that BM25 was not trained on any data.

## Finding the optimal number of retrieved documents.

We conduct two experiments to understand the dynamic between $\text{top}_{k,\text{Retriever}}$ and performance. Ultimately, we wish to find an optimal $\text{top}_{k,\text{Retriever}}$ for a general Retriever-Reader pipeline. First, we inspect its effect on Retriever recall. Second, we observe its effect on end-to-end performance.

**Effect on Retriever recall.** To expose the relationship between Retriever recall and $\text{top}_{k,\text{Retriever}}$, we compute the recall while varying $\text{top}_{k,\text{Retriever}} \in [1, 50]$. This evaluation is done on Natural Questions and SQuADv2 dev sets for each of the Retrievers in Table 5.1. Using this, we can define a recall threshold $\tau_{\text{recall}} = 0.90$ which can be used as an indicator for a reasonable value of $\text{top}_{k,\text{Retriever}}$ (by choosing $\text{top}_{k,\text{Retriever}}$ such that the recall is at least $\tau_{\text{recall}}$).

**Effect on end-to-end performance.** Next, we assess the effect of varying $\text{top}_{k,\text{Retriever}}$ on end-to-end performance. We do this by calculating the end-to-end F1 score for different values

**Encoder-based Reader models**

| Architecture | Fine-tuned on | HuggingFace identifier |
|---|---|---|
| DeBERTa | SQuADv2 | `deepset/deberta-v3-base-squad2` |
| MiniLM | SQuADv2 | `deepset/minilm-uncased-squad2` |
| MiniLM | Natural Questions | `remunds/MiniLM_NaturalQuestions` |
| DistilBERT | Natural Questions | `AsmaAwad/distilbert-base-uncased-NaturalQuestions` |
| DistilBERT | TriviaQA | `datarpit/distilbert-base-uncased-finetuned-natural-questions` |
| BigBird | TriviaQA | `google/bigbird-base-trivia-itc` |

**Table 5.2:** Reader models evaluated in the canonical Retriever-Reader system. We try both a set of different architectures and also vary the question-answering datasets on which the models have been fine-tuned on.

**Multiple-answer pipelines**

| Retriever | Reader |
|---|---|
| DPR | DeBERTa |
| DPR | BigBird |
| SBERT | BigBird |

**Table 5.3:** Components used in the three pipelines evaluated when allowing for multiple answers. Each row represents a pipeline configuration.

of $\text{top}_{k,\text{Retriever}} \in [1, 50]$, with a step size of 5. This is done on the TriviaQA Wikipedia dev set, using an SBERT Retriever and BigBird-based Reader (see Tables 5.1 and 5.2, respectively).

## Evaluating performance when allowing for multiple answers.

We also evaluate the performance of a Retriever-Reader pipeline when allowing for more answers. Here, we vary $\text{top}_{k,\text{Reader}} \in [1, 8]$, and fix the number of documents fetched by the Retriever. Table 5.3 shows the components of the three pipelines being evaluated. All pipelines are evaluated on the TriviaQA Wikipedia dev set and using about 36,000 documents in the document store. This experiment gives us an idea of the marginal improvement of the system when allowing for more answers. This could be particularly important for our banking application, as we might want to allow for a set of alternative answers to be the output of the system.

**Generative Reader models**

| Architecture | Type | HuggingFace identifier |
|---|---|---|
| FLAN-T5 | Encoder-decoder | `google/flan-t5-xl` |
| GPT-3.5 (text-davinci-003) | Decoder | - |

**Table 5.4:** Generative Reader models evaluated in the canonical Retriever-Reader system.

### 5.1.2 Evaluating component configurations

We evaluate 32 system configurations on four benchmark datasets to determine which model architectures to test further, and which benchmark datasets to fine-tune and evaluate. Our subsequent experiments will then use these best-performing components and the most difficult datasets. In the following, we evaluate pipelines with extractive and generative Readers, respectively.

**Assessing extractive Readers.** First, we evaluate pipelines containing extractive Readers. Tables 5.1 and 5.2 shows the Readers and Retrievers tested, respectively. All combinations of components are evaluated in terms of F1 score. To display the models' abilities to generalize well, the pipelines are evaluated on four development datasets: SQuAD Open, Natural Questions, TriviaQA verified Wikipedia, and TriviaQA verified Web. We use a value of $\text{top}_{k,\text{Retriever}}$ which gives the best results from Section 5.1.1. The number of returned answers from the Reader is set to 1, to comply with official evaluation scripts. Note that we use models that have already been fine-tuned by members of the HuggingFace community; we do not fine-tune ourselves.

**Assessing generative Readers** Next, we evaluate pipelines containing encoder-decoder and decoder-based Readers, respectively. The experiments are mostly identical to those of the extractive Readers, except that we only evaluate them only TriviaQA verified Wikipedia and Web. Table 5.4 shows the generative Readers evaluated. We evaluate both the zero-shot and one-shot performance of the Readers. In the one-shot setting, we provide an example from SQuAD in the Reader's prompt. The context of this example is truncated for FLAN-T5 in order to fit in the maximal input length of the model (512 tokens). Appendix B contains details on the prompts used. We do not do extensive experimentation with the prompt formatting or contents.

## 5.2 Augmenting the pipelines

Next, we create two types of new pipelines by adding some additional components to the Retriever-Reader system. The goal is to see if there is an improvement to be found by creating a more sophisticated pipeline, but still using simple components. First, we additionally filter relevant retrieved documents using a *Ranker*. Second, we include an augmented scoring mechanism for determining the final predicted span.

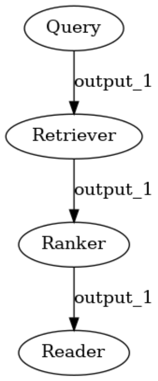


**Figure 5.3:** A schematic overview of the data flow in a Retriever-Reader pipeline augmented with a Ranker component filtering only the most relevant retrieved documents.

### 5.2.1 Filtering documents using a Ranker

**Ranker.** The simplest addition to the Retriever-Reader pipeline is to insert a so-called *Ranker* component between the Retriever and Reader. Its purpose is to filter out from a medium-sized set of contexts (say up to 50) only a few very relevant contexts (say 1 to 10). Ideally, the Ranker should have a similar recall but better precision compared to the Retriever. Formally, it is a classifier that, given a query, for every document predicts the probability of the gold answer being contained in that document. For this, we can employ a model that is trained on a semantic textual similarity task, for instance, an SBERT-type model, if we assume that documents that are more similar to the query also are more likely to contain the answer. Since we want a more sophisticated similarity measure than the Retriever-provided dot-product between embeddings, we will use an SBERT with a cross-encoder architecture: namely `cross-encoder/ms-marco-MiniLM-L-12-v2`. Just as for the Reader, the query and document are concatenated before being regressed by the Ranker to form the similarity prediction. Lastly, only the $\text{top}_{k,\text{Ranker}}$ documents with the largest predicted probabilities are passed on to the Reader for further processing. Note that $\text{top}_{k,\text{Ranker}}$ is considered a hyperparameter to be tuned. Figure 5.3 shows a pipeline with a Ranker component. For brevity, we refer to Retriever-Reader pipelines augmented with a Ranker as *Ranker pipelines*.

**Tuning hyperparameters.** The hyperparameter tuning is carried out in two steps. First, we grid search the optimal $\text{top}_{k,\text{Ranker}} \in [1, 10]$ using two pipelines. Both pipelines are using SBERT Retrievers, an SBERT Ranker mentioned previously, and the two previous best extractive Readers (from the experiments carried out in Section 5.1.2). We only use a single Retriever type as we hypothesize that it is inherent properties of the Reader that determine how well it is able to find an answer among more or fewer documents. Second, using this value of $\text{top}_{k,\text{Ranker}}$, we grid search for $\text{top}_{k,\text{Retriever}} \in [1, 50]$, with steps of 5. This is done using an SBERT + BigBird-based Ranker pipeline.

**Ranker pipeline evaluation.** Lastly, we evaluate four Ranker pipeline configurations using these hyperparameters. We assess the performance of all combinations of the (a) two overall best Retrievers and (b) the two overall best extractive Readers (from the experiments in Section 5.1.2). We also evaluate Ranker pipelines using the generative Readers in Table 4.4.2, both the zero-shot and one-shot setting. The same prompts are used as in the canonical Retriever-Reader evaluation. Furthermore, as the hyperparameters were found on TriviaQA verified Wikipedia, the results may be overly optimistic on that dataset. Therefore, the evaluation is done on two datasets: TriviaQA verified Wikipedia and Web. Here, F1 is used as the metric.

### 5.2.2 Augmenting answers with Final Evidence Fusion

So far, no information about the relevance of documents fetched by the Retriever (or Ranker) has been incorporated in the final scoring mechanism of the Reader. The Reader simply sees the retrieved contexts as an unordered set. Next, we create a pipeline component, referred to as *final evidence fusion*, which enables us to include information about document relevance in the answer span prediction.

**Final Evidence Fusion.** *Final evidence fusion* merges the Reader-predicted answer span score $S_{i,d}$ of span $i$ with the relevance scores $R_d$ of the document $d$ from which the span $i$ is extracted, as proposed by Ma et al. (2022). The authors observe modest improvements when including scores from the Retriever in the final prediction score. In contrast to the authors, we use the relevance score from the Ranker as these should give a more precise measure of document relevance. Formally, we assign the final score $f_{i,d}$ of a span to the *final evidence fusion score*

$$f_{i,d} = w_{\text{span}} S_{i,d} + w_{\text{document}} R_d, \tag{5.1}$$

a simple linear combination. The score weights $w_{\text{span}}$ and $w_{\text{document}}$ for the spans and documents, respectively, are considered hyperparameters. The output from the entire pipeline is then the answer spans with the largest final evidence fusion scores. To comply with official benchmark evaluation scripts, we always output only one answer span. Figure 5.4 shows schematically the architecture of such a pipeline architecture. Here, the Ranker is used both as a document filter and an enhancer of retrieval scores.

**Tuning hyperparameters.** We begin by tuning hyperparameters on TriviaQA verified Wikipedia dev set, with 36,000 Wikipedia documents in the knowledge base. It is necessary to find values of $\text{top}_{k,\text{Ranker}}$ and $\text{top}_{k,\text{Reader}}$ as the optimal ones might not necessarily coincide with those found for the previous pipeline architectures. For instance, when $\text{top}_{k,\text{Ranker}} = \text{top}_{k,\text{Reader}} = 1$, final evidence fusion has no effect. First, we conduct a random search on $\text{top}_{k,\text{Ranker}} \in [1, 10]$ and $\text{top}_{k,\text{Reader}} \in [1, 10]$ while keeping the number of retrieved documents fixed to the best value found in Section 5.2.1. Then we follow Ma et al. (2022) and grid search to find optimal $w_{\text{span}}$ and $w_{\text{document}}$, using $w_{\text{span}} = 1$ and search for $w_{\text{document}}$ in [0.1, 1] with step size 0.1. In both cases, we optimize for end-to-end F1 score.

**Evaluating pipeline configurations.** Using these hyperparameters, we evaluate in terms of F1 and EM four Ranker pipeline configurations augmented with final evidence fusion. As the hyperparameters have been tuned on the verified Wikipedia dev set, an evaluation on the same dataset may give overly optimistic results. Hence, we run the evaluation on two datasets: TriviaQA's verified Wikipedia dev set and verified Web dev set. The component configurations tested are the same as in the Ranker experiments in Section 5.2.1.

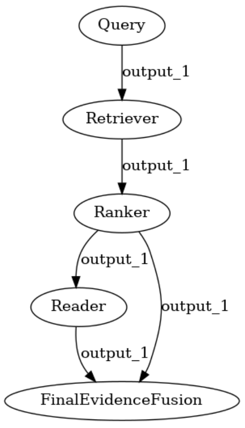


**Figure 5.4:** A Ranker pipeline using Final Evidence Fusion to augment the final span prediction. The final score of an answer span is an aggregation of Reader and Ranker scores.

## 5.3 Evaluating the system in a banking application

Lastly, we evaluate our systems on SEBQuAD, i.e. in a banking setting. The experiment is divided into three parts. First, we conduct a quantitative evaluation on all of SEBQuAD using the most performant pipelines from previous experiments, both from a lexical and semantic point of view. Second, we qualitatively analyze the "incorrect" predictions (in the sense of having zero EM) of a generative pipeline to get a better understanding of where and how it errs. Lastly, we fine-tune an extractive model on SEBQuAD and evaluate its performance after being trained on some pertinent examples.

### 5.3.1 Quantitative evaluation on SEBQuAD.

**Lexical evaluation.** We begin by evaluating a handful of pipelines on the full SEBQuAD dataset. In particular, we evaluate 12 Ranker pipelines, and 4 pipelines employing ranking and final evidence fusion. We assess all combinations of (a) our two best Retriever, (b) our two best extractive Readers, and (c) our two generative Readers, both in the zero-shot and one-shot setting. Again, the prompts are found in Appendix B. We use the same hyperparameters as found in the previous experiments for each pipeline architecture, respectively. For the generative Readers, we use $\text{top}_{k,\text{Ranker}} = 1$. Note that final evidence fusion cannot be directly used for generative Readers – we omit testing those pipeline configurations. Here, F1 is used as the evaluation metric.

**Semantic evaluation.** Next, we quantitatively evaluate 6 Ranker pipelines in terms of semantic answer similarity (SAS). This gives us an idea of the systems' abilities to return semantically correct answers. The evaluation is conducted on SEBQuAD and also on TriviaQA verified Wikipedia and Web for comparability. The pipelines use either any of (a) our two best extractive Readers, (b) or any of the two generative Readers (both zero-shot or one-shot). All pipelines use an SBERT Retriever and an SBERT Ranker. We use the SBERT model `stsb-distilroberta-base` for computing the semantic similarity and use $\text{top}_{k,\text{Retriever}} = 20$, and $\text{top}_{k,\text{Ranker}} = \text{top}_{k,\text{Reader}} = 1$.

Furthermore, we investigate whether SAS metrics extrapolate better than F1 to non-benchmark data. We compare the two metrics' ability to extrapolate to banking data by computing the (Pearson) correlation between scores on benchmark and banking data across pipeline configurations. We compute – for both F1 and SAS scores – the correlation between scores on SEBQuAD and either TriviaQA Wikipedia or Web. We can then compare the average correlation across these two benchmark datasets to see if it is possible to claim that one of the metrics is better for benchmark to non-benchmark performance extrapolation.

