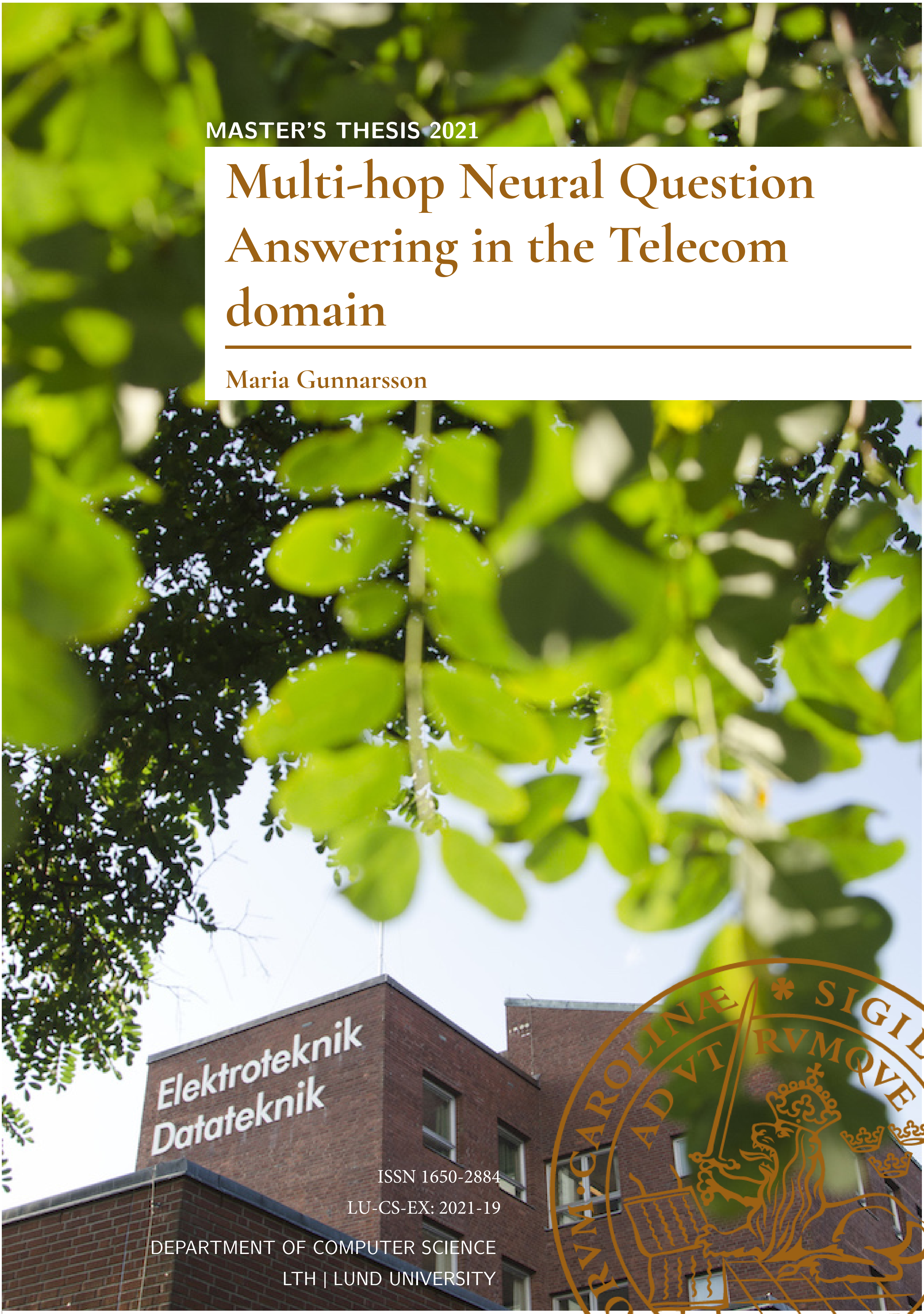
MASTER'S THESIS 2021

Multi-hop Neural Question Answering in the Telecom domain

Maria Gunnarsson



ISSN 1650-2884
LU-CS-EX: 2021-19

DEPARTMENT OF COMPUTER SCIENCE
LTH | LUND UNIVERSITY

EXAMENSARBETE
Datavetenskap

LU-CS-EX: 2021-19

Multi-hop Neural Question Answering in the Telecom domain

Neurala frågesvarssystem med multipla steg för telekomdomänen

Maria Gunnarsson

Multi-hop Neural Question Answering in the Telecom domain

Maria Gunnarsson
tfy15mgu@student.lu.se

June 10, 2021

Master's thesis work carried out at Ericsson.

Supervisors: Fitsum Gaim Gebre, fitsum.gaim.gebre@ericsson.com
Pierre Nugues, pierre.nugues@cs.lth.se

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

Abstract

Neural systems for open-domain question answering are often developed for general domain data, due to the availability of large-scale question answering datasets. In this master thesis, I investigate how three question answering models perform within a narrower domain: The telecommunication domain. I also examine how these models can be adapted for this specific domain. To evaluate the models, together with the Ericsson research group, I built two telecommunication domain question answering datasets: one with simple questions and one with more complex ones. I also used a part of the in-domain dataset with simple questions to fine-tune neural parts in the question answering systems.

My results show that a significant domain adaptation can be achieved by fine-tuning pre-trained language models with a relatively small dataset. This finding should pave the way to more effective construction of domain specialized question answering systems as annotating large datasets consumes a lot of time.

Keywords: Natural language processing, question answering, language model, domain adaptation

Acknowledgements

My Master's thesis project would have been far from this good and run half as smoothly if it would not have been for my two supervisors: Fitsum Gaim Gebre at Ericsson and Pierre Nugues at LTH. I am deeply grateful for all the support they have given me throughout this project. Due to the circumstances, I have needed to sit a lot on my own while working on this, and then it has been extra helpful to know that I always can reach out to them.

I also want to thank the rest of the Telecom AI group within team Muon Spruce in GAIA, Ericsson; Vincent Huang A, Jieqiang Wei, and Henrik Holm. Thank you for the great collaboration and feedback during my work. To Henrik, who has also written his Master's thesis within the Telecom AI project, I owe a great deal of gratitude for his company and his help.

Finally, I am extremely thankful for all the support I have been receiving from my friends and family during this semester.

Contents

Chapter 1

Introduction

Systems providing answers to natural language questions given only unstructured text have recently become a popular application of several machine learning techniques. This task is referred to as *open-domain question answering* (openQA). More complex questions, that for instance require reasoning over several documents, cannot be answered by regular openQA systems. Several different approaches have been proposed to deal with these *multi-hop* questions. Iteratively retrieving more context documents, building cognitive reasoning graphs, and decomposing the complex questions into simpler ones are examples of tackling this problem.