### 5.3.2 Qualitative error study.

To get a better intuition behind the performance of the generative models, we perform some analysis on the incorrect predictions. Particularly, we are interested in understanding where the generative models fail, and if the lexical metrics are indeed representative of the model's ability to answer questions. We proceed in two steps and consider only the predictions from GPT-3.5 due to time restrictions. First, we manually look at 6 randomly sampled predictions that are considered incorrect (in the sense of having 0 EM, but may have non-zero F1) to understand in what ways the model is considered to fail. For these predictions, we compare the gold answers and generated answers to see how they differ, and whether they are semantically correct from two independent analysts' points of view. Moreover, we compare these qualitative semantic overlaps with the corresponding F1 and SAS scores. Second, we manually inspect all incorrect (0 EM) predictions and calculate the fraction of the predictions which indeed are semantically correct, but transformed or rephrased in some way. The final, reported set of false negatives is computed as the intersection of the two analysts' assessed sets of false negatives.

### 5.3.3 Fine-tuning an extractive Reader.

Lastly, we fine-tune an extractive Reader on SEBQuAD to see if the model can be improved. We fine-tune the extractive pipeline that performs the best in the previous evaluation. The fine-tuning is set up as follows. First, we split SEBQuAD into a training set (80%) and a validation set (20%). Then, we fine-tune for 10 epochs using a learning rate of 1e-5, batch size of 10, and linear warmup in 5 iterations. Due to time constraints, we do not do extensive experimentation with the training setup. Finally, the fine-tuned model is evaluated on the held-out validation set.

## 5.4 Miscellaneous experiment details

**Preprocessing documents.** The documents provided by the datasets at hand require some preprocessing, particularly due to their lengths (as described in Sections 3.3 and 3.4) being longer than the token limit for many of the Retrievers. In particular, the SBERT Retriever we use has a token limit of 256. Although certain extractive Readers that we use (such as BigBird) either accept and are performant on much longer input sequences, or can use a sliding-window approach for reading longer documents, we follow Luan et al. (2021) and split documents into smaller chunks when documents are too long. Additionally, FLAN-T5 only accepts a maximum of 512 input tokens and cannot use a sliding window. The authors show that the recall of many dense Retrievers deteriorates on the *Natural Questions* dataset when sequence lengths are too long (around 400 tokens), likely due to the compression of documents into low-dimensional fixed-sized vectors.

Thus, we conduct preprocessing mostly to improve retrieval rather than the reading comprehension task.

For Natural Questions, we use the conversion script provided by Kwiatkowski et al. (2019), outlined in Section 3.3, resulting in passages that are on average around 240 tokens long and do contain the short answer to a question in the dataset. We note that this simplifies the task of the dataset as redundant information is pruned from the dataset. This also means that we do not have to bother with processing HTML tokens and allows us for working with cleaner text data. We consider this simplification of the dataset a reasonable intervention for the scope of this project. As there is no supplied preprocessed version of TriviaQA, we split each document into chunks of 200 words with a 30-word overlap between adjacent chunks. The overlap helps to not miss out on answers that can be found in the seam between two chunks. We do not preprocess SQuAD as the average document length is merely around 130 tokens. These preprocessing steps are conducted for each of the aforementioned experiments.

**Answer scope.** Throughout all experiments, we only compare answers against the gold answer. We do not account for which document the answer was found and subsequently do not require the answer to come from the gold document. This is a less strict form of evaluation compared to additionally requiring that a correctly predicted answer should reside in the gold document to be deemed correct.

**Document stores.** For all dense retrievers, we use a FAISS-based document store. Due to our datasets being relatively small (at most around 200,000 documents, depending on the dataset) we use an index with uncompressed (non-quantized) vectors and which does not require any training to construct (i.e. a so-called *flat* index). When using BM25, we use a document store with an Elasticsearch backend.

**Implementation.** All of the experiments are implemented using the packages `farm-haystack` by *deepset* and `transformers` by *HuggingFace*, both indexed on PyPI. Experiments are run on Nvidia T4 or P100 cards.

# Chapter 6

# Results

We present the results of the aforementioned experiments. First, we evaluate in Section 6.1 different configurations of canonical Retriever-Reader pipelines on popular benchmark datasets. This includes finding good pipeline hyperparameters, and testing different Retrievers and of both extractive and generative Readers. Second, in Section 6.2 we show the performances of pipelines that have been augmented with a Ranker and final evidence fusion. Lastly, Section 6.3 covers the resulting performance of pipelines evaluated on the SEBQuAD dataset: a real banking application.

## 6.1 Evaluating configurations of a Retriever-Reader pipeline

In this section, we present the results from the experimentation with canonical Retriever-Reader systems as outlined in Section 5.2. We begin by exploring the effect that pipeline hyperparameters have on the components' performances and then use these results when evaluating different configurations of simple Retriever-Reader OpenQA systems.

### 6.1.1 Choosing pipeline hyperparameters

Here, we explore the impact of varying the number of retrieved documents $\text{top}_{k,\text{Retriever}}$ on (1) the Retriever and (2) the pipeline as a whole; this was outlined in Section 5.1.1. Unsurprisingly, we find that a large $\text{top}_{k,\text{Retriever}}$ increases Retriever recall, however, at the cost of end-to-end performance due to the Reader being "confused" when fed too many documents. Our experimentation with hyperparameter tuning also suggests that dense Retrievers are suitable for the NQ dataset. Conversely, sparse retrievers appear to perform better on SQuAD Open.

## Tuning $\text{top}_{k,\text{Retriever}}$: Retriever's perspective

Having a high Retriever recall is crucial for OpenQA systems to be effective in answering questions. This is because the likelihood of the system returning the correct answer to a question is closely tied to the Reader's accessibility to the golden document. As such, by optimizing for Retriever recall, we increase the likelihood that the system will be able to access and analyze the most relevant information in order to produce accurate and useful answers. Figure 6.1 displays how the recall of the Retrievers varies with the number of fetched documents on two QA datasets.

**NQ.** Figure 6.1a shows the Retriever recall on Natural Questions. For NQ, DPR obtains the highest recall for every value of $\text{top}_{k,\text{Retriever}}$ in the interval. This is intutive as DPR is partly fine-tuned on Natural Questions, and agrees with the findings of Ma et al. (2022). DPR is tightly followed by SBERT, lagging behind with around 5 points for $\text{top}_{k,\text{Retriever}} < 15$, and BM25 is behind SBERT with a similar amount. However, only DPR and SBERT manage to get a recall significantly over 90% for the $\text{top}_{k,\text{Retriever}}$ in the interval. Unsurprisingly, we note that the recall is strictly increasing for all retrievers when more documents are increased.

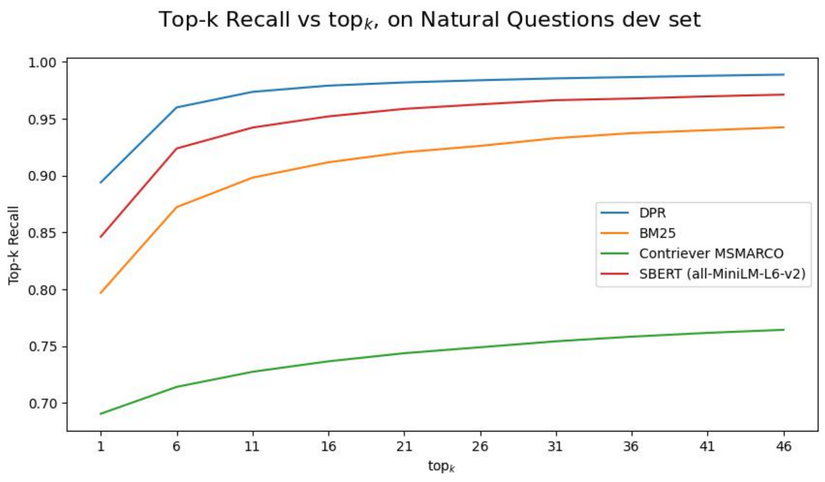
**SQuAD.** Figure 6.1b shows the results from the same experiment but now on SQuAD Open. BM25 and SBERT have significantly better recall than both DPR for low values of $\text{top}_{k,\text{Retriever}}$, and for all $\text{top}_{k,\text{Retriever}}$ compared to Contriever. These results are congruent with those of Ma et al. (2022) and indicate that certain unsupervised or self-supervised Retrievers, such as BM25 and SBERT, perform better on datasets not seen during training compared to supervised counterparts (such as DPR). However, the fact that Contriever lags so far behind all other Retrievers might indicate that such a claim requires further analysis.

## Tuning $\text{top}_{k,\text{Retriever}}$: end-to-end perspective

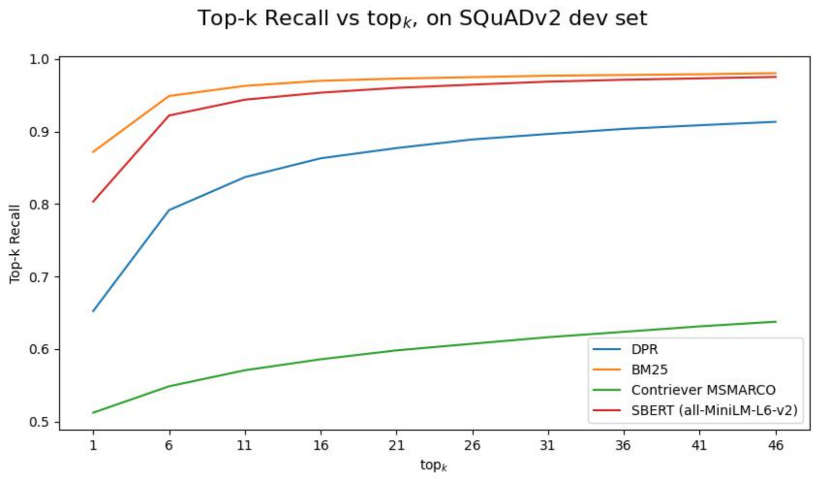
A large $\text{top}_{k,\text{Retriever}}$ appears to improve the Retriever recall at the cost of end-to-end performance. Figure 6.2 displays the end-to-end F1 score on TriviaQA Wikipedia for different values of $\text{top}_{k,\text{Retriever}}$. In the domain $\text{top}_{k,\text{Retriever}} \in [1, 5]$ the end-to-end F1 score increases to its maximum value. This is likely caused by the increased probability of the Reader receiving the gold document. However, the end-to-end performance deteriorates as $\text{top}_{k,\text{Retriever}}$ is increased further. This can be interpreted as the Reader being "confused" when being fed more, lower-quality documents. We hypothesize that this confusion stems from the confident (and in principle correct) prediction of empty spans in the irrelevant documents. Due to the high scores of these empty spans, it is likely that these empty answers will constitute the final output of the system. Although the gold document might indeed be among the retrieved documents, the gold span might receive a lower score than these empty spans. This implies that there is a trade-off between retrieving more documents, increasing the likelihood of retrieving the gold document, and disorienting the Reader with many irrelevant documents. Since we are ultimately interested in end-to-end performance, we use $\text{top}_{k,\text{Retriever}} = 5$ in subsequent experiments.

## Allowing for multiple answers

Next, we investigate the relationship between the Reader's performance and the number of returned answers $\text{top}_{k,\text{Reader}}$. Figure 6.3 shows this for three pipeline configurations on TriviaQA verified Wikipedia. Here, we use a fixed value of $\text{top}_{k,\text{Retriever}} = 5$. When only one answer is allowed,



**(a)**



**(b)**

**Figure 6.1:** Retriever recall vs number of documents retrieved, $\text{top}_{k,\text{Retriever}}$, for a set of retrievers on the development set of (a) Natural Questions, and (b) SQuADv2. All of the Retrievers have an increasing recall with $\text{top}_{k,\text{Retriever}}$, although only BM25 and SBERT manage to attain at least 90% recall with less than 45 retrieved documents on Natural Questions. Only SBERT attains at least 90% recall on both datasets with $\text{top}_{k,\text{Retriever}} < 6$.

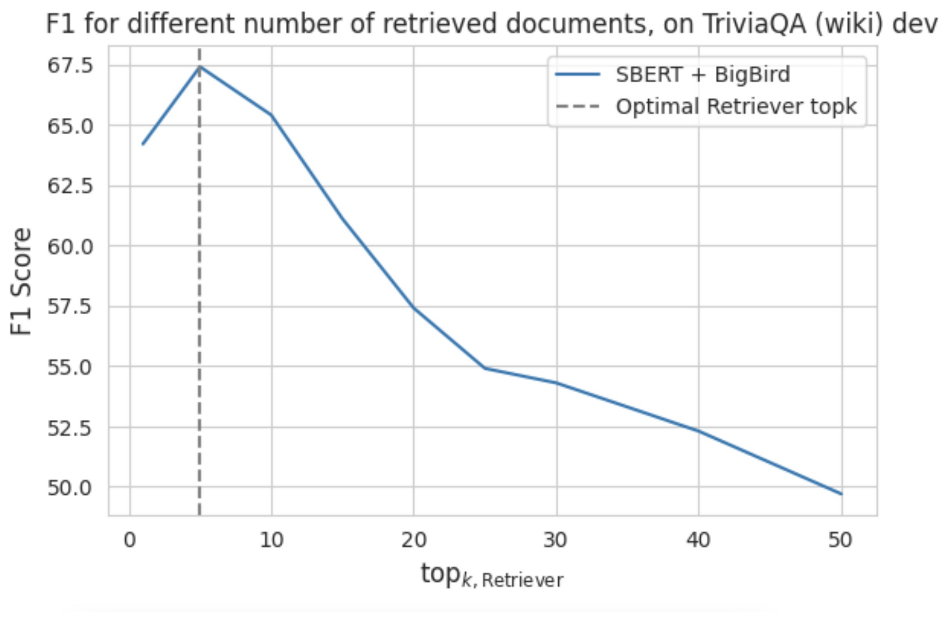


**Figure 6.2:** End-to-end F1 score when varying $\text{top}_{k,\text{Retriever}}$ on TriviaQA dev set. The pipeline uses an SBERT Retriever and BigBird-based Reader. The performance is maximal at around $\text{top}_{k,\text{Retriever}} = 5$, and deteriorates when more documents are retrieved.

both DPR-based pipelines have almost identical performances, while the pipeline using SBERT has performs significantly better. All pipelines are improving significantly when the number of answers permitted is increased. The SBERT+BigBird-based pipeline attains SOTA performance when three answers are returned, and continues to improve (although with diminishing returns) when more answers are returned. While the score of the DPR+DeBERTa-based pipeline stagnates at around 5 answers, the score of the DPR+BigBird-based pipeline continues to increase even at 8 answers and is at that point only 2 points behind the SOTA result (see Table C.1). Thus, although depending on the architecture used, it is possible to attain SOTA performance with a simple Retriever-Reader pipeline if more than one answer is permitted to be returned from the system. Note that the official SOTA performance is obtained using only one answer as the output.

### 6.1.2 Canonical Retriever-Reader pipeline performances

Using the optimal hyperparameters, we show the performance of canonical Retriever-Reader pipelines on four benchmark datasets. We find that it is possible to reach close to state-of-the-art performance using canonical Retriever-Reader pipelines. While extractive Readers seem to be sufficient for SQuAD Open and NQ, TriviaQA appears to be more challenging for these pipelines. Instead, one-shot generative Retriever-Reader systems perform better on TriviaQA but still fall behind the SOTA. Moreover, we observe that SBERT and BM25 generalize best to multiple benchmark datasets.

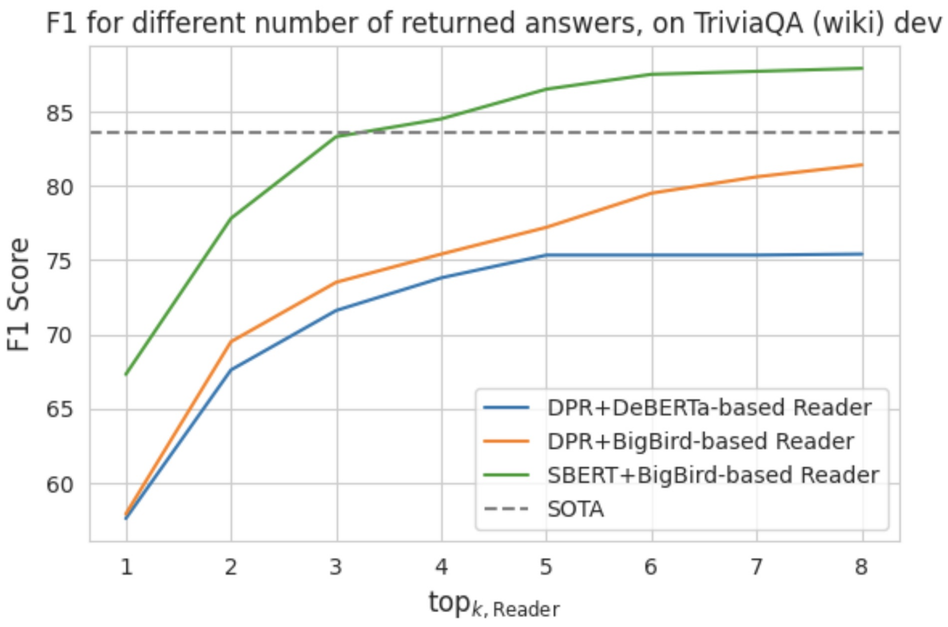


**Figure 6.3:** F1 score versus the number of returned answers for three pipelines. The performances are improving when top$_{k,\text{Reader}}$ is increased for all pipelines. A fixed value of top$_{k,\text{Retriever}}$ = 5 is used.

## Encoder-based Readers

Table 6.1 shows the performances of a set of Retriever-Reader pipeline configurations, as measured in F1 score. Each sub-table displays the performance on each of the four benchmark datasets. Here, all pipelines are run with top$_{k,\text{Retriever}}$ = 5 (from the previous section's result) and top$_{k,\text{Reader}}$ = 1 to comply with official evaluation scripts. The performance of the different Retriever-Reader pipeline configurations varies significantly across the different benchmark datasets, indicating that the optimal configuration is dataset-specific.

Unsurprisingly, Readers in most cases perform better on the datasets they have been fine-tuned on. However, this behavior is not as clear as one would have guessed. For example, most of the pipelines using a dense Retriever, coupled with Readers fine-tuned on TriviaQA or SQuAD, achieve close to SOTA results on NQ. One possible explanation is that a significant proportion of the NQ dataset consists of unanswerable questions, and both SQuAD and TriviaQA contain a large number of such questions. Therefore, Readers fine-tuned on these datasets may have learned to identify and handle unanswerable questions effectively, which could generalize well to the NQ task. It is also evident that the Reader fine-tuned on NQ achieve relatively poor results on the other datasets. However, this might be a consequence of the fact that the models are directly fetched from HuggingFace – meaning we are not guaranteed that the models have been optimally fine-tuned.

Another key observation – which was hinted at in Figure 6.1 – is that DPR appears to generalize poorly to datasets that are not NQ, in agreement with Ma et al. (2022). Similarly, the Contriever performance on datasets other than NQ is by far the worst, hinting at poor generalization capabilities. Note that SBERT and BM25 are the Retriever architectures that perform the best in the aggregate. This suggests that these Retriever architectures are more robust and with greater generalization capabilities. Thus, in the following experiments, only BM25 and SBERT will be used.

| | | Reader fine-tuned on | | |
|---|---|---|---|---|
| **Dataset** | **Retriever** | SQuAD | NQ | TriviaQA |
| **SQuAD Open** | DPR | 59 | 36 | 49 |
| | BM25 | **72**† | 51 | 50 |
| | Contriever MS MARCO | 50 | 36 | 31 |
| | SBERT | 68 | 38 | 50 |
| **NQ (simplified)** | DPR | 62† | **63**† | 58 |
| | BM25 | 45 | 40 | 60† |
| | Contriever MS MARCO | 58† | 33 | **63**† |
| | SBERT | 52 | 42 | 59 |
| **TriviaQA Wikipedia (verified)** | DPR | 58 | 47 | 58 |
| | BM25 | 63 | 56 | 64 |
| | Contriever MS MARCO | 12 | 9 | 12 |
| | SBERT | 65 | 54 | **68** |
| **TriviaQA Web (verified)** | DPR | 51 | 35 | 49 |
| | BM25 | **65** | 47 | 63 |
| | Contriever MS MARCO | 10 | 10 | 11 |
| | SBERT | 64 | 50 | 61 |

**Table 6.1:** Open domain end-to-end F1 scores for different extractive pipeline configurations. All experiments are run with $\text{top}_{k,\text{Reader}} = 1$ and $\text{top}_{k,\text{Retriever}} = 5$. We only display the result for *best* performing Reader fine-tuned on each of the datasets. The configuration performing best on each dataset is marked in bold font. The symbol † indicates that the results are within 10% of the state-of-the-art (SOTA) results. SOTA results for each dataset are displayed in Table C.1.

Table 6.2 summarizes the best-performing pipeline architectures across the benchmarks. The results indicate that near-SOTA results can be achieved using the canonical extractive pipeline. This holds for the SQuAD Open and NQ datasets, however, TriviaQA seems to be more of a challenge. Thus, we focus on TriviaQA in the subsequent experiments.

## Generative Readers

Table 6.3 shows the performance of generative Retriever-Reader pipelines. The improvement in performance between zero-shot generative pipelines and the extractive pipelines is merely at most 3 points. However, in the one-shot setting, a greater performance lift is observed.

The results in Table 6.3 also suggest that the encoder-decoder-based FLAN-T5 Reader outperforms the decoder-based GPT-3.5 in the zero-shot setting. It is likely that this disparity stems from the difference in training objective between the two models. As Chung et al. (2022) mention, FLAN-T5 is fine-tuned on a mixture of tasks, including question-answering on TriviaQA. In contrast, GPT-3.5 has not been fine-tuned on question-answering (Floridi and Chiriatti, 2020). However,

| Dataset | Retriever | Reader | F1 | EM |
|---|---|---|---|---|
| SQuAD Open | BM25 | DeBERTA | 72† | 69† |
| NQ (Simplified) | DPR | DistilBERT | 63† | 62† |
| TriviaQA (Wikipedia) | SBERT | BigBird | 68 | 64 |
| TriviaQA (Web) | BM25 | DeBERTA | 65 | 57 |

**Table 6.2:** Best performing encoder-based pipelines on benchmark datasets. Every Reader is fine-tuned on the corresponding training split of each respective dataset. The symbol † indicates that the results are within 10% of the state-of-the-art (SOTA) results. SOTA results for each dataset are displayed in Table C.1.

| | | Reader | | | |
|---|---|---|---|---|---|
| **Dataset** | **Retriever** | FLAN-T5 | FLAN-T5 (one-shot) | GPT-3.5 | GPT-3.5 (one-shot) |
| **TriviaQA Wiki (verified)** | BM25 | 60 | 58 | 63 | 69 |
| | SBERT | 69 | 64 | 64 | **71** |
| **TriviaQA Web (verified)** | BM25 | 68 | 65 | 64 | 75 |
| | SBERT | 65 | 64 | 66 | **77** |

**Table 6.3:** F1 scores for generative pipelines. We use $\text{top}_{k,\text{Retriever}} = 1$ for all pipelines, and report only the performance in both the zero-shot and one-shot settings. The exact prompts used are found in Appendix B. The colors represent different Reader types: encoder-decoder-based, and decoder-based.

the dynamics are different when the models are provided with an example of the task to solve at inference time. While FLAN-T5's performance deteriorates in the one-shot setting, GPT-3.5 improves between 10-15% on both datasets. This aligns with previous research (Radford et al., 2019; Floridi and Chiriatti, 2020) and is expected as GPT-3.5 is specifically designed to be a good few-shot learner.

**Size improves performance.** To better understand the correlation between model size and performance, FLAN-T5s of different sizes are evaluated on TriviaQA Wikipedia. Figure 6.4 shows the end-to-end F1 score for a DPR pipeline using different sizes of FLAN-T5. The results suggest that a larger language model increases performance. However, this increase in performance is subject to diminishing returns. Still, as `FLAN-T5-xl` is the best-performing model version (that fits on our hardware), we continue to use this version in all of our subsequent experimentation.

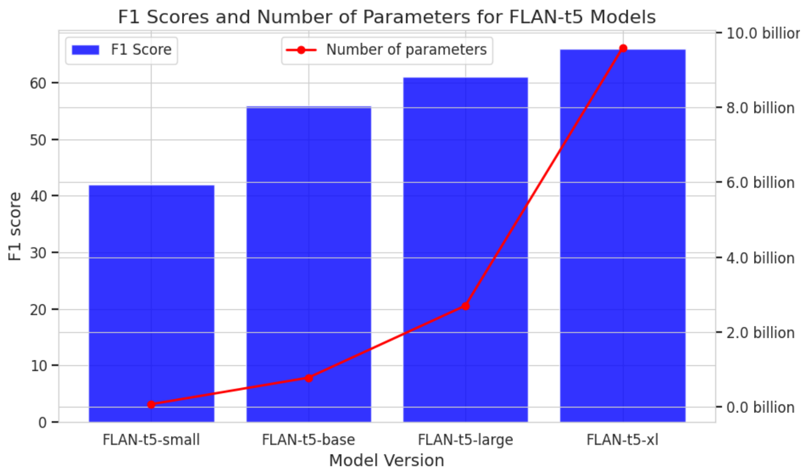


**Figure 6.4:** F1 score for different sizes of the generative `FLAN-T5` Reader on the TriviaQA verified Wikipedia dev set. The Retriever used is DPR. Only the documents from the TriviaQA dev set are fed into the document store. The left veritcal axis shows the F1 score of the system; the right vertical axis shows the number of parameters of the model used. The horizontal axis contains each of the models tested.

## 6.2 Augmenting the pipelines

In this section, we present the results for the pipeline augmentation outlined in Section 5.3. We begin, in Section 6.2.1, by examining the impact of augmenting the canonical pipeline architecture with a Ranker node. Second, in Section 6.2.2, we explore the effect of using final evidence fusion (FEF). We find that a Ranker node increases end-to-end performance by increasing precision in the retrieval system. We also find that FEF in some cases increases performance, but that it is prone to overfitting.

### 6.2.1 Filtering retrieved documents using a Ranker

Here we show the performance of Ranker-augmented pipelines, as outlined in Section 5.2.1. We find that the inclusion of a Ranker node mitigates the confusion effect (as identified in Section 6.1.1), and as a consequence increases the performance of both generative and extractive pipelines.

**The confusion domain.** The inclusion of a Ranker node appears to mitigate the effect of Reader confusion, appearing when the Reader is fed many documents. Figure 6.5 shows how end-to-end performance varies when changing $\text{top}_{k,\text{Retriever}}$ for pipelines either *with* or *without* a Ranker node, respectively. Here, $\text{top}_{k,\text{Ranker}}$ is fixed to 1. Again, it is clear that the pipeline *without* a Ranker suffers from confusion. In contrast, the Ranker of the pipeline *with* a Ranker seems to help mitigate the confusion, yielding a superior end-to-end performance for all values of $\text{top}_{k,\text{Retriever}}$. Furthermore, the performance of the augmented pipeline increases faster in the domain $\text{top}_{k,\text{Retriever}} \in (1, 5]$, indicating that the existence of the Ranker helps the model to better capture the increase in Retriever recall. It is also noteworthy that retrieving much more than say

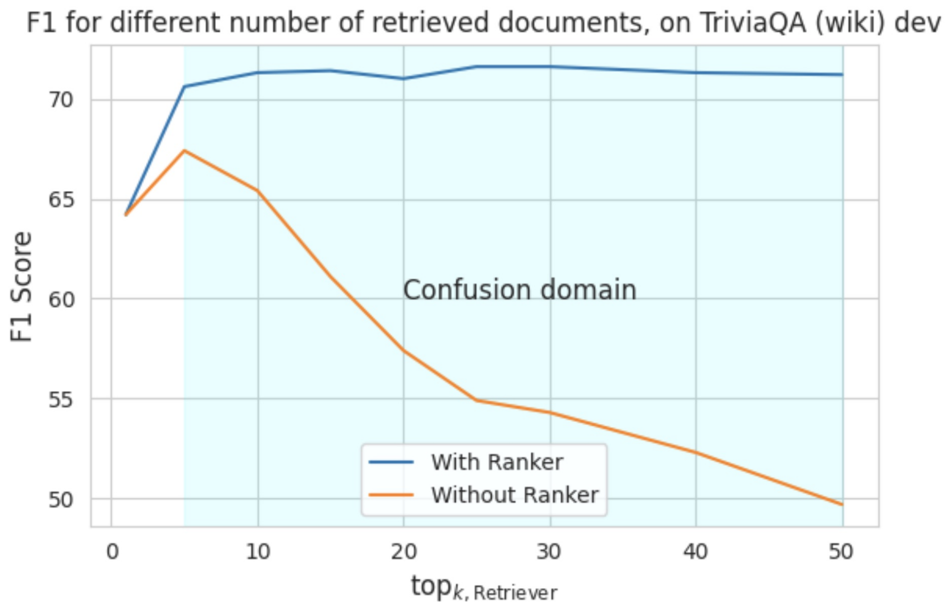


**Figure 6.5:** End-to-end F1 score for pipelines *with* and *without* a Ranker component, respectively. Both pipelines use an SBERT Retriever and BigBird-based Reader. Here, $\text{top}_{k,\text{Ranker}} = 1$. The line *without Ranker* is identical to the one from Figure 6.2. In contrast to the pipeline without a Ranker, the Ranker-based pipeline improves up until about 15 retrieved documents, and remains at the same performance when $\text{top}_{k,\text{Retriever}}$ is further increased. The Ranker seems to prevent the Reader from getting confused when more documents are retrieved.