However, many of the published techniques are developed and tailored for general domain data. This makes them unfit to adapt to one specific domain. In this Master's thesis, I show the requirements that need to be met to apply an open-domain question answering system to a new domain. More specifically, I considered the adaptation of systems to the telecom domain.

My findings are that published systems could perform reasonably well on a telecom question answering dataset. I achieved a significant improvement in performance when fine-tuning a pre-trained language model on a small domain-specific dataset. My results show how openQA models can be domain-adapted in a relatively simple manner, without the need to create a large training dataset.

With much open-domain question answering research focusing on general domain knowledge, I show in this thesis that openQA systems can be useful also in a narrower context. This fact, along with the systems' ability to deal with complex multi-hop questions, will make them useful in a large variety of contexts in the future.

Chapter 2
Previous work

Natural language processing is a very hot research area at the moment. The topic is stretched over a vast variance of tasks, and one of them is *question answering*. The aim of question answering is as easy as it sounds: Provide an answer to a question that is posed in natural language. But as easy as the task is, as hard is the solution.

2.1 Question answering

Question answering dates back to the 1960s. Baseball (Green et al., 1961) is said to be the first question answering system and could provide answers regarding the US baseball league. The information inside the system was stored in a well-defined dictionary and included facts about for example team names, game times, and locations.

Following Baseball, question answering systems did for a long time require information to be structured in a certain way. Around 2007, large knowledge bases such as DBpedia and Freebase were developed as well as open-domain datasets such as WebQuestions and SimpleQuestions, making the knowledge-based QA techniques evolve. The knowledge bases are constructed according to some predefined structure, which often requires manual work. Therefore, scaling these systems is difficult.

Textual QA systems are on the other hand easier to scale. These are systems that provide answers to questions given only unstructured data. Textual question answering is often regarded from two different settings: *machine reading comprehension* (MRC) and *open-domain question answering* (openQA). The difference is the availability of context data. In the former setting is the context paragraph given, but in the latter is the context not specified at all.

Open-domain QA can be seen as an extension of MRC, which is why it is sometimes referred to as *machine reading comprehension at scale*. Open-domain question answering gained some public fame when IBM Watson won the *Jeopardy!* game against human players in 2011. Watson was created with techniques both from information retrieval, natural language processing, and knowledge bases.

The term *closed-domain question answering* is used for some QA systems. They are constructed just as openQA systems, but they are built for questions under a specific domain.

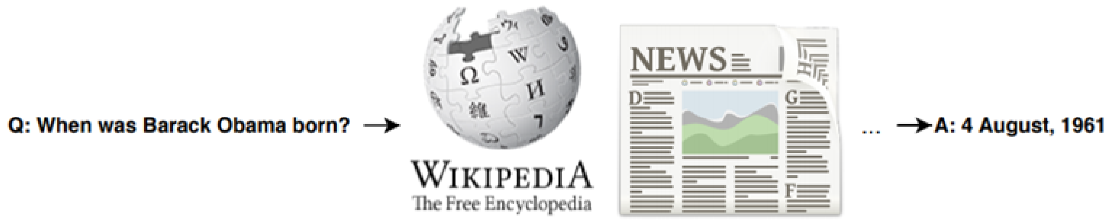


Figure 2.1: A conceptual overview of open-domain question answering (Zhu et al., 2021).

2.1.1 Multi-hop question answering

Questions that require information from several documents or paragraphs to be answered are in the NLP world named *multi-hop questions* or questions that require *multi-step reasoning*. Due to the design of most large-scale datasets, general open-domain question answering models cannot handle questions that need information from more than one paragraph.

This limitation has been addressed by several scientists lately. Publication of multi-hop question answering datasets has made it possible to further develop question answering systems to conduct multi-step reasoning.

To distinguish between the questions requiring one, respectively several, paragraphs to be answered, I will use the terms *single-hop* and *multi-hop* questions throughout this report.

2.2 Datasets

Datasets are key components within most machine learning tasks. In question answering, the research picked up speed after the release of several large-scale question answering datasets during 2017-2018 (Zhu et al., 2021). Since these releases, the inclusion of one or several neural parts in question answering systems has become standard practice. This section accounts for some benchmark datasets that are applicable for open-domain question answering.

2.2.1 Single-hop datasets

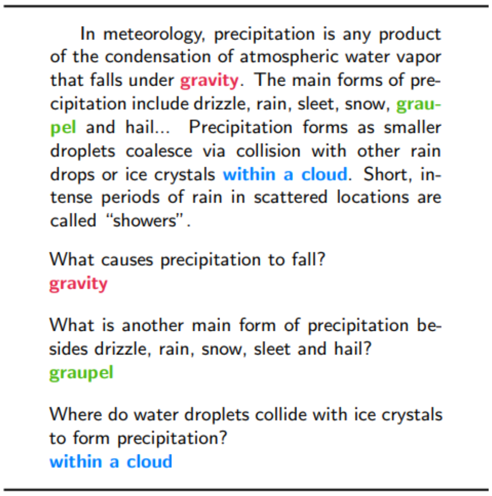
A distinction is made between MRC datasets and openQA datasets. In machine reading comprehension datasets, the system has easy access to the information source, i.e. a paragraph of text which contains the correct answer. Hermann et al. (2015) published what is probably the first large-scale dataset for machine reading comprehension. The authors argued that supervised machine learning techniques had been absent from the field of machine reading comprehension due to the lack of large-scale training datasets. They proposed a novel approach to constructing a supervised reading comprehension dataset from the CNN and Daily Mail websites with over one million question-answer pairs. The approach includes converting summary and paraphrase sentences with their associated documents to context-query-answer triples by using entity recognition and anonymization algorithms.

The magnitude of the CNN/Daily Mail dataset is a great advantage, but drawbacks have been noted. Rajpurkar et al. (2016) pointed out that the dataset is semi-synthetic and does not share the same characteristics as explicit reading comprehension questions. Rajpurkar et al. (2016) instead presented a new dataset named SQuAD, *Stanford question answering dataset*. It consists of over 100,000 question-answer pairs, where the questions are written by crowdworkers and the answers are text spans from Wikipedia articles.

To ensure high-quality articles Rajpurkar et al. (2016) used Project Nayuki's Wikipedia's internal PageRanks to identify the top 10,000 articles of English Wikipedia. From these, they sampled 536 articles uniformly at random. Then, they extracted paragraphs from each article. They removed the images, figures, tables, and paragraphs shorter than 500 characters, and at last 23,215 paragraphs remained. The 536 articles with their corresponding question-answer pairs were randomly divided into a training set (80 %), a development set (10 %), and a test set (10 %).