15 documents does not affect the end-to-end performance in the presence of a Ranker.

**Ranker boosts precision.** A possible explanation for why the Ranker helps to alleviate Reader confusion is as follows. Since the Reader is sensitive to low-quality documents, Retriever recall is not the most suitable metric to compare the end-to-end performance against. Instead, precision more accurately accounts for both (1) the *quality* of documents and (2) the *number* of documents fed to the Reader. Figure 6.6 shows the precision and recall against the number of fetched documents, for both a pure Retriever and a Retriever-Ranker pipeline. Here, $\text{top}_{k,\text{Ranker}}$ is fixed to 1. While the Retriever recall converges to 1 as $\text{top}_{k,\text{Retriever}}$ increases, its precision decreases significantly. This means the proportion of relevant documents fed to the Reader decreases. In contrast, the Ranker's precision (which coincides with the recall as $\text{top}_{k,\text{Ranker}} = 1$) is kept stable around a value of 0.7, implying that its ability to filter out a single relevant document from the retrieved documents is at a reasonable level and does *not* deteriorate when more document are fetched. Although the likelihood of feeding the Reader a gold document decreases in the presence of a Ranker, the deterioration from this effect is less prominent compared to the deterioration arising from Reader confusion. It is also clear that the precision curves correlate better than the recall curves with the end-to-end F1 in Figure 6.5. Hence, given that Readers suffer from confusion, a Ranker stabilizes precision and therefore improves end-to-end performance.

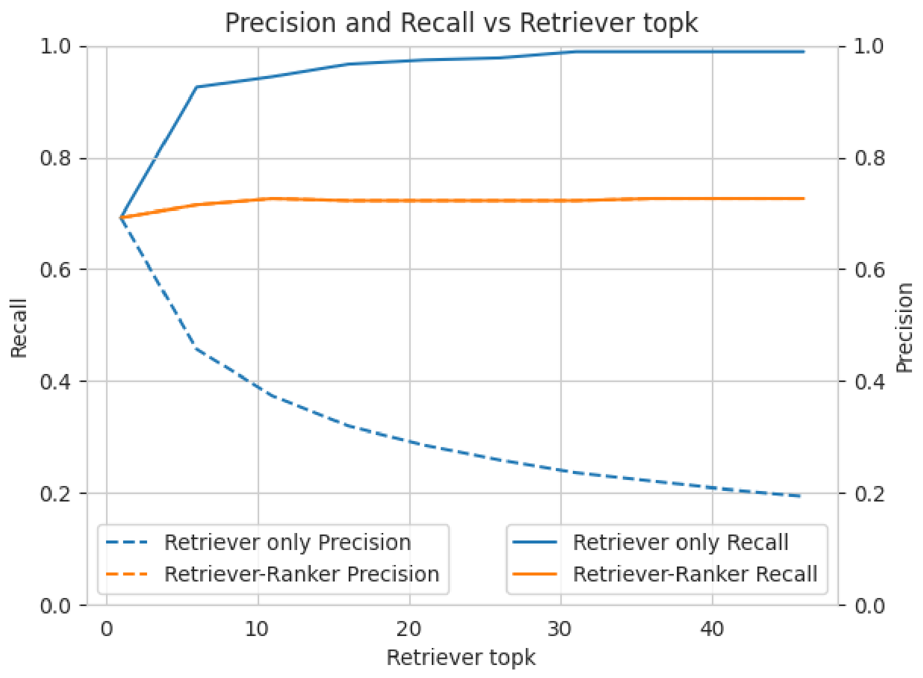


**Figure 6.6:** Precision and recall for Retriever and Ranker (in a Retriever-Ranker pipeline) across values of $\text{top}_{k,\text{Retriever}}$. $\text{top}_{k,\text{Ranker}}$ is fixed to 1. Although the Ranker's recall is lower than that the Retriever, it helps keep the precision stable. Note that the Ranker's precision and recall coincides as $\text{top}_{k,\text{Ranker}} = 1$.

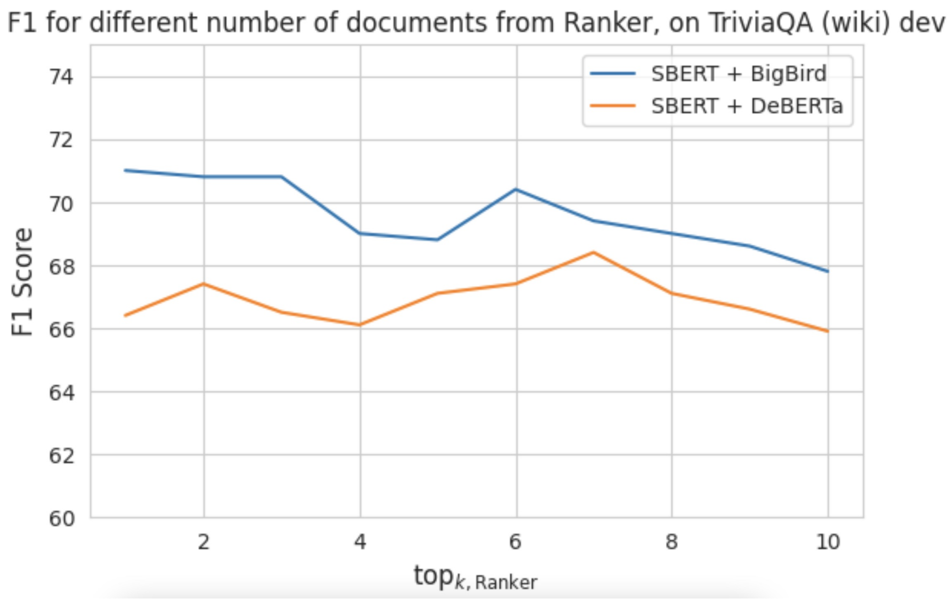


**Figure 6.7:** End-to-end F1 for different number of documents passed on from the Ranker to the Reader. The performance is quite stable for both pipelines; however, the maximal score is obtained at top$_{k,\text{Ranker}}$ equal to 1 and 7 for the BigBird and DeBERTa-based Readers, respectively.

**Tuning top$_{k,\text{Ranker}}$.** Unsurprisingly, the number of documents top$_{k,\text{Ranker}}$ fed to the Reader in the Ranker pipeline impacts the end-to-end performance. Figure 6.7 shows the end-to-end F1 scores, using two different Readers, on TriviaQA. The plot indicates that the optimal top$_{k,\text{Ranker}}$ appears to be architecture specific. For the BigBird-based Reader, top$_{k,\text{Ranker}} = 1$ appears to be optimal. Conversely, the DeBERTa-based Reader appears to perform the best on TriviaQA when top$_{k,\text{Ranker}} = 7$.

**Ranker boosts end-to-end performance.** Augmenting the best-performing pipelines from Section 6.1.2 with a Ranker node improves performance. Table 6.4 displays the end-to-end F1 scores on TriviaQA using the best-performing extractive and generative Readers. Compared to the canonical Retriever-Reader pipelines, the Ranker boosts the performance by 3-10 F1 points for all configurations. The performance increases on both the Wikipedia and Web versions of TriviaQA, however, the increase is more significant on the Web split. Moreover, the generative pipelines appear to benefit more from a Ranker node compared to the extractive ones. The improvement is especially large for the zero-shot setting of FLAN-T5 Reader and the one-shot setting of the GPT-3.5 Reader. Similarly to the generative Retriever-Reader pipelines, GPT-3.5 benefits from being provided gold examples at inference time while FLAN-T5 does not.

### 6.2.2 Augmenting answer scores to improve span prediction

Here we outline the results when further augmenting the extractive Ranker pipeline through final evidence fusion (FEF), as outlined in Section 5.2.2. We find that FEF can improve performance, but that this performance increase is highly dependent on hyperparameter tuning making it non-robust

| Datasets | Retriever | Reader: BigBird | DeBERTa | FLAN-T5 | FLAN-T5 (one-shot) | GPT-3.5 | GPT-3.5 (one-shot) |
|---|---|---|---|---|---|---|---|
| **TriviaQA Wiki (verified)** | BM25 | 70 | 67 | 74 | 69 | 63 | 76 |
| | SBERT | 71 | 68 | 75 | 71 | 64 | **78** |
| **TriviaQA Web (verified)** | BM25 | 75 | 66 | 71 | 67 | 65 | **83**† |
| | SBERT | 75 | 66 | 71 | 68 | 66 | **83**† |

**Table 6.4:** F1 scores for Ranker pipelines. The configurations correspond to the two overall best Retrievers and Readers from the canonical Retriever-Reader testing in Section 6.1.2. The pipelines are evaluated on the TriviaQA Wikipedia (verified) dev set using $\text{top}_{k,\text{Retriever}} = 20$ and $\text{top}_{k,\text{Ranker}}$ equal to 1 and 7 for BigBird and DeBERTa based Readers, respectively. The generative models use $\text{top}_{k,\text{Ranker}} = 1$. The best scores are marked in bold fond. The symbol † indicate that the results are within 10% of the SOTA results found in Table C.1. The colors represent different Reader types: encoder-based, encoder-decoder-based, and decoder-based.

| FEF Hyperparameters | |
|---|---|
| $\text{top}_{k,\text{Retriever}}$ | 20 |
| $\text{top}_{k,\text{Ranker}}$ | 9 |
| $\text{top}_{k,\text{Reader}}$ | 3 |
| $w_{\text{span}}$ | 1 |
| $w_{\text{document}}$ | 0.2 |

**Table 6.5:** Optimal hyperparameters for the FEF-pipeline from tuning on TriviaQA Wikipedia.

in our application.

**Tuning final evidence fusion.** Table 6.5 shows the optimal FEF hyperparameters found by the tuning scheme presented in Section 5.2.2. As previously mentioned, the Reader must output multiple documents to be fused in the FEF node for the technique to have any effect; this results in an optimal $\text{top}_{k,\text{Reader}} > 1$. Also, the optimal FEF weights attribute a larger weight to the span scores than to the document relevancy, $w_s > w_d$, meaning that span scores are more meaningful than document relevancy in the context of span classification. However, as $w_d$ is non-zero, the inclusion of document relevancy scores seems to be useful for improving performance on TriviaQA Wikipedia.

**FEF hyperparameters are prone to overfitting.** Table 6.6 shows the end-to-end performance of the FEF pipelines, using the tuned hyperparameters. The results indicate that FEF is a non-robust technique. On TriviaQA Wikipedia, the best pipeline experiences a 9% relative improvement in both F1 and EM, compared to pure Ranker pipelines. However, the Web performance

| Dataset | Reader / Retriever | F1 Score BigBird | F1 Score DeBERTa | Exact match BigBird | Exact match DeBERTa |
|---|---|---|---|---|---|
| **TriviaQA Wiki (verified)** | BM25 | 70 | 69 | 67 | 62 |
| | SBERT | **74** | 69 | **70** | 62 |
| **TriviaQA Web (verified)** | BM25 | 66 | 66 | **63** | 60 |
| | SBERT | **67** | 65 | **63** | 59 |

**Table 6.6:** F1 and EM scores for pipelines employing Ranking and Final Evidence Fusion. All pipelines use $\text{top}_{k,\text{Retriever}} = 20$, $\text{top}_{k,\text{Ranker}} = 9$, and $\text{top}_{k,\text{Reader}} = 3$. Only one answer is output from the Final Evidence Fusion node. The best-performing pipeline configuration on each dataset is marked with bold font; we achieve a 9% improved performance (in terms of both F1 and EM) compared to our best canonical Retriever-Reader pipeline on TriviaQA verified Wikipedia.

is 11% lower compared to that of the pure Ranker pipeline. As the Wikipedia split of TriviaQA was used for hyperparameter tuning, this result is perhaps unsurprising. The fact that the performance is worse on the Web split – the hold-out dataset for hyperparameter tuning – indicates that the FEF-hyperparameters have been overfitted to the Wikipedia split. Hence, it appears that FEF is a non-robust technique that has to be used with caution. Moreover, the end-to-end performance does not generally vary a lot when changing Retriever from BM25 to SBERT. This is true for all configurations except for the BigBird-based Reader on TriviaQA Wikipedia.

## 6.3 Evaluating the system in a banking application

Here we present the results from the qualitative and quantitative evaluation of the OpenQA systems in the banking setting, as presented in Section 5.3. We find that the best-performing systems generalize moderately well to the SEBQuAD task. Furthermore, we find that SAS on benchmark data seems to be a better indicator of banking data performance compared to F1. Moreover, we find that the performance of generative pipelines is sometimes poorly described by F1 and EM metrics in the banking application. Finally, we show that further fine-tuning an extractive Reader on the SEBQuAD dataset has no effect on performance.

### 6.3.1 Quantitative evaluation on SEBQuAD

First, we show the quantitative results on SEBQuAD in terms of lexical metrics. Then, we compare the performance in terms of semantic answer similarity between benchmark datasets and banking data.

| | | Reader | | | | | |
|---|---|---|---|---|---|---|---|
| **Pipeline type** | **Retriever** | BigBird | DeBERTa | FLAN-T5 | FLAN-T5 (one-shot) | GPT-3.5 | GPT-3.5 (one-shot) |
| **Ranker** | BM25 | 31 | **59** | 53 | 58 | 45 | 55 |
| | SBERT | 32 | 58 | 53 | 58 | 47 | 56 |
| **Ranker + FEF** | BM25 | 20 | 54 | - | - | - | - |
| | SBERT | 19 | 53 | - | - | - | - |

**Table 6.7:** F1 scores on the full SEBQuAD dataset. For extractive Ranker pipelines, $\text{top}_{k,\text{Retriever}} = 30$, and $\text{top}_{k,\text{Ranker}} = \text{top}_{k,\text{Reader}} = 1$. GPT-3.5 uses $\text{top}_{k,\text{Ranker}} = 3$. For Final evidence fusion pipelines the settings in Table 6.5 are used. Final evidence fusion cannot be applied to generative readers. If "one-shot" is not specified, we assume the zero-shot setting. The colors represent different Reader types: encoder-based, encoder-decoder-based, and decoder-based.