To address the issue of extractive reading comprehension systems making unreliable guesses for questions without an answer, Rajpurkar et al. (2018) released SQuAD2.0. This dataset is an extension of the former version SQuAD1.1. In addition to the answerable questions, SQuAD2.0 also includes over 50,000 unanswerable ones. As for the answerable ones, the unanswerable questions were written by crowdworkers to resemble answerable ones. For a system to perform well on SQuAD2.0, it must both produce correct answers to the answerable questions and identify when a question does not have an answer in the provided context.

The two SQuAD datasets were created for machine reading comprehension. However, Chen et al. (2017) extended SQuAD1.1 to what was later named *SQuAD$_{open}$* to evaluate their open-domain question answering system. The extension meant that the system was not provided with the associated paragraph when asked a question but the whole of Wikipedia.



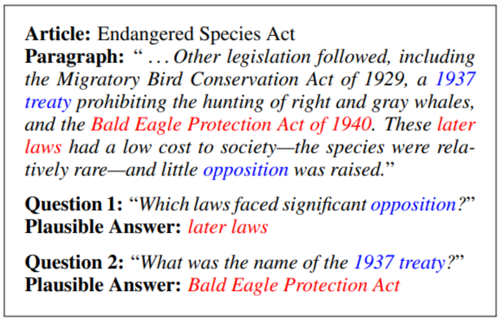
In meteorology, precipitation is any product of the condensation of atmospheric water vapor that falls under gravity. The main forms of precipitation include drizzle, rain, sleet, snow, graupel and hail... Precipitation forms as smaller droplets coalesce via collision with other rain drops or ice crystals within a cloud. Short, intense periods of rain in scattered locations are called "showers".

What causes precipitation to fall?
gravity

What is another main form of precipitation besides drizzle, rain, snow, sleet and hail?
graupel

Where do water droplets collide with ice crystals to form precipitation?
within a cloud

Figure 2.2: Questions and answers from SQuAD 1.1 (Rajpurkar et al., 2016).



Article: Endangered Species Act
Paragraph: "*. . . Other legislation followed, including the Migratory Bird Conservation Act of 1929, a 1937 treaty prohibiting the hunting of right and gray whales, and the Bald Eagle Protection Act of 1940. These later laws had a low cost to society—the species were relatively rare—and little opposition was raised.*"

Question 1: "*Which laws faced significant opposition?*"
Plausible Answer: *later laws*

Question 2: "*What was the name of the 1937 treaty?*"
Plausible Answer: *Bald Eagle Protection Act*

Figure 2.3: Questions and answers from SQuAD 2.0 (Rajpurkar et al., 2018).

As mentioned earlier, the answers to the questions in SQuAD are a text span from a certain paragraph. The same answer type is adopted in several other large-scale QA datasets. Examples of these include NewsQA (Trischler et al., 2016), where the context consists of new articles from CNN news; SearchQA (Dunn et al., 2017), with context from Google Search and Natural Questions (Kwiatkowski et al., 2019) which also have Wikipedia as context. Other answer types used in QA datasets are multiple choices, boolean, and free form.

2.2.2 Multi-hop datasets

The datasets mentioned so far have one thing in common, the questions are answerable, or unanswerable, with just one supporting paragraph. Yang et al. (2018) point out that this fails to train question answering systems to perform complex reasoning. Therefore, they presented HotpotQA, a question answering dataset, where the questions require finding and reasoning over multiple supporting documents to be answered. At the time of publication of HotpotQA, the existing multi-hop question answering datasets were constructed using some existing knowledge base. These were for example QAngaroo (Welbl et al., 2018) and ComplexWebQuestions (Talmor and Berant, 2018). Yang et al. (2018) mention that these datasets are limited by the architecture of the knowledge base, which therefore also limits the diversity of the questions and answers.

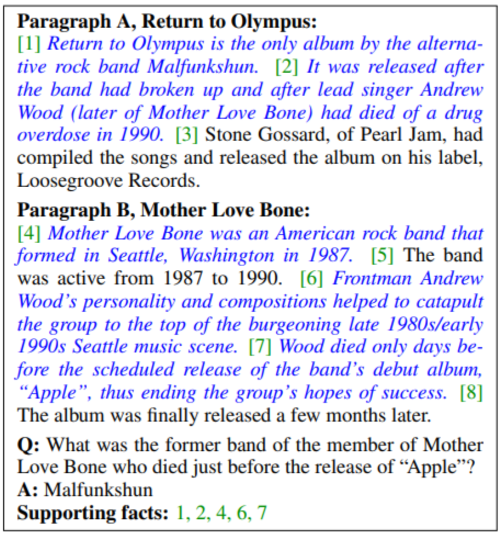
HotpotQA is not created around some knowledge base or schema and has been named the first dataset for the open-domain multi-hop task (Khattab et al., 2021). The content is manually annotated which ensures natural questions. The full dataset consists of slightly more than 100,000 question-answer pairs. The answers are text spans, and for each question, there are two supporting paragraphs, which are required to answer correctly. Two benchmark settings are presented together with the dataset:

Distractor setting The distractor setting combines the two supporting paragraphs with 8 distractor paragraphs. These 10 paragraphs are shuffled before sent to a model, and the task becomes to identify which are the relevant ones.

Full-wiki setting The full-wiki setting is even more challenging. In this setting, the first paragraphs from all Wikipedia articles are provided along with the question.

While still being the most used dataset for multi-hop question answering, there are limitations to it that have been pointed out. Khattab et al. (2021), among others, mention the limitation to only two-hop questions. Many of the questions also do not require strong multi-hop reasoning capabilities. Therefore, Khattab et al. (2021) evaluated their proposed model on another dataset called HoVer (Jiang et al., 2020) in addition to HotpotQA.

HoVer is a many-hop verification dataset, which requires models to extract relevant facts from several Wikipedia articles to classify whether a statement is *supported* or *not supported* by the facts. The dataset includes both two-, three- and four-hop examples. Multi-evidence FEVER is a similar dataset, where facts need to be verified using multiple documents (Thorne et al., 2018). It has been used by for example Xiong et al. (2021) in their evaluations, along with HotpotQA.

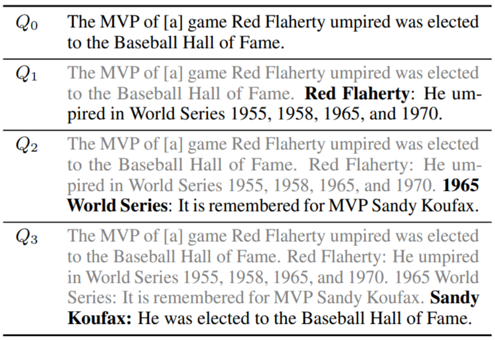


Paragraph A, Return to Olympus:
[1] *Return to Olympus is the only album by the alternative rock band Malfunkshun.* [2] *It was released after the band had broken up and after lead singer Andrew Wood (later of Mother Love Bone) had died of a drug overdose in 1990.* [3] Stone Gossard, of Pearl Jam, had compiled the songs and released the album on his label, Loosegroove Records.

Paragraph B, Mother Love Bone:
[4] *Mother Love Bone was an American rock band that formed in Seattle, Washington in 1987.* [5] The band was active from 1987 to 1990. [6] *Frontman Andrew Wood's personality and compositions helped to catapult the group to the top of the burgeoning late 1980s/early 1990s Seattle music scene.* [7] *Wood died only days before the scheduled release of the band's debut album, "Apple", thus ending the group's hopes of success.* [8] The album was finally released a few months later.

Q: What was the former band of the member of Mother Love Bone who died just before the release of "Apple"?
A: Malfunkshun
Supporting facts: 1, 2, 4, 6, 7

Figure 2.4: Questions and answers from HotpotQA (Yang et al., 2018).



Q_0	The MVP of [a] game Red Flaherty umpired was elected to the Baseball Hall of Fame.
Q_1	The MVP of [a] game Red Flaherty umpired was elected to the Baseball Hall of Fame. **Red Flaherty**: He umpired in World Series 1955, 1958, 1965, and 1970.
Q_2	The MVP of [a] game Red Flaherty umpired was elected to the Baseball Hall of Fame. Red Flaherty: He umpired in World Series 1955, 1958, 1965, and 1970. **1965 World Series**: It is remembered for MVP Sandy Koufax.
Q_3	The MVP of [a] game Red Flaherty umpired was elected to the Baseball Hall of Fame. Red Flaherty: He umpired in World Series 1955, 1958, 1965, and 1970. 1965 World Series: It is remembered for MVP Sandy Koufax. **Sandy Koufax:** He was elected to the Baseball Hall of Fame.

Figure 2.5: Claim verification example from HoVer (Jiang et al., 2020).

2.2.3 Evaluation metrics

The most popular evaluation metrics in question answering are *exact match* and *F1 score*. These were used when Rajpurkar et al. (2016) and Yang et al. (2018) published the datasets SQuAD respectively HotpotQA to measure the performance on these datasets.

Exact match (EM) When calculated over a whole dataset, it measures the percentage of predictions that matches the corresponding correct answer exactly.

F1 score A measure of a test's accuracy, where the *test* is the performance of a QA system. In question answering, the F1 score measures the average overlap between the correct answer and the predicted one.

The F1 score is calculated as:

$$F1 = \frac{2 \cdot P \cdot R}{P + R}, \tag{2.1}$$

where P stands for *precision* and R for *recall*. These, in their turn, are calculated according to

$$P = \frac{tp}{tp + fp} \quad \text{and} \quad R = \frac{tp}{tp + fn}. \tag{2.2}$$

tp, fp, and fn, respectively stand for *true positive*, *false positive* and *false negative*. In natural language processing, tp corresponds to the number of tokens that are shared between the correct answer and the prediction. fp measures the number of tokens that are in the prediction but not in the correct answer. Last, fn describes the number of tokens that are in the correct answer but not in the predicted one.

Yang et al. (2018) suggest two new settings for calculating the F1 score and exact match in the multi-hop setting with their dataset HotpotQA. This is to assess the explainability of the models. The first extra measurement is to evaluate the model's performance on finding the right supporting facts. The F1, respectively the EM, are then calculated on the set of the predicted supporting facts, compared to the correct ones. These are called *support* F1 respectively EM. The second proposed measurement is a combination of the evaluations on the answers and the supporting facts. These are called the *joint* EM and F1. Joint EM is 1 if both the answer EM and supporting facts EM are 1, otherwise, it is 0. The joint F1 is calculated as:

$$\text{Joint F1} = \frac{2 \cdot P^{joint} \cdot R^{joint}}{P^{joint} + R^{joint}}, \tag{2.3}$$

where the precision and recall of the joint F1 is calculated from the precision and recall from the answer respectively the supporting fact according to

$$P^{joint} = P^{ans} \cdot P^{sup} \quad \text{and} \quad R^{joint} = R^{ans} \cdot R^{sup}. \tag{2.4}$$

Two other metrics that have been used by several authors of openQA systems are *precision at k* and *recall at k*. They are mainly used to evaluate models' ability to identify paragraphs or documents that include the correct answer to a given question. Precision and recall at k are defined as follows:

Precision at k (P@k) Describes the proportion of items in the top-k ones that are relevant.

Recall at k (R@k) Describes the proportion of relevant items that are among the top-k ones.

2.3 Neural open-domain QA systems

Chen et al. (2017) were probably the first ones who included neural techniques in the open-domain question answering task. Since then, the usage of neural networks in some form has become the state-of-the-art way of tackling openQA. In this section, I explain some of the key components and different designs of neural open-domain question answering systems.

2.3.1 Retriever-reader

Chen et al. (2017) proposed a model architecture consisting of two modules, one for retrieving documents and one for reading the retrieved ones. This retriever-reader design has afterward been adopted and extended in numerous open-domain QA systems. The retriever in the system, called DrQA, uses information retrieval (IR) techniques and retrieves the most relevant documents according to the TF-IDF metric. The neural technique is only incorporated in the reader in DrQA, which is a recurrent neural network (RNN). The RNN is applied to each retrieved paragraph in turn together with the question. It predicts which span in the current paragraph is most probably the answer and calculates a score of how likely it is the answer. Finally, the scores of all possible answer spans are compared, and the one with the highest is returned as the predicted answer.

2.3.2 Reranker

Wang et al. (2017) proposed a new pipeline for open-domain question answering, named *reinforced ranker-reader* (R^3). Their contributions are mainly two things: they introduced the *ranker* component to complete the retriever and reader, and they proposed to jointly train the ranker and reader model with reinforcement learning. R^3's retriever adopts, just as DrQA, a traditional IR technique. For each input question, the retriever finds the top-200 articles according to the BM25 metric. Then these articles are split into sentences and the 200 ones that best match the question when comparing with TF-IDF are then passed on in the pipeline.

2.3.3 Pre-trained language models

Pre-trained language models have recently become standard to use in at least one module in open-domain question answering systems. Devlin et al. (2018) published a new type of pre-trained language model that has been adopted and extended in numerous ways. They named it BERT, *bidirectional transformers for language understanding*. BERT is a language representation model with a few new features compared to earlier models. First, it has a *masked language model* (MLM) pre-training objective. That consists of randomly masking out some of the input tokens, and the objective is to predict the masked out token based on the context. Second, the model is *bidirectional* as mentioned in the name. That means the MLM can make use of the context both to the left and right of the masked out token.