**Lexical evaluation.** The quantitative evaluation on SEBQuAD shows that the performances do not correlate perfectly with the performance on the benchmark datasets. Table 6.7 shows the F1 scores on SEBQuAD for both pure Ranker pipelines, and Ranker pipelines augmented with FEF.

Although the overall performances on SEBQuAD is lower compared to the benchmark datasets, most pipelines appear to generalize moderately well. The performance decreases by 25-29% compared to our best systems' benchmark performances. The performance is stable across Retriever types, indicating that both sparse and dense retrieval architectures work in the banking setting. Both extractive and generative Readers appear to be satisfactory for this application. For example, both the DeBERTa-based Reader and one-shot FLAN-T5 achieve F1 sores close to 60. In contrast to the behavior on benchmark datasets, FLAN-T5 benefits from examples at inference time on SEBQuAD. Surprisingly, BigBird – the best extractive Reader on both TriviaQA datasets – performs between 29-46% worse compared to the other pipelines. By the same token, GPT-3.5 – previously achieving the best results on both TriviaQA datasets – lags behind in terms of performance. This holds for both the zero-shot and one-shot settings. Furthermore, the use of FEF deteriorates performance on the SEBQuAD dataset when using hyperparameters tuned on the TriviaQA Wikipedia dataset. Again, this indicates on the lack of robustness of FEF.

**Semantic evaluation.** Table 6.8 shows the performance in terms of semantic answer similarity of several Ranker pipelines across both benchmark and banking datasets. All pipelines attain lower SAS scores on SEBQuAD compared to the benchmark datasets – the same behaviour we as observed in terms of F1. On SEBQuAD, there is a tie between extractive and decoder-based Readers. This contrast the previous result; in terms of lexical metrics, the best extractive Reader slightly outperformed the generative models. Furthermore, GPT-3.5 outperforms FLAN-T5 in terms of SAS on SEBQuAD, while FLAN-T5 is better in terms of F1. The fact that both F1 and SAS scores are lower on SEBQuAD indicates that the dataset contains certain characteristics that are difficult for the models to handle, or that the dataset is poorly annotated.

| Dataset | Reader | | | | | |
|---|---|---|---|---|---|---|
| | BigBird | DeBERTa | FLAN-T5 | FLAN-T5 (one-shot) | GPT-3.5 | GPT-3.5 (one-shot) |
| **TriviaQA Wiki** | 71 | 70 | 76 | 72 | 73 | **78** |
| **TriviaQA Web** | 75 | 69 | 72 | 71 | 76 | **84** |
| **SEBQuAD** | 42 | **61** | 47 | 57 | 57 | **61** |

**Table 6.8:** Semantic answer similarity (SAS) score on benchmark and banking datasets. All pipelines use an SBERT Retriever and SBERT Ranker. The best scores are marked with bold font. The colors represent different Reader types: encoder-based, encoder-decoder-based, and decoder-based.

| | SEBQuAD | |
|---|---|---|
| **Benchmark dataset** | **Corr. F1** | **Corr. SAS** |
| TriviaQA Wiki | 0.32 | 0.67 |
| TriviaQA Web | 0.57 | 0.96 |
| **Avg. correlation** | **0.44** | **0.82** |

**Table 6.9:** Correlation between scores on benchmark datasets and SEBQuAD across 6 pipeline configurations, in terms of either F1 or semantic answer similarity. The correlation between SAS scores is greater than that of F1. The pipelines are the same as those found in Table 6.8.

Next, we compute the Pearson correlation between benchmark and banking dataset scores for different performance metrics. Table 6.9 displays the correlation between scores on SEBQuAD and two benchmark datasets, respectively, for both F1 and SAS. The correlation between SAS scores is greater than the correlation between F1 scores; it is on average close to being twice as large. This indicates that SAS scores on benchmark datasets better describe than F1 a system's ability to generalize to our banking data.

### 6.3.2 Qualitative error study.

The quantitative results on SEBQuAD may be misleading due to the nature of the dataset and evaluation metrics. Therefore, it may be beneficial to qualitatively examine the predicted answers.

**Limitations of lexical metrics.** One might gain additional insight from further examining failures and successes of the pipeline, as covered in Section 5.3.2. Table 6.10 displays six randomly sampled negatives (in the sense of zero EM) from the evaluation of the SBERT+GPT-3.5 pipeline, accompanied by F1 and SAS scores. The colors indicate how closely the scores align with the analysts' assessments.

These examples show how the F1 metric might not be suitable for evaluating the performance of generative models as it fails to capture semantic overlap and the correctness of numerical answers.

| Query | Gold Answer | Generated Answer | F1 | SAS | Analysts' comment |
|---|---|---|---|---|---|
| How often should a manager inform employees about [redacted] | Once a year | Employees should be informed annually | 0 | 30 | |
| How long is the required holding period for selling financial instruments at profit | one-month | One month | 0 | 98 | Correct but rephrased; low F1 but greater SAS. |
| What was the volume produced by [redacted] in the second quarter of 2022 | 110,5 (109,2) thousand tonnes | 110 500 tonnes | 29 | 61 | |
| What was the net profit of [redacted] in October-December | 26 MSEK | 102 MSEK | 50 | 25 | Incorrect number, but non-zero F1 and SAS (although lower). |
| What were the earning per share for [redacted] in the second quarter of 2022 | 1,13 (0,49) SEK | The earnings per share for [redacted] in the second quarter of 2022 was 12.52 SEK | 13 | 9 | |
| Who is the CEO of [redacted] | <Some name> | <A different name> is the CEO of [redacted] | 0 | 2 | Incorrect answer; low F1 and SAS. |

**Table 6.10:** Randomly sampled negatives (0 EM) on SEBQuAD, as produced by a SBERT + GPT-3.5 pipeline. F1, semantic answer similarity (SAS) scores, and analysts' comments on the semantic overlap between the gold and the predicted answer are included. Some words have been censored. Words between <> are placeholders for comparability. The colors represent metrics that align with qualitative assessment well, neutrally, or poorly.

The first three rows of Table 6.10 show predicted answers that are semantically correct, but which due to rephrasing are penalized in terms of F1. Such rephrasing is generally not possible in extractive systems. The SAS score is non-zero and greater than the F1 score for these three correct but rephrased answers; this indicates that SAS better describes the answer correctness. In contrast to TriviaQA, SEBQuAD contains at most one answer for each question; this penalizes rephrasing and thus generative models. This might be one of the reasons why GPT-3.5 – which outperformed the extractive ones on TriviaQA – struggles in terms of F1 on the SEBQuAD dataset. Note that some information in the table is redacted due to confidentiality.

**Failure missclasification.** The examples in Table 6.10 show how some answers that are deemed incorrect (0 EM) might in fact be semantically correct and useful. Here, we let two independent analysts classify these incorrect answers as either *semantically correct* or *semantically incorrect*. From this analysis, we conclude that 59% of the zero-EM answers on SEBQuAD are indeed semantically correct. This further illustrates that lexical metrics might be poor indicators of system performance as it fails to capture semantic overlap.

**The issue with numerics.** Generative systems seem to struggle with inferring correct numerics from the context. In particular, SAS better captures incorrect numerical answers than F1. The fourth and fifth rows of Table 6.10 display how the generative model is indeed capable of understanding that the question *is about* numerics, but fails to return the correct number. Although the answers are incorrect, they have non-zero F1 e.g. due to the correct currency specifier. The corresponding SAS scores are lower for both of these incorrect answer. In contrast to the previous result of F1 penalizing generative models through prohibiting rephrasing, these results instead show how F1 occasionally rewards generative models on numerical questions. The analysts conclude that out of all numeric questions with non-zero F1 score, 61% are numerically incorrect. This adds to the observation of F1 being a poor indicator of system performance, as it fails to correctly capture numerical information. Instead, SAS better captures (although not perfectly) incorrect numerical answers.

**Financial jargon.** Based on the sampled negatives, we observe that the system is capable of understanding financial jargon well. In fact, the model never struggles in any of the examples due to the questions and documents containing banking-specific expressions.

### 6.3.3 Fine-tuning an extractive Reader.

Table 6.11 shows the results from additionally fine-tuning an extractive Reader on a subset of SEBQuAD. The results indicate that fine-tuning has no effect on performance on the corresponding holdout dataset. This likely stems from the SEBQuAD-train dataset being too small (89 samples) and too similar to SQuAD. This effect is further discussed in Section 7.1. The fact that the performance on the SEBQuAD validation split compared to the whole of SEBQuAD (69 vs. 59) is a consequence of the validation set, by chance, containing a larger share of less challenging samples.

| Dataset | Reader | Fine-tuned on | F1 | EM |
|---|---|---|---|---|
| **SEBQuAD-validation** | DeBERTa | SQuADv2 | 69 | 61 |
| | | SQuADv2 + SEBQuAD-train | 69 | 61 |

**Table 6.11:** F1 and EM scores on the holdout (validation) split of the SEBQuAD dataset. Here, a DeBERTa-based Reader is fine-tuned on either SQuAD only, or both SEBQuAD and SQuAD. Further fine-tuning on the (small) SEBQuAD dataset appears to have no effect on banking domain performance.

# Chapter 7

# Discussion

Here, we discuss the results of our experimentation. In Section 7.1 we cover the significance of our results and whether we can draw conclusions based on our findings. Section 7.2 considers restrictions and limitations of our systems. In Section 7.3 we briefly discuss our OpenQA systems from the perspective of production deployment. Lastly, Section 7.4 concerns the ethical and environmental impact of building and executing our systems.

## 7.1 Significance of results

**Performance on benchmark datasets.** It is difficult to make claims about the ability of a system to generalize. In particular, one has to make a trade-off between the number of samples to evaluate a system on, and the cost of doing so. The results of the canonical Retriever-Reader pipeline experimentation showed that the performance of the same configuration varied greatly across the four datasets. In the subsequent experiments, the performances were only evaluated on the two (small) instances of TriviaQA. Although the datasets are of high quality, they compromise less than 1,000 samples in total. Thus there is a high likelihood that the variance of all subsequent performance estimates is quite high. This inhibits our ability to make confident claims about certain architecture being *statistically significant* better than others, particularly as we do not have any measures of variance of our performance metrics.

Additionally, it is important to note that achieving close-to-state-of-the-art (SOTA) performance on Natural Questions (NQ) using a simplified dataset may not be entirely comparable. The simplification of the dataset, outlined in Section 3.3, may result in a less challenging task and thus hinder us from making definite claims about the competitiveness of our system compared to the ones on the NQ leaderboard.

**Performance on SEBQuAD.** Although the performance on SEBQuAD is significantly lower than that of the SOTA on benchmark datasets, it is not evident that it is possible to claim that QA in banking is more difficult than in other domains. Particularly, SEBQuAD is a *very* small dataset in comparison to benchmark datasets; recall that it contains shy of 15% of the number of labels compared to the smallest of our benchmark datasets. This puts demands on the quality of the data annotation procedure: a single low-quality label can alter the final F1 and EM scores by almost a full point. A low-quality label would in this case be a question-context-answer pair where it is not clear that the context *completely* contains sufficient information for the questions. Different models might handle such lack of context information variably well. As such, it is not possible to draw a conclusion about whether extractive or generative Readers are better for the banking setting: the best-performing models vary at most by 3 F1 points. However, the deterioration of performance by at most 29% compared to benchmark datasets strongly indicate that all our systems struggle on SEBQuAD.

**Feasibility of lexical metrics.** It is not clear that we can draw conclusions about the feasibility of lexical metrics in the banking application. This is due to the fact that only a very few negatives were sampled (6), and the analysts' assessment was conducted on only about 70 negatives. Although the results indicate that the lexical metrics are poor indicators, one would have to conduct the experiments on a larger scale to ascertain that the result is not coming from noise.

Furthermore, it is not evident that we can claim that SAS is a better performance metric than F1 on banking data, since our correlation computation was only carried out on 6 pipeline configurations and on two benchmark datasets. Again, we have no measure of variance in our correlation computations, meaning we cannot be certain that the result is not coming by pure chance. While the result indeed indicates that SAS extrapolate better than F1 from benchmark data to SEBQuAD, we cannot claim that this holds for all non-benchmark or banking data. For that we would again be required to perform the evaluation in much larger scale. Nevertheless, this relationship could be interesting to understand in particular in settings where labeled data is expensive, and one has to resort to evaluation on benchmark data. In that case, it would be desirable to have the benchmark performance correlate well with application-specific performance. Moreover, SAS as a metric in itself can be questioned as it relies on a well-trained and performant semantic textual similarity model – the metric can thus never be better than the accuracy of that model. Thus, although SAS provides an orthogonal evaluation metric to F1 and EM, it carries significantly less interpretability which can be an issue in certain applications.

**Handling financial jargon.** The results indicate that financial jargon is well handled by all our systems, although not being explicitly fine-tuned on such data. This could be the explanation for why the fine-tuning on SEBQuAD give no effect: the use of financial terms in SEBQuAD does not provide much additional information to the model being fine-tuned as it already handles financial jargon well. These financial expressions were likely learnt during the pre-training phase of the models, where the models can indeed encounter financial articles. However, it might also be the case that the fine-tuning gave no effect due to the limited size of SEBQuAD; oftentimes model require much larger datasets and variably formatted labels that are about several different topics.

**Overfitted final evidence fusion hyperparameters.** While our results indicate that systems employing final evidence fusion generalize poorly, we cannot confidently claim that the technique is disadvantageous in general. This is because our results also indicate that the hyperparameters for final evidence fusion has been overfitted to a single, small dataset. Instead, it seems like FEF has to be used with caution: its hyperparameters should instead be tuned on several, large datasets. Moreover, they should likely be revised frequently in a real application. This is in particular the case if the application is observed to suffer from data or concept drift as this might require the hyperparameters to be revised even more frequently.

## 7.2 Limitations

**Temporal sensitivity** Our systems are limited to answering questions using information contained in the knowledge base. In all our experiments, we use static knowledge bases. As new knowledge becomes available, a static system will evidently fail to return the correct information. This can obviously be solved by continuously updating the document store. Our experimentation does not provide insights into how it would behave in such a situation as the knowledge bases of most of our benchmark datasets are snapshots of the web or Wikipedia at a certain point in time.