When released, BERT pushed forward the state-of-the-art score of several NLP benchmark tasks. On the SQuAD1.1 test set, the F1 score obtained 1.5 points absolute improvement, and on the SQuAD2.0 test set, the F1 was improved by 5.1 points. Since Devlin et al. (2018) published BERT, a large number of extensions, as well as applications of the model have been released. Most open-domain question answering systems include pre-trained language models in at least one module since they have shown to be very powerful.

Partly inspired by BERT, Clark et al. (2020) published their pre-trained language model named ELECTRA. The main difference compared to BERT is that instead of masking out tokens, ELECTRA replaces them with plausible alternatives. The objective when training this model becomes then to identify which token that has been replaced. The part of ELECTRA that handles the replacement of tokens is called the *generator*, and the part that is supposed to identify the replaced ones is the *discriminator*. ELECTRA is mainly used in the reader module of many recent openQA systems.

2.3.4 Sparse and dense retrievers

Initially, retrievers were implemented with classic IR methods, using sparse vector space models such as TF-IDF or BM25 (Chen et al., 2017; Kratzwald and Feuerriegel, 2018). In these models, words in the question and the context are matched with an inverted index. It can be seen as a representation of the question and the context in a high-dimensional, sparse vector space. These types of retrievers are therefore generally referred to as *sparse retrievers*. While they are both efficient and easy to implement, they cannot handle semantics beyond term matching. This means that they cannot handle synonyms in questions or understand the difference between two or more senses of a word.

Different architectures of so-called *dense retrievers* have been presented in order to obtain a better semantic representation (Karpukhin et al., 2020; Das et al., 2019). These adopt a dense, latent semantic encoding of the questions and context, which ideally results in synonyms being mapped to vectors close to each other since their meanings are the same. The dense encodings are also more flexible in the way that they are learnable by adjusting the embedding functions.

Lee et al. (2019) were first to adopt a dense retrieval method that outperformed the ones based on TF-IDF and BM25 metrics in their *open retrieval question answering* (ORQA) model. ORQA's retriever and reader are jointly trained on the question answering task. The retriever part is proposed to be pre-trained to solve an unsupervised task that is similar to evidence retrieval in openQA. In more detail, the task to be pre-trained on is an *inverse cloze task* (ICT). In the cloze task, the goal is to predict masked-out text given the context. Here, the inverse is requested: that given a sentence, the context should be predicted.

A significant difference between ORQA and other preceding work on improving retrieval in openQA systems is the *open retrieval* that ORQA adopts, in contrast to retrieving a closed set of evidence. After pre-training the retriever on the ICT, the joint model is end-to-end fine-tuned by optimizing the marginal log-likelihood of correct answers that were found.

However, Karpukhin et al. (2020) pointed out two weaknesses with ORQA:

- First, the ICT pretraining is computationally intensive and it is not clear that regular sentences are good alternatives to questions in the objective function.

- Second, since the context encoder is not fine-tuned using question-answer pairs, the corresponding representations could be sub-optimal.

Therefore, Karpukhin et al. (2020) wanted to see if a better dense embedding model could be trained using only question-passage pairs without extra pre-training. They succeeded in proving this, by leveraging a pre-trained BERT model, where the embedding is optimized for maximizing inner products of the question and relevant passage vectors. Karpukhin et al. (2020) also verified that, in the context of openQA, a higher retrieval precision leads to higher end-to-end accuracy.

2.3.5 Adaptive retrievers

Kratzwald and Feuerriegel (2018) addressed the fact that the interplay between the retriever and reader in open-domain question answering is poorly understood. They showed that retrieving a fixed number of documents suffers from a noise-information trade-off which is suboptimal. Kratzwald and Feuerriegel (2018) proposed a novel design that adaptively selects the optimal number of documents to retrieve. The retrieval model learns the optimal number conditioned on the size of the corpus and the query.

2.3.6 Answer verification

With the arrival of datasets including unanswerable questions, demands were put on systems to handle these. Zhang et al. (2020) suggested that the machine reading comprehension task including unanswerable questions could be divided into two subtasks: answerability verification and reading comprehension. This requires stronger MRC models in openQA systems, generally implying stronger readers. A common reader consists of two parts: an encoder and a decoder. Pre-trained language models have recently been dominating as encoders since they have a strong capacity for capturing the contextualized sentence-level language representation. The decoder represents the task-specific part in an MRC system, like question-passage interaction in question answering systems.

One popular solution to handle unanswerable questions was to include some verification module that could determine answerability. The verification module was generally stacked along with the encoder or the decoder. This showed to be suboptimal and Zhang et al. (2020) investigated better designs of the verifiers. Their *retro-reader* has two stages of reading and verification: first, sketchy reading that yields an initial judgment, and second, intensive reading that verifies and returns the final predicted answer. The sketchy reading module includes embedding, interaction, and an external front verification. In the intensive reading module, there are question-aware matching, span prediction, and internal front verification.

2.4 Multi-hop question answering systems

The *retrieve and read* approach to open domain question answering is widely used. However, many systems adopting this approach can only answer questions, where the relevant context can be obtained in one retrieval step. Several authors have identified the need for systems that can handle more complex questions, often referred to *multi-hop questions* or questions requiring *multi-step reasoning*.

2.4.1 Sparse, iterative retrievers

Multi-hop open domain question answering is a popular area of research, but it has not been investigated for very long. Many of these systems are built on the retriever-reader architecture that is well used in the single-hop setting but with some addition that makes it suitable for multi-hop questions.

The general solution is to have an iterative retriever, which retrieves relevant documents in multiple steps. What, in this context, can be considered as early multi-hop QA systems generally adopted a sparse retriever as their iterative one.

Along with one of the first large-scale multi-hop question answering datasets, HotpotQA, Yang et al. (2018) released a baseline model. This was in turn a reimplementation of the model by Clark and Gardner (2017). In addition to this original model, Yang et al. (2018) also included character-level models, self-attention, and bi-attention in the baseline.

GoldEn Retriever (Qi et al., 2019) is another early multi-hop question answering system, where *GoldEn* is a shortening of *Gold Entity*. The GoldEn retriever iterates between reading context and retrieving more supporting documents. It is made up of the following main parts:

Query generator From the question and available context in each iteration, a natural language search query is generated to be able to retrieve more supporting documents. In the first hop, only the original question is used to generate a search query. The natural language search query guarantees interpretability to the provided answers. The query generator is built on the DrQA reader module (Chen et al., 2017) and formulated as a question answering task. The module consists of a neural network, which in the GoldEn retriever is trained to select a span from the already retrieved context, given the original question.

Retriever The document retriever in the GoldEn retriever makes use of an index including all introductory paragraphs from the English Wikipedia. When retrieving, the BM25 ranking function is used, and documents whose titles match the search query are boosted. This is to obtain a better recall for entities with common names.