For instance, information clashes could occur. That happens when essentially the same information but from different points in time is contained simultaneously in the document store. For instance, consider two documents from 2021 and 2022, respectively, containing the sentences `Foo Barson is the CEO of Example AB` and `Baz Quxson is the CEO of Example AB`. Given a question about Example AB, either of the sentences may indeed answer the question. It is crucial to understand how a system handles information clashes, and that the necessary tools to handle the problem are employed in a real-life scenario. Using our experimentation, we cannot make claims about our systems' abilities to handle this issue of temporal sensitivity.

**Data modalities.** Another limitation of our system, which have not been assessed in this thesis, is the fact that it can only handle unstructured textual data. In a real application, such as one in banking, one might want to include for instance structured data (like tables), source code, images, and audio in the knowledge base. This further inhibits our ability to make universal claims about the feasibility of our systems in the banking domain. However, given that all of the assumptions stated in Section 1.1.3 are fulfilled, our results give a reasonable indication of the viability of the systems.

**Multilingualism.** Another limitation of our system is its ability to only process data in the English language. Depending on the company, many internal documents might be in the same language as the official language of the country in which the country operates. Depending on the language, there might or might not exist out-of-the-box solutions like monolingual or multilingual models to solve this problem. If not, models would have to be trained oneself to extend them to new languages. In this thesis, we have not built nor evaluated our systems in other languages such as Swedish or other Nordic languages. Hence, we cannot make claims about the feasibility of our system if the banking application requires non-English or multilingual support.

**Non-factoid.** Our systems are limited to answering factoid questions. If the specific banking application instead is concerned with answering non-factoid questions, our experimentation cannot provide many indications of system feasibility or performance.

**Size of knowledge base.** In our experiments, we have not examined how the performance of our systems behave under varying sizes of the knowledge base. In particular, we have used knowledge bases of at most around 200 000 documents (for Natural Questions). In a real application, the number of documents may be drastically larger than this. We cannot make claims regarding the behaviour of our systems' when the number of documents grow significantly.

## 7.3 Production readiness

**Latency, throughput, and cost.** In our experiments we have not considered any technical system performance metrics, which would be vital to monitor in a real application. For instance, depending on the use case, one has a certain degree of demand for system latency and throughput. Larger models generally require more compute power to obtain the same latency and throughput, which could to some degree be mitigated by better hardware (and higher costs). However, in certain applications, very low latency is required. In that case, LLM-based systems might be insufficient for those demands. Particularly, components such as the Ranker might have to be excluded from the pipeline. To get a better idea of how task performance (like F1) correlates with technical system performance (like latency), future work must be done. Thus. we cannot make claims about the feasibility of our system in applications where there are restrictions on factors such as latency, throughput, and cost.

**Dependency management.** In this report, we have not evaluated our system in terms of dependency management. However, in a real application this is a vital factor to account for. We have implemented our systems using `farm-haystack`, and open-source project run by *deepset*. If that project would seize to exist, it might cause problems for our application. Hence, it is crucial to assess the viability of a certain PyTorch/HuggingFace wrapper library before committing to using them in a real, long-term application.

## 7.4 Ethical and environmental considerations

**LLMs require heavy computational resources.** The components of our systems in general require lots of computational resources, which themselves require electricity to run. This may harm the environment, particularly if the servers that the models are deployed to are not utilizing renewable sources of energy. In particular, larger models require more resources. This creates a conflict of interest, as we in Figure 6.4 saw that larger encoder-decoder models seem to perform better. The situation is even worse if one needs to pre-train a new model – for instance, to extend the system to a new language – as pre-training is a data and compute-intensive operation. Although we have not accounted for the environmental impact of the building and execution of our systems, this is a crucial factor to consider in a real application.

**Biased or harmful knowledge bases.** Our system may answer with unintentionally biased or harmful information. If the knowledge base contains information which is biased, harmful, inappropriate, or private, our systems may propagate that information back to the end user. In our experimentation, we have not accounted for such situations. It is crucial that the knowledge base in a real application is thoroughly and continuously revised for this type of information.

**Generative models can be biased.** Generative models adds another layer of considerations in terms of unintentional bias. Since pure generative models are simply generating the next most likely token, they might output information which is harmful and biased against underrepresented or marginalized groups, genders, certain sexual identities, races, among other things. This can happen even if the external knowledge base is pruned from any such information. Although this has not been studied in this report, it is a crucial factor to account for in a real application.

# Chapter 8

# Conclusion

Our thesis examines the adoption of OpenQA systems in the banking domain, and provides valuable insights into their performances and limitations in this setting. We test over 60 different system configurations, including both extractive and generative models, and apply two pipeline augmentation techniques. We achieve close to state-oft-the-art performance on various benchmark datasets. Our systems generalize moderately well to banking domain data, and we find a decrease in performance of 25-29% compared to the benchmarks.

We also investigate the limitations of using lexical evaluation metrics for generative models on banking data. We discover that these metrics do not fully capture the accuracy of numerical and rephrased answers. Moreover, we observe that semantic answer similarity has a stronger correlation between performance on benchmark and banking data compared to the lexical F1 score. This highlights (a) the need for more comprehensive evaluation techniques tailored to specific applications and (b) that lexical benchmark metrics can be less representative of application-specific performance. The latter is crucial to consider in applications where labeled data is expensive.

Our thesis helps bridge the gap between benchmark performance and real-world applications of OpenQA systems in the banking domain. These findings contribute to understanding the practicality and effectiveness of adopting and implementing QA systems in the banking industry.

**Answers to research questions.** For completeness, we once again list our research questions and the answers to these.

- Which OpenQA architectures exhibit superior performance on benchmark datasets and banking-specific data?
  - There is no architecture that performs the best across all benchmark datasets. Instead, performance appears to be dataset-specific. A combination of BM25 and DeBERTA performs the best on SQuAD Open, and DPR coupled with DistilBERT performs the

    best on Natural Questions. An SBERT Retriever coupled with a GPT-3.5 Reader, in the one-shot setting, is the clear winner on both TriviaQA splits. No architecture is clearly superior on the SEBQuAD dataset. Architectures using SBERT or BM25 Retrievers all perform adequately on SEBQuAD when they are coupled with DeBERTA, FLAN-T5, or GPT-3.5.

- Is there a correlation between the performance of OpenQA systems on benchmark datasets and their performance on banking-specific data?
    - Benchmark performance does not seem representative of performance on the SEBQuAD dataset in terms of F1, due to its low correlation. However, there appears to be a correlation in performance measured in SAS score between benchmark datasets and SEBQuAD.
- Do lexical evaluation metrics effectively evaluate the performance of generative models on banking data?
    - Lexical metrics give a general idea of how likely a model is to return an answer that is good or useful. However, these metrics are not perfect as is evident by the examples in Table 6.10. The examples show how lexical metrics fail when a generative model rephrases the answer. It is also evident that the performance, in terms of F1, can be inflated when the answers in a dataset contain words that are very common in the dataset. This is shown by the examples including the MSEK currency specifier in the SEBQuAD dataset.
- What application-specific characteristics significantly impact the performance of QA systems in the banking domain?
    - The number of questions regarding numerics appears to be the largest factor impacting performance in the banking setting. As the examples in Table 6.10 show, these questions make it difficult to measure performance in a meaningful way. Surprisingly, our QA systems appear to handle financial jargon well. This factor does not appear to deteriorate the performance of our systems.

**Future work.** Looking ahead, we are excited about the promising avenues for future research that can further the development and understanding of OpenQA systems in the dynamic banking domain. First, obtaining variance estimates for performance metrics could be achieved through experiments with slight parameter variations or evaluation on additional QA datasets. This would provide insights into the robustness and generalizability of the models beyond the specific dataset used in our study. Second, collecting a richer banking dataset and employing formal annotation guidelines would offer a deeper understanding of the factors influencing model performance, enabling researchers to refine and improve OpenQA systems in this domain. Lastly, hyperparameter tuning on a larger and more diverse dataset could enhance the performance of OpenQA systems by better capturing the intricacies of banking-related queries. Pursuing these avenues of future work can address the limitations of our current study and advance the field of OpenQA systems in the banking domain. By doing so, researchers can contribute to the ongoing improvement and practical applicability of question-answering technologies in the banking industry.

# References

Abuzaid, F., Sethi, G., Bailis, P., and Zaharia, M. (2017). Simdex: Exploiting model similarity in exact matrix factorization recommendations. *CoRR*, abs/1706.01449.

Agirre, E., Cer, D., Diab, M., Gonzalez-Agirre, A., and Guo, W. (2013). * sem 2013 shared task: Semantic textual similarity. In *Second joint conference on lexical and computational semantics (* SEM), volume 1: proceedings of the Main conference and the shared task: semantic textual similarity*, pages 32–43.

Alanazi, S., Mohamed, N., Jarajreh, M., and Algarni, S. (2021). Question answering systems: A systematic literature review. *International Journal of Advanced Computer Science and Applications*, 12.

Arya, S., Mount, D. M., Netanyahu, N. S., Silverman, R., and Wu, A. Y. (1998). An optimal algorithm for approximate nearest neighbor searching fixed dimensions. *J. ACM*, 45(6):891–923.

Ba, J. L., Kiros, J. R., and Hinton, G. E. (2016). Layer normalization.

Bao, S., He, H., Wang, F., Wu, H., Wang, H., Wu, W., Guo, Z., Liu, Z., and Xu, X. (2020). PLATO-2: towards building an open-domain chatbot via curriculum learning. *CoRR*, abs/2006.16779.

Beltagy, I., Peters, M. E., and Cohan, A. (2020). Longformer: The long-document transformer. *CoRR*, abs/2004.05150.

Bengio, Y., Ducharme, R., Vincent, P., and Janvin, C. (2003). A neural probabilistic language model. *J. Mach. Learn. Res.*, 3:1137–1155.

Brown, T. B., Mann, B., Ryder, N., Subbiah, M., Kaplan, J., Dhariwal, P., Neelakantan, A., Shyam, P., Sastry, G., Askell, A., Agarwal, S., Herbert-Voss, A., Krueger, G., Henighan, T., Child, R., Ramesh, A., Ziegler, D. M., Wu, J., Winter, C., Hesse, C., Chen, M., Sigler, E., Litwin, M., Gray, S., Chess, B., Clark, J., Berner, C., McCandlish, S., Radford, A., Sutskever, I., and Amodei, D. (2020). Language models are few-shot learners. *CoRR*, abs/2005.14165.

Calijorne Soares, M. A. and Parreiras, F. S. (2020). A literature review on question answering techniques, paradigms and systems. *Journal of King Saud University - Computer and Information Sciences*, 32(6):635–646.

Carlini, N., Tramèr, F., Wallace, E., Jagielski, M., Herbert-Voss, A., Lee, K., Roberts, A., Brown, T. B., Song, D., Erlingsson, Ú., Oprea, A., and Raffel, C. (2020). Extracting training data from large language models. *CoRR*, abs/2012.07805.

Chakravarti, R., Ferritto, A., Iyer, B., Pan, L., Florian, R., Roukos, S., and Sil, A. (2020). Towards building a robust industry-scale question answering system. In *Proceedings of the 28th International Conference on Computational Linguistics: Industry Track*, pages 90–101, Online. International Committee on Computational Linguistics.

Chandrasekaran, D. and Mago, V. (2021). Evolution of semantic similarity—a survey. *ACM Comput. Surv.*, 54(2).

Chen, D., Fisch, A., Weston, J., and Bordes, A. (2017). Reading Wikipedia to answer open-domain questions. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 1870–1879, Vancouver, Canada. Association for Computational Linguistics.

Chen, D. and Yih, W.-t. (2020). Open-domain question answering. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: Tutorial Abstracts*, pages 34–37, Online. Association for Computational Linguistics.

Chen, T., Kornblith, S., Norouzi, M., and Hinton, G. E. (2020). A simple framework for contrastive learning of visual representations. *CoRR*, abs/2002.05709.

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. (2022). Scaling instruction-finetuned language models.

Ciosici, M. R., Cecil, J., Hedges, A., Lee, D., Freedman, M., and Weischedel, R. M. (2021). Perhaps ptlms should go to school - A task to assess open book and closed book QA. *CoRR*, abs/2110.01552.

Conneau, A., Khandelwal, K., Goyal, N., Chaudhary, V., Wenzek, G., Guzmán, F., Grave, E., Ott, M., Zettlemoyer, L., and Stoyanov, V. (2019). Unsupervised cross-lingual representation learning at scale. *CoRR*, abs/1911.02116.

Dai, A. M. and Le, Q. V. (2015). Semi-supervised sequence learning. *CoRR*, abs/1511.01432.

Dai, Z. and Callan, J. (2020). Context-aware term weighting for first stage passage retrieval. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '20, page 1533–1536, New York, NY, USA. Association for Computing Machinery.

Devlin, J., Chang, M., Lee, K., and Toutanova, K. (2018). BERT: pre-training of deep bidirectional transformers for language understanding. *CoRR*, abs/1810.04805.

Dong, L., Yang, N., Wang, W., Wei, F., Liu, X., Wang, Y., Gao, J., Zhou, M., and Hon, H. (2019). Unified language model pre-training for natural language understanding and generation. *CoRR*, abs/1905.03197.

Dwivedi, S. K. and Singh, V. (2013). Research and reviews in question answering system. *Procedia Technology*, 10:417–424. First International Conference on Computational Intelligence: Modeling Techniques and Applications (CIMTA) 2013.

Farea, A., Yang, Z., Duong, K., Perera, N., and Emmert-Streib, F. (2022). Evaluation of question answering systems: Complexity of judging a natural language.

Floridi, L. and Chiriatti, M. (2020). Gpt-3: Its nature, scope, limits, and consequences. *Minds and Machines*, 30:681–694.

Gao, M., Wu, H., Hua, H., Zhang, W., Chen, Z., and Wang, D. (2021). Research on industry question answering system based on near neighbor vector search. *Journal of Physics: Conference Series*, 1827(1):012009.

Gm, H., Gourisaria, M. K., Pandey, M., and Rautaray, S. S. (2020). A comprehensive survey and analysis of generative models in machine learning. *Computer Science Review*, 38:100285.

Gramer, A. and Danielsson, S. (2023). Predicting Forex Rates using Sentiment Analysis on Financial Articles. Student Paper.

Harris, Z. S. (1954). Distributional structure. *<i>WORD</i>*, 10(2-3):146–162.

He, P., Liu, X., Gao, J., and Chen, W. (2020). Deberta: Decoding-enhanced BERT with disentangled attention. *CoRR*, abs/2006.03654.

Hinton, G., Vinyals, O., and Dean, J. (2015). Distilling the knowledge in a neural network.

Houlsby, N., Giurgiu, A., Jastrzebski, S., Morrone, B., De Laroussilhe, Q., Gesmundo, A., Attariyan, M., and Gelly, S. (2019). Parameter-efficient transfer learning for NLP. In Chaudhuri, K. and Salakhutdinov, R., editors, *Proceedings of the 36th International Conference on Machine Learning*, volume 97 of *Proceedings of Machine Learning Research*, pages 2790–2799. PMLR.

Howard, J. and Ruder, S. (2018). Fine-tuned language models for text classification. *CoRR*, abs/1801.06146.

Hu, M., Peng, Y., and Qiu, X. (2017). Mnemonic reader for machine comprehension. *CoRR*, abs/1705.02798.

Izacard, G., Caron, M., Hosseini, L., Riedel, S., Bojanowski, P., Joulin, A., and Grave, E. (2021). Towards unsupervised dense information retrieval with contrastive learning. *CoRR*, abs/2112.09118.

Izacard, G. and Grave, E. (2020). Leveraging passage retrieval with generative models for open domain question answering. *CoRR*, abs/2007.01282.

Izacard, G., Lewis, P., Lomeli, M., Hosseini, L., Petroni, F., Schick, T., Dwivedi-Yu, J., Joulin, A., Riedel, S., and Grave, E. (2022). Atlas: Few-shot learning with retrieval augmented language models.

Järvelin, K. and Kekäläinen, J. (2002). Cumulated gain-based evaluation of ir techniques. *ACM Trans. Inf. Syst.*, 20(4):422–446.

Jin, N., Siebert, J., Li, D., and Chen, Q. (2022). A survey on table question answering: Recent advances.

Johnson, J., Douze, M., and Jégou, H. (2017). Billion-scale similarity search with gpus. *CoRR*, abs/1702.08734.

Joshi, M., Choi, E., Weld, D. S., and Zettlemoyer, L. (2017). Triviaqa: A large scale distantly supervised challenge dataset for reading comprehension. *CoRR*, abs/1705.03551.

Jégou, H., Douze, M., and Schmid, C. (2011). Product quantization for nearest neighbor search. *IEEE transactions on pattern analysis and machine intelligence*, 33:117–28.

Karpukhin, V., Oguz, B., Min, S., Wu, L., Edunov, S., Chen, D., and Yih, W. (2020). Dense passage retrieval for open-domain question answering. *CoRR*, abs/2004.04906.

Keivani, O., Sinha, K., and Ram, P. (2017). Improved maximum inner product search with better theoretical guarantees. In *2017 International Joint Conference on Neural Networks (IJCNN)*, pages 2927–2934.

Keskar, N. S., McCann, B., Varshney, L. R., Xiong, C., and Socher, R. (2019). Ctrl: A conditional transformer language model for controllable generation.

Kononenko, O., Baysal, O., Holmes, R., and Godfrey, M. W. (2014). Mining modern repositories with elasticsearch. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, MSR 2014, page 328–331, New York, NY, USA. Association for Computing Machinery.

Kuc, R. and Rogozinski, M. (2013). *Elasticsearch server*. Packt Publishing Ltd.

Kwiatkowski, T., Palomaki, J., Redfield, O., Collins, M., Parikh, A., Alberti, C., Epstein, D., Polosukhin, I., Devlin, J., Lee, K., Toutanova, K., Jones, L., Kelcey, M., Chang, M.-W., Dai, A. M., Uszkoreit, J., Le, Q., and Petrov, S. (2019). Natural Questions: A Benchmark for Question Answering Research. *Transactions of the Association for Computational Linguistics*, 7:453–466.

Lampert, A. (2004). A quick introduction to question answering. *ACM Comput. Surv.*

Lazaridou, A., Gribovskaya, E., Stokowiec, W., and Grigorev, N. (2022). Internet-augmented language models through few-shot prompting for open-domain question answering.

Lee, K., Chang, M.-W., and Toutanova, K. (2019). Latent retrieval for weakly supervised open domain question answering. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 6086–6096, Florence, Italy. Association for Computational Linguistics.

Lewis, M., Liu, Y., Goyal, N., Ghazvininejad, M., Mohamed, A., Levy, O., Stoyanov, V., and Zettlemoyer, L. (2019). BART: denoising sequence-to-sequence pre-training for natural language generation, translation, and comprehension. *CoRR*, abs/1910.13461.

Lewis, P. S. H., Perez, E., Piktus, A., Petroni, F., Karpukhin, V., Goyal, N., Küttler, H., Lewis, M., Yih, W., Rocktäschel, T., Riedel, S., and Kiela, D. (2020). Retrieval-augmented generation for knowledge-intensive NLP tasks. *CoRR*, abs/2005.11401.

Lindsay, R. K. (1963). *Inferential Memory as the Basis of Machines Which Understand Natural Language*, page 217–233. MIT Press, Cambridge, MA, USA.

Liu, P. J., Chung, Y., and Ren, J. (2019a). Summae: Zero-shot abstractive text summarization using length-agnostic auto-encoders. *CoRR*, abs/1910.00998.

Liu, S., Zhang, X., Zhang, S., Wang, H., and Zhang, W. (2019b). Neural machine reading comprehension: Methods and trends. *Applied Sciences*, 9(18).

Liu, Y., Ott, M., Goyal, N., Du, J., Joshi, M., Chen, D., Levy, O., Lewis, M., Zettlemoyer, L., and Stoyanov, V. (2019c). Roberta: A robustly optimized BERT pretraining approach. *CoRR*, abs/1907.11692.

Luan, Y., Eisenstein, J., Toutanova, K., and Collins, M. (2021). Sparse, Dense, and Attentional Representations for Text Retrieval. *Transactions of the Association for Computational Linguistics*, 9:329–345.

Lund, B. D. and Wang, T. (2023). Chatting about chatgpt: how may ai and gpt impact academia and libraries? *Library Hi Tech News*.

Ma, X., Sun, K., Pradeep, R., Li, M., and Lin, J. (2022). Another look at dpr: Reproduction of training and replication of retrieval. In Hagen, M., Verberne, S., Macdonald, C., Seifert, C., Balog, K., Nørvåg, K., and Setty, V., editors, *Advances in Information Retrieval*, pages 613–626, Cham. Springer International Publishing.

Malkov, Y. A. and Yashunin, D. A. (2018). Efficient and robust approximate nearest neighbor search using hierarchical navigable small world graphs. *IEEE transactions on pattern analysis and machine intelligence*, 42(4):824–836.

Mao, Y., He, P., Liu, X., Shen, Y., Gao, J., Han, J., and Chen, W. (2021). Generation-augmented retrieval for open-domain question answering. In *Proceedings of the 59th Annual Meeting of the Association for Computational Linguistics and the 11th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, pages 4089–4100, Online. Association for Computational Linguistics.

Mikolov, T., Chen, K., Corrado, G., and Dean, J. (2013). Efficient estimation of word representations in vector space. *CoRR*, abs/1301.3781.

Mishra, A. and Jain, S. K. (2016). A survey on question answering systems with classification. *Journal of King Saud University - Computer and Information Sciences*, 28(3):345–361.

Moldovan, D., Harabagiu, S., Pasca, M., Mihalcea, R., Girju, R., Goodrum, R., and Rus, V. (2000). The structure and performance of an open-domain question answering system. In *Proceedings of the 38th Annual Meeting of the Association for Computational Linguistics*, pages 563–570, Hong Kong. Association for Computational Linguistics.

Mu, C., Yang, B., and Yan, Z. (2019). An empirical comparison of FAISS and FENSHSES for nearest neighbor search in hamming space. *CoRR*, abs/1906.10095.

Nguyen, T., Rosenberg, M., Song, X., Gao, J., Tiwary, S., Majumder, R., and Deng, L. (2016). MS MARCO: A human generated machine reading comprehension dataset. *CoRR*, abs/1611.09268.

OpenAI (2023). Gpt-4 technical report.

Pearce, K., Zhan, T., Komanduri, A., and Zhan, J. (2021). A comparative study of transformer-based language models on extractive question answering. *CoRR*, abs/2110.03142.

Pennington, J., Socher, R., and Manning, C. (2014). GloVe: Global vectors for word representation. In *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 1532–1543, Doha, Qatar. Association for Computational Linguistics.

Peters, M. E., Neumann, M., Iyyer, M., Gardner, M., Clark, C., Lee, K., and Zettlemoyer, L. (2018). Deep contextualized word representations. *CoRR*, abs/1802.05365.

Prager, J. (2007). *Open-Domain Question Answering*. Foundations and trends in information retrieval. Now Publishers.

Qu, C., Yang, L., Chen, C., Qiu, M., Croft, W. B., and Iyyer, M. (2020). Open-retrieval conversational question answering. *CoRR*, abs/2005.11364.

Radford, A. and Narasimhan, K. (2018). Improving language understanding by generative pre-training.

Radford, A., Wu, J., Child, R., Luan, D., Amodei, D., and Sutskever, I. (2019). Language models are unsupervised multitask learners.

Raffel, C., Shazeer, N., Roberts, A., Lee, K., Narang, S., Matena, M., Zhou, Y., Li, W., and Liu, P. J. (2019). Exploring the limits of transfer learning with a unified text-to-text transformer. *CoRR*, abs/1910.10683.

Rajpurkar, P., Jia, R., and Liang, P. (2018). Know what you don't know: Unanswerable questions for squad. *CoRR*, abs/1806.03822.

Ramachandran, P., Liu, P. J., and Le, Q. V. (2016). Unsupervised pretraining for sequence to sequence learning. *CoRR*, abs/1611.02683.

Reimers, N. and Gurevych, I. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks. *CoRR*, abs/1908.10084.

Risch, J., Möller, T., Gutsch, J., and Pietsch, M. (2021). Semantic answer similarity for evaluating question answering models. *CoRR*, abs/2108.06130.

Roberts, A., Raffel, C., and Shazeer, N. (2020). How much knowledge can you pack into the parameters of a language model? *CoRR*, abs/2002.08910.

Robertson, S. E. and Walker, S. (1994). Some simple effective approximations to the 2-poisson model for probabilistic weighted retrieval. In *Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*.

Roller, S., Dinan, E., Goyal, N., Ju, D., Williamson, M., Liu, Y., Xu, J., Ott, M., Shuster, K., Smith, E. M., Boureau, Y., and Weston, J. (2020). Recipes for building an open-domain chatbot. *CoRR*, abs/2004.13637.

Sanh, V., Debut, L., Chaumond, J., and Wolf, T. (2019). Distilbert, a distilled version of BERT: smaller, faster, cheaper and lighter. *CoRR*, abs/1910.01108.

Sennrich, R., Haddow, B., and Birch, A. (2015). Neural machine translation of rare words with subword units. *CoRR*, abs/1508.07909.

Shuster, K., Urbanek, J., Dinan, E., Szlam, A., and Weston, J. (2020). Deploying lifelong open-domain dialogue learning. *CoRR*, abs/2008.08076.

Simmons, R. F. (1965). Answering english questions by computer: A survey. *Commun. ACM*, 8(1):53–70.

Sparck Jones, K. (1972). A statistical interpretation of term specificity and its application in retrieval. *Journal of documentation*, 28(1):11–21.

Stahlberg, F. (2020). Neural machine translation: A review. *Journal of Artificial Intelligence Research*, 69:343–418.

Surdeanu, M., Ciaramita, M., and Zaragoza, H. (2008). Learning to rank answers on large online qa collections. In *Annual Meeting of the Association for Computational Linguistics*.

Sutskever, I., Vinyals, O., and Le, Q. V. (2014). Sequence to sequence learning with neural networks. *CoRR*, abs/1409.3215.

Svore, K. and Burges, C. (2009). A machine learning approach for improved bm25 retrieval. pages 1811–1814.

Tang, R., Nogueira, R. F., Zhang, E., Gupta, N., Cam, P., Cho, K., and Lin, J. (2020). Rapidly bootstrapping a question answering dataset for COVID-19. *CoRR*, abs/2004.11339.

Taori, R., Gulrajani, I., Zhang, T., Dubois, Y., Li, X., Guestrin, C., Liang, P., and Hashimoto, T. B. (2023). Stanford alpaca: An instruction-following llama model. `https://github.com/tatsu-lab/stanford_alpaca`.

Thakur, N., Reimers, N., Rücklé, A., Srivastava, A., and Gurevych, I. (2021). BEIR: A heterogenous benchmark for zero-shot evaluation of information retrieval models. *CoRR*, abs/2104.08663.

Touvron, H., Lavril, T., Izacard, G., Martinet, X., Lachaux, M.-A., Lacroix, T., Rozière, B., Goyal, N., Hambro, E., Azhar, F., Rodriguez, A., Joulin, A., Grave, E., and Lample, G. (2023). Llama: Open and efficient foundation language models.

Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A. N., Kaiser, L., and Polosukhin, I. (2017). Attention is all you need. *CoRR*, abs/1706.03762.

Wang, S., Zhou, L., Gan, Z., Chen, Y., Fang, Y., Sun, S., Cheng, Y., and Liu, J. (2020a). Clusterformer: Clustering-based sparse transformer for long-range dependency encoding. *CoRR*, abs/2009.06097.

Wang, W., Wei, F., Dong, L., Bao, H., Yang, N., and Zhou, M. (2020b). Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. *CoRR*, abs/2002.10957.

White, J., Fu, Q., Hays, S., Sandborn, M., Olea, C., Gilbert, H., Elnashar, A., Spencer-Smith, J., and Schmidt, D. C. (2023). A prompt pattern catalog to enhance prompt engineering with chatgpt.

Wilkinson, R. (1994). Effective retrieval of structured documents. In *Proceedings of the 17th Annual International ACM SIGIR Conference on Research and Development in Information Retrieval*, SIGIR '94, page 311–317, Berlin, Heidelberg. Springer-Verlag.

Wolf, T., Debut, L., Sanh, V., Chaumond, J., Delangue, C., Moi, A., Cistac, P., Rault, T., Louf, R., Funtowicz, M., Davison, J., Shleifer, S., von Platen, P., Ma, C., Jernite, Y., Plu, J., Xu, C., Le Scao, T., Gugger, S., Drame, M., Lhoest, Q., and Rush, A. (2020). Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, pages 38–45, Online. Association for Computational Linguistics.