Reader The reader component in GoldEn retriever is based on the baseline model from Yang et al. (2018), where all retrieved documents are processed separately with shared encoder RNN parameters to obtain paragraph order-insensitive representations for each document. Possible answer spans are then predicted from each document and probabilities are calculated for each of them to select the most probable one.

Qi et al. (2019) claim that GoldEn retriever is much more efficient, scalable, and interpretable at retrieving gold documents compared to its neural retrieval counterparts. This is because GoldEn does not rely on a QA-specific IR engine tuned to a specific dataset. One major challenge however is to train query generation models efficiently.

In the *iterative retriever, reader and reranker* (IRRR), Qi et al. (2020) included a retrieving approach that is similar to the one proposed by Qi et al. (2019). IRRR aims at building a reasoning path from the question, through all the necessary supporting documents, to the answer. It loops between retrieving, reading, and reranking to expand the reasoning path with new documents. Just as the GoldEn Retriever, it uses natural language search queries.

These natural language search queries are sent to a text search engine. Elasticsearch is used to search for relevant documents, to reduce the context size for the transformer encoder in IRRR. Similarities between both question and paragraph, respectively question and article are calculated to select the most relevant context. The standard BM25 similarity function, respectively an extended version, is applied in these calculations. The extended version takes the square of the IDF term and sets the TF normalization term to zero.

2.4.2 Dense, iterative retrievers

Dense retrieval techniques have also been adapted to the multi-hop question answering setting. Feldman and El-Yaniv (2019) proposed a solution named MUPPET, *multi-hop paragraph retrieval*. It consists of two main parts: a paragraph and question encoder and a paragraph reader.

The encoder generates dense encodings and is trained to encode paragraphs into d-dimensional vectors and questions into search vectors in the same vector space. The *maximum inner product search* (MIPS) algorithm is applied to find the most similar paragraphs to a given question. The selected paragraphs are then sent to the paragraph reader which extracts the most probable answer, given the posed question.

Xiong et al. (2021) used a similar recursive dense retrieval approach in their *multi-hop dense retrieval* (MDR) system. They adopted a different query reformulation technique, where they concatenate the original question and retrieved documents as input to the query encoder. Just as Feldman and El-Yaniv (2019)'s work, MDR uses the MIPS method to search for relevant documents in the retrieval.

Khattab et al. (2021) also argues that multi-hop QA systems need highly expressive query representations. They point out that multi-hop questions include multiple information needs, like information from several different paragraphs. However, they mean that earlier systems like MDR (Xiong et al., 2021) have a limited capacity to model open-domain questions in general and multi-hop questions specifically. The single dense vector encoding for each sentence that MDR adopts is not sufficient, they argue. Instead, they propose *late interaction*, which uses a vector for each constituent token.

2.4.3 Graph retrievers

Graph-based approaches to question answering have been proposed in several systems. These systems are often made for multi-hop datasets, such as HotpotQA, which encourage reasoning based on a chain on for example entities.

Ding et al. (2019) introduced a *cognitive graph QA* (CogQA), that builds a cognitive graph by iterating between two systems: one implicit extraction module and one explicit reasoning module. Given a question:

1. The first system extracts relevant entities and answer candidates from paragraphs. These are organized in a cognitive graph, where the nodes are either entities or answer candidates. It is implemented with a BERT model.

2. The second system conducts reasoning over the graph and finds clues that the first system uses to extract next-hop entities.

Iteration between the two systems continues until all possible answers are found, and then the final answer is chosen based on the reasoning from system two.

Asai et al. (2019) proposed a similar solution in their *graph recurrent retriever* (GRR) model. The retriever part in GRR is an RNN that is trained to sequentially retrieve evidence paragraphs to the reasoning path, conditioned on the previously retrieved information. In each iteration, the model selects a paragraph based on the hidden state of the RNN. The iteration continues until an end-of-evidence symbol is selected, allowing the model to create reasoning paths of arbitrary lengths. When the reasoning is done, the graph is passed on to the system's reader module, which extracts possible answer spans from the reasoning paths and then reranks the paths by computing the probability that they include the answer. The reader is a BERT model that is fine-tuned for this purpose.

2.4.4 Question decomposition

Talmor and Berant (2018) presented a model for answering complex questions through decomposition. The question is decomposed into a sequence of simpler questions, which are then posed to a search engine. Answers to each one of the simpler questions are extracted from the search results, and the final answer is chosen by computing symbolic operations.

Min et al. (2019) adopts a similar approach in their *DecompRC* model. The questions are first decomposed and then answered by off-the-shelf single-hop reading comprehension models. They identified four different *reasoning types* in multi-hop questions: *bridging*, *intersection*, *comparison* and *keep the question as originally formulated.*

A posed question is decomposed according to the four reasoning types, and each sub-question is then passed on to a single-hop reading comprehension system. The answers to the sub-questions are combined based on the respective reasoning type. At last, a decomposition scorer decides which decomposition is the most suitable and returns the corresponding answer as the final one.

In contrast to the supervised decomposition of questions in DecompRC (Min et al., 2019), Perez et al. (2020) instead suggested an unsupervised question decomposition procedure. Perez et al. (2020) point out that labeling questions with decomposition is difficult, which motivates an unsupervised approach. They presented an algorithm for one-to-N unsupervised sequence transduction that learns to map one complex question into N simpler, single-hop, sub-questions. A recomposition model is trained to combine the answers to the sub-questions into a final answer, given the original input.

2.5 Domain adaptation

Wiese et al. (2017) published a paper concerning neural domain adaptation for biomedical question answering. They pointed out that neural question answering systems had not been applied to more specialized domains due to the lack of large enough datasets to train a neural network from scratch. To create a dataset for training for a specific domain would be very expensive because of the need for domain experts. Therefore, they establish that this approach is not desirable.

The domain adaptation approach Wiese et al. (2017) propose is shortly described to apply various transfer learning techniques to a neural QA system that is trained on a large open-domain dataset, such as SQuAD. With transfer learning, or *fine-tuning*, the system is adapted to a biomedical dataset.

A little more detailed description of the procedure is that they suggest pre-training the model on SQuAD, using the token F1 score as the training objective. The result from the pre-training is referred to as the base model. Next, the fine-tuning is initialized with the model parameters from the base model, and the optimization of the objective is resumed with the biomedical dataset BioASQ, but with a lower learning rate.

The architecture is based on a state-of-the-art QA system, which Wiese et al. (2017) extended with biomedical word embeddings. A brief overview of it is described in figure 2.6. The system does not rely on ontologies, parsers, or entity taggers that are domain-specific, since they are expensive to create. With this approach, they manage to achieve state-of-the-art performance on biomedical QA.