Xie, S. M., Raghunathan, A., Liang, P., and Ma, T. (2021). An explanation of in-context learning as implicit bayesian inference. *CoRR*, abs/2111.02080.

Yang, W., Xie, Y., Lin, A., Li, X., Tan, L., Xiong, K., Li, M., and Lin, J. (2019). End-to-end open-domain question answering with. In *Proceedings of the 2019 Conference of the North*. Association for Computational Linguistics.

Yao, X., Zheng, Y., Yang, X., and Yang, Z. (2021). NLP from scratch without large-scale pretraining: A simple and efficient framework. *CoRR*, abs/2111.04130.

Ye, S., Hwang, H., Yang, S., Yun, H., Kim, Y., and Seo, M. (2023). In-context instruction learning.

Yin, X. and Wan, X. (2022). How do Seq2Seq models perform on end-to-end data-to-text generation? In *Proceedings of the 60th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, pages 7701–7710, Dublin, Ireland. Association for Computational Linguistics.

Zaheer, M., Guruganesh, G., Dubey, A., Ainslie, J., Alberti, C., Ontañón, S., Pham, P., Ravula, A., Wang, Q., Yang, L., and Ahmed, A. (2020). Big bird: Transformers for longer sequences. *CoRR*, abs/2007.14062.

Zeng, J., Lin, X. V., Hoi, S. C., Socher, R., Xiong, C., Lyu, M., and King, I. (2020). Photon: A robust cross-domain text-to-SQL system. In *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics: System Demonstrations*, pages 204–214, Online. Association for Computational Linguistics.

Zhang, H., Gong, Y., Shen, Y., Li, W., Lv, J., Duan, N., and Chen, W. (2021). Poolingformer: Long document modeling with pooling attention. *CoRR*, abs/2105.04371.

Zhang, Y. (2019). Bert for question answering on squad 2 . 0.

Zhao, T., Lu, X., and Lee, K. (2021). SPARTA: Efficient open-domain question answering via sparse transformer matching retrieval. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, pages 565–575, Online. Association for Computational Linguistics.

Zhu, F., Lei, W., Wang, C., Zheng, J., Poria, S., and Chua, T. (2021). Retrieving and reading: A comprehensive survey on open-domain question answering. *CoRR*, abs/2101.00774.

Zhuang, F., Qi, Z., Duan, K., Xi, D., Zhu, Y., Zhu, H., Xiong, H., and He, Q. (2019). A comprehensive survey on transfer learning. *CoRR*, abs/1911.02685.

Zuva, K. and Zuva, T. (2012). Evaluation of information retrieval systems. *International journal of computer science & information technology*, 4(3):35.

# Appendices

# Appendix A
# Intuition behind query-document similarity

The notion of encoding passages and questions as embedding vectors may appear opaque at first. In order to get an intuition for the functionality of the retriever we can make use of a visual example. using a Wikipedia-based knowledge base containing multiple passages related to steam engines. Using the encoding function of the Retriever, these passages could be represented as embeddings in a vector space of arbitrary dimension (often referred to as the *latent space*). In order to get a visual intuition for this vector space we may project it onto three dimensions using a dimensionality reduction technique such as PCA or t-SNE. This results in the graphical representation shown in Figure A.1. Here, the dots represent documents of the knowledge base in their vector representations. Looking at the highlighted embedding vectors in the figure (annotated with the content of the passages) we may note that documents covering steam engines appear close to each other in the 3D projected space. This is exactly what we desire.

We may also examine the embedding vector of the question, as shown in Figure A.2a. Here, the question embedding is placed within the latent space of the documents. Note that it too appears to be close to passages related to steam engines, as desired. These nearby passages would probably be good candidates for finding an answer to our question. As shown in Figure A.2b, a passage covering the Australian Labour Party – that is not likely to be relevant to the question about steam engines – appears far away from the question in the latent space. This example highlights the function of the Retriever: to represent questions and passages in a vector space where geometric distance fruitfully represents semantic overlap. The latent space in the steam engine example can be further explored using this interactive tool[1].

[1]`http://projector.tensorflow.org/?config=https://gist.githubusercontent.com/NilRom/94c0e5eae9449722bbc500bf4ad491be/raw/002bef4c0fde2c520b670d8ed3ff549a9a6c4d45/template_projector_config.json`

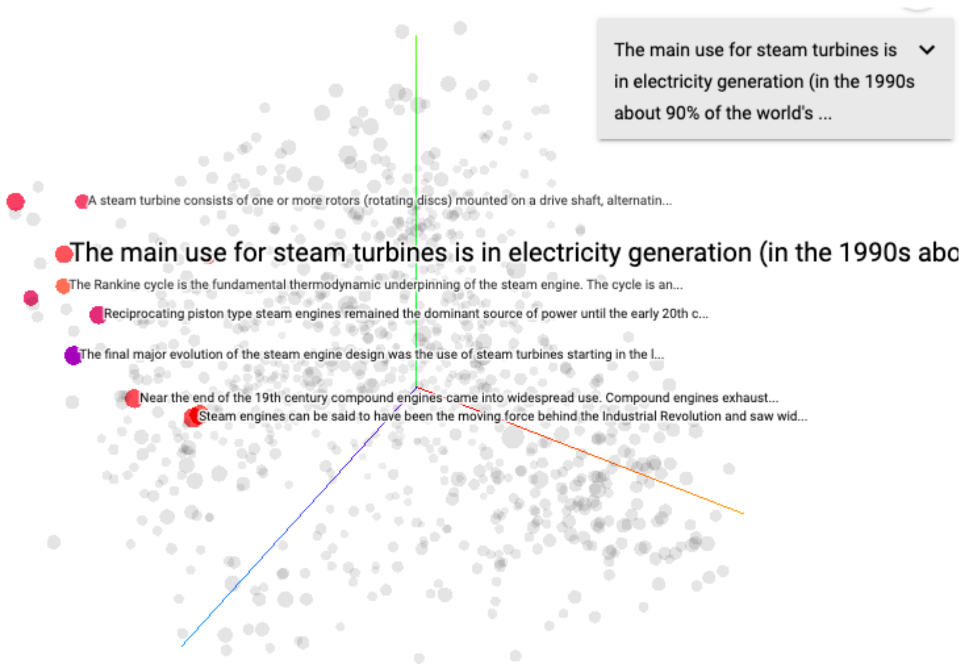
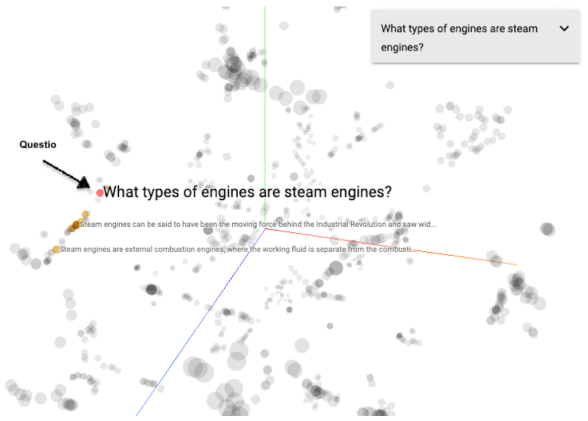
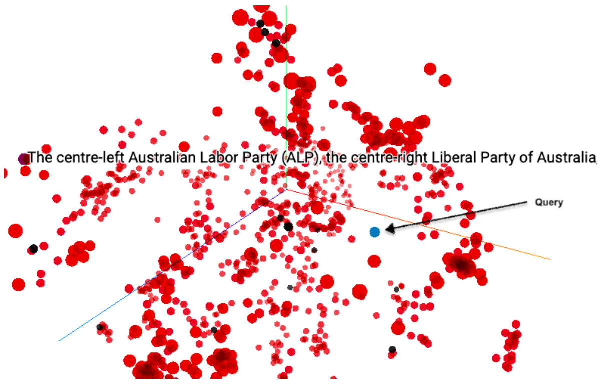


**Figure A.1:** Graphical representation of the latent space of the documents, I.e the embeddings of the documents as encoded by the retriever projected onto a 3D space using t-SNE. A document and its ten closest (in terms of cosine similarity) neighbors are highlighted.



**(a)** Embeddings of a question and the top ten closest documents in terms of cosine similarity. Documents that are probable to contain the answer to the question appear close to it in the latent space



**(b)** A document covering the Australian Labour Party is not very relevant to the steam engine question and appears far away from the question highlighted in blue.

**Figure A.2:** Graphical representations of questions and passages encoded in the latent space.

# Appendix B

# Prompts

The following prompts are used in the experimentation with the generative models. In the zero-shot setting, we use

```
Given the context please answer the question.

Context: {documents};

Question: {query};

Answer:,
```

where content is to be inserted at inference time between the curly brackets `{}`. In the one-shot setting we use for GPT-3.5 the prompt

~~~
Task:  Given a context and a question, answer the question by extracting
a span from the context.

The format of the input is between the following triple backticks,
where the content will input at the ...  symbol:

```

Context:  ...;

Question:  ...;

```

The output should be an answer to the question.

```
~~~

```
Context: The English name 'Normans' comes from the French words
Normans/Normanz, plural of Normant, modern French normand, which is
itself borrowed from Old Low Franconian Nortmann 'Northman' or directly
from Old Norse Norðmaðr, Latinized variously as Nortmannus, Normannus, or
Nordmannus (recorded in Medieval Latin, 9th century) to mean 'Norseman,
Viking'.;

Question: What is the original meaning of the word Norman?;

```

Viking

```

Context: {documents};

Question: {query};

```
```

For the one-shot setting with FLAN-T5 we require a shorter example to fit in the 512 maximal token length:

```
Given a context and a question, answer the question.

Context: It is a replica of the grotto at Lour- des, France where
the Virgin Mary reputedly appeared to Saint Bernadette Soubirous in
1858.;

Question: To whom did the Virgin Mary allegedly appear in 1858 in
Lourdes France??;

Answer: Saint Bernadette Soubirous;

Context: {documents};

Question: {query};

Answer:
```
```

# Appendix C

# SOTA Results on Public Datasets

Table C.1 displays the state-of-the-art (SOTA) results on the public benchmarking datasets outlined in Chapter 3.

| Dataset | F1 | EM |
|---|---|---|
| **SQuAD Open** | 78.8 | 69.5 |
| **NQ** | 64.1 | 64.0 |
| **TriviaQA Wikipedia (verified)** | 95.9 | 94.0 |
| **TriviaQA Web (verified)** | 92 | 90 |

**Table C.1:** SOTA Results on public datasets.

# Appendix D

# Division of Work

Both authors have contributed equally to both the experimentation and writing aspects of this thesis. The division of responsibilities for specific topics is outlined in Table D.1.

| **Technical Topics** | **Simon Danielsson** | **Nils Romanus** |
|---|---|---|
| Experiment design and setup | 50% | 50% |
| Benchmark data collection and preprocessing | 50% | 50% |
| Implementation | 60% | 40% |
| OpenAI Integration | 70% | 30% |
| Google Cloud Platform Integration | 30% | 70% |
| Dockerization | 100% | 0% |
| Dependency Management | 100% | 0% |
| Evaluation and analysis | 50% | 50% |
| Results interpretation | 50% | 50% |
| Prompt engineering for generative models | 100% | 0% |
| SEBQuAD annotation | 0% | 100% |
| **Report Topics** | **Simon Danielsson** | **Nils Romanus** |
| Writing of the thesis | 50% | 50% |
| Coordinating objectives of LTH and SEB | 50% | 50% |
| Graphic design | 0% | 100% |

**Table D.1:** Division of responsibilities between the authors.

INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2016-04-28

**EXAMENSARBETE** Application Specific Instruction-set Processor Using a Parametrizable multi-SIMD Synthesizeable Model Supporting Design Space Exploration
**STUDENT** Magnus Hultin
**HANDLEDARE** Flavius Gruian (LTH)
**EXAMINATOR** Krzysztof Kuchcinski (LTH)

# Parametrisk processor modell för design utforskning

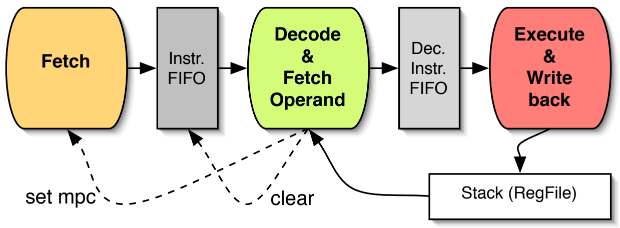
POPULÄRVETENSKAPLIG SAMMANFATTNING **Magnus Hultin**

Applikations-specifika processorer är allt mer vanligt för få ut rätt prestanda med så lite resurser som möjligt. Detta arbete har en parametrisk modell för att kunna testa hur mycket resurser som behövs för en specifik applikation.

För att öka prestandan i dagens processorer finns det vektorenheter och flera kärnor i processorer. Vektorenheten finns till för att kunna utföra en operation på en mängd data samtidigt och flera kärnor gör att man kan utföra fler instruktioner samtidigt. Ofta är processorerna designade för att kunna stödja en mängd olika datorprogram. Detta resulterar i att det blir kompromisser som kan påverka prestandan för vissa program och vara överflödigt för andra. I t.ex. videokameror, mobiltelefoner, medicinsk utrustning, digital kameror och annan inbyggd elektronik, kan man istället använda en processor som saknar vissa funktioner men som istället är mer energieffektiv. Man kan jämföra det med att frakta ett paket med en stor lastbil istället för att använda en mindre bil där samma paketet också skulle få plats.

I mitt examensarbete har jag skrivit en modell som kan användas för att snabbt designa en processor enligt vissa parametrar. Dessa parametrar väljs utifrån vilket eller vilka program man tänkta köra på den. Vissa program kan t.ex. lättare använda flera kärnor och vissa program kan använda korta eller längre vektorenheter för dess data.

Modellen testades med olika multimedia program. Den mest beräkningsintensiva och mest upprepande delen av programmen användes. Dessa kallas för kärnor av programmen. Kärnorna som användes var ifrån MPEG och JPEG, som används för bildkomprimering och videokomprimering.



Resultatet visar att det finns en prestanda vinst jämfört med generella processorer men att detta också ökar resurserna som behövs. Detta trots att den generella processorn har nästan dubbelt så hög klockfrekvens än dem applikationsspecifika processorerna. Resultatet visar också att schemaläggning av instruktionerna i programmen spelar en stor roll för att kunna utnyttja resurserna som finns tillgängliga och därmed öka prestandan. Med den schemaläggningen som utnyttjade resurserna bäst var prestandan minst 79% bättre än den generella processorn.