Hazen et al. (2019) also discussed domain adaptation from limited data for question answering. Just as Wiese et al. (2017) also pointed out, they mention that there are few question answering datasets in specialized domains. With their work, Hazen et al. (2019) show that transfer learning techniques as fine-tuning a pre-existing model work surprisingly well. Their work focus on domain data from car manuals. An example of a question-answer pair used in their study is shown in figure 2.7. One of their conclusions is that question answering models based on transformer encoders like BERT, that are trained on large amounts of general-domain data, can efficiently be adapted to new domains with limited data.

Qi et al. (2020) mentioned a few properties of open-domain question answering systems that make them unsuitable for domain adaptation. They identify three assumptions that

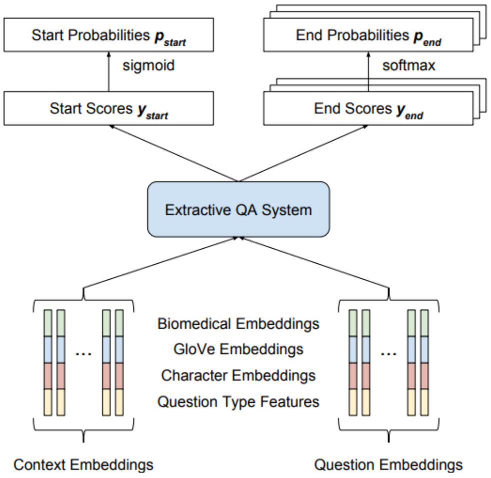
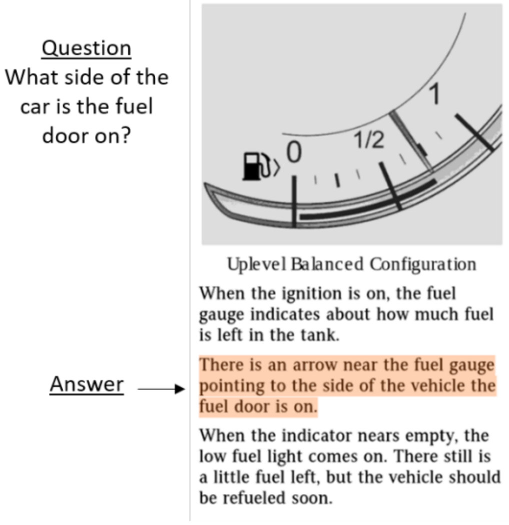


Figure 2.6: Overview of model architecture by Wiese et al. (2017).


Question
What side of the car is the fuel door on?
Answer
0
1/2
1
Uplevel Balanced Configuration

When the ignition is on, the fuel gauge indicates about how much fuel is left in the tank.

There is an arrow near the fuel gauge pointing to the side of the vehicle the fuel door is on.

When the indicator nears empty, the low fuel light comes on. There still is a little fuel left, but the vehicle should be refueled soon.

Figure 2.7: Question-answer example used in domain adaptation by Hazen et al. (2019).

these systems make:

1. Access to a well-tuned parameterized retrieval system that helps navigate the large amount of text.

2. Access to non-textual metadata such as knowledge bases, entity linking, and Wikipedia hyperlinks when retrieving supporting facts is assumed.

3. That every input question in the dataset is either single-hop or multi-hop, and tailor the approach to one of these either in model design or training.

The mentioned properties work as a motivation for their proposed model, that lacks these specific attributes.

Chapter 3
Method

In this thesis, I implemented and evaluated a telecom domain adaptation on two single-hop openQA systems and one multi-hop version. To be able to evaluate the performance of the single-hop systems in the telecom domain, I and a group of people from Ericsson built a telecom question answering dataset. Besides myself, the group consisted of Fitsum Gaim Gebre, Henrik Holm, Vincent Huang, and Jieqiang Wei. The dataset is called *TeleQuAD*, short for *telecom question answering dataset*. I also built a smaller multi-hop question answering dataset, called *mTeleQuAD*. This was to evaluate the performance of the multi-hop question answering system on telecom domain data. In this chapter, I will go through these datasets, the mentioned models, and how I conducted the evaluations.

3.1 TeleQuAD

TeleQuAD is short for *telecom question answering dataset*, and it is the question answering dataset that I built, together with the four other people in the Telecom AI project within GAIA, Ericsson. Our intention with it was to evaluate telecom language understanding.

The data was mainly collected from 3GPP, a partnership project that unites telecommunications standard development organizations to produce reports and specifications for mobile telecommunication. Data were both scraped from their website and their product specifications. Some data was also scraped from Sharetechnote, a website that gathers information regarding different aspects of telecommunication. Fitsum Gaim Gebre, the technical lead of the project group, took care of selecting a couple of hundred source documents which he cleaned by removing figures, tables, code snippets, etc. He split up each document into paragraphs, and also set up a web interface where we annotated the questions and answers.

We did the annotation of the answers in the dataset by marking out text spans in the different paragraphs. To each answer, we then manually wrote a corresponding question. All the question-answer pairs were divided into three different types:

Short Phrases only including one cohesive answer.

List An enumeration of things; answers containing for example commas or the word *and*.

Multi-hop Questions that require more than one paragraph to be answered.

The annotation of the dataset was arranged so that you would first select one of the source documents, then read through it and when you found a suitable formulation you would annotate the answer and question. If you found no relevant content in the document, you would mark it out as *remove*.

The full dataset includes 2021 question-answer pairs. The formatting of the dataset follows the one of SQuAD (Rajpurkar et al., 2016, 2018), to make it compatible with existing QA systems. Figure 3.1, 3.2, 3.3 show overviews of, respectively, the length of the paragraphs, the questions and the answers counted in tokens. Figure 3.4 explains the shares of different answer types in the dataset.

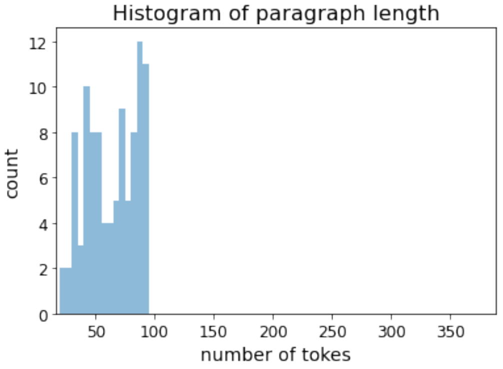


Figure 3.1: Paragraph lengths in TeleQuAD.

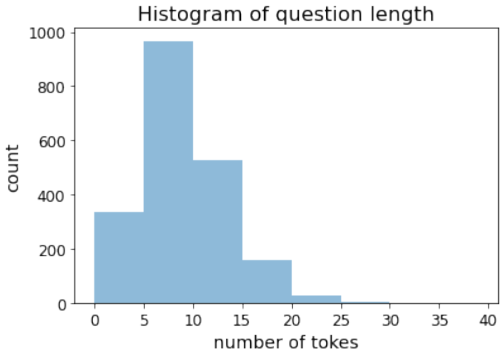


Figure 3.2: Question lengths in TeleQuAD.

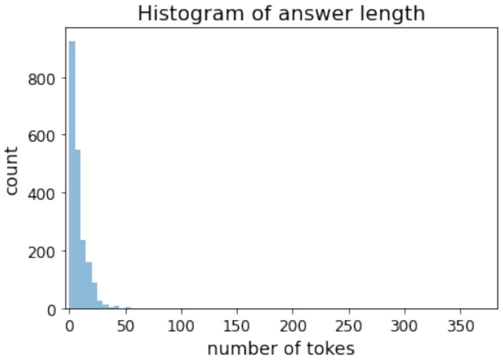


Figure 3.3: Answer lengths in TeleQuAD.

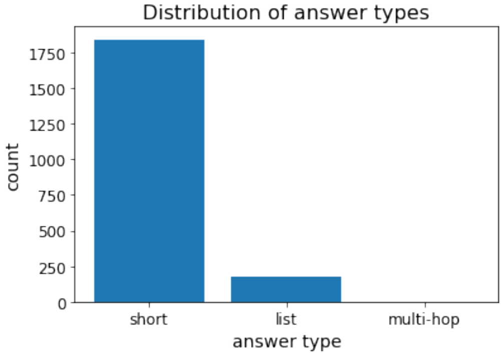


Figure 3.4: Answer types in TeleQuAD.

Table 3.1 includes the mean value of the number of tokens in context paragraphs, questions respectively answers for SQuAD1.1 development set and TeleQuAD. The tokenization for these comparisons is a simple whitespace one, meaning that the text is divided into tokens by the whitespaces.

	SQuAD1.1 dev	TeleQuAD
Paragraphs	122.8	158.8
Questions	10.2	8.6
Lengths	3.0	8.2

Table 3.1: Mean length in number of tokens for content in SQuAD1.1 dev set and TeleQuAD.

To extend the usage of TeleQuAD, we split it into a development (dev) set and a test set. More specifically, the dev set is supposed to be used when fine-tuning language models or other neural machine reading comprehension systems. Naturally, the test set is then used to evaluate with. The idea was to split the total dataset into 50/50 dev and test set. We did not want text documents to correspond to questions in both subsets and therefore made the split according to the context documents. Since the documents contained a different number of questions, the split did not become exactly 50/50. Instead, the dev set has 1018 question-answer pairs and the test set has 1003 ones.

3.2 mTeleQuAD

In addition to the single-hop TeleQuAD, I created a small multi-hop telecom question answering dataset, which will be referred to as *mTeleQuAD*. The data I annotated come from 3GPP specifications of the same type that were used in TeleQuAD. I annotated the multi-hop version similarly to HotpotQA (Yang et al., 2018), to make it compatible with other multi-hop QA systems. I created 50 question-answer pairs in total, where each has two *supporting paragraphs*: paragraphs that contain the required information.

Inspired by HotpotQA, there are two different types of questions in mTeleQuAD. These are *bridge* and *comparison*. In the former one, the original questions include *one* supporting fact, that leads to a certain paragraph. The original question together with the information in this paragraph refers to a new entity called *bridge entity*. In that one, the final answer is found. The comparison questions instead include two supporting facts, which refer to two different entities. Information from both needs to be identified and compared in some way to be able to produce the answer.

Table 3.2 shows some statistics for mTeleQuAD and the comparable numbers for HotpotQA dev set. These are the mean lengths in tokens for different parts of the dataset. The tokenization is made by dividing the strings at the whitespaces.

	HotpotQA dev	mTeleQuAD
Paragraphs	509.1	65.0
Questions	15.7	14.3
Lengths	2.5	3.3

Table 3.2: Mean length in number of tokens for content in HotpotQA dev set and mTeleQuAD.

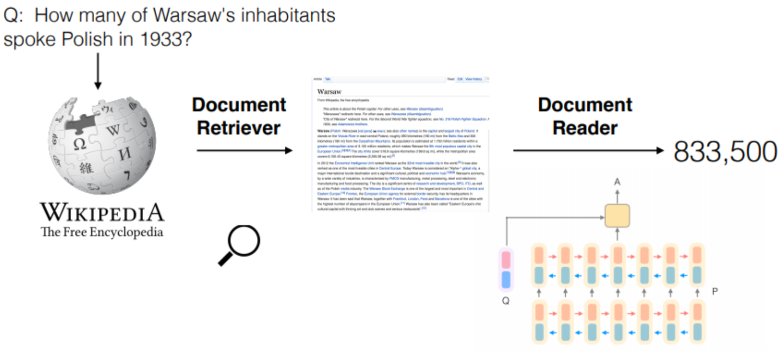


Figure 3.5: Conceptual view over DrQA (Chen et al., 2017).

3.3 DrQA

DrQA (Chen et al., 2017) is a well-known system in open-domain question answering. It introduced a new state-of-the-art design of open-domain question answering systems by including neural techniques and the retriever-reader architecture. Since DrQA introduced a new state-of-the-art way of designing openQA systems, it has many times been used to compare other models. This is one of the reasons why I chose to investigate this model. After looking into the architecture, I also realized that it would be quite easy to adapt to a new domain. It does not rely on for example hyperlinks or similar that our telecom data lacks. In addition, the repository where the code to DrQA is published is quite well commented, making it easy to understand and run on my own instance.

DrQA consists of two modules: a retriever that adopts a classic IR technique and a reader in the form of an RNN. Figure 3.5 shows a brief overview of the architecture. The reader aims to return the most relevant documents for a specific question, i.e. to narrow down the search space for the answer. The search for relevant documents is made through a simple inverted index, and the articles and the question are compared as TF-IDF weighted bag-of-word vectors. The local word order is taken into account with bigram features.

When the retrieved documents are passed on to the reader, each paragraph in the documents is considered one at a time. All tokens in a specific paragraph are represented as a sequence of feature vectors before being passed as input to the RNN, which is a multi-layer bidirectional *long short-term memory network* (LSTM). The encoding of each token is obtained by concatenating each layer's hidden units in the end. The feature vector of each token consists of four parts:

Word embeddings The 300-dimensional Glove word embeddings (Pennington et al., 2014) are used in the first feature vector part. This embeds a token p_i as $\mathbf{E}(p_i)$. The 1,000 most frequent question words are fine-tuned since they are significant for question answering, and the rest is kept from the pre-training.

Exact match In the exact match, three binary features are used which show if the token can be exactly matched to one of the question words in, respectively, its original, lowercase, or lemma form.

Token features The token features include the token's part-of-speech, named entity recognition, and normalized term frequency.

Aligned question embedding Last, the aligned question embedding is calculated from the attention score which captures the similarity between the current token p_i and each question word q_j. This is supposed to capture similarities between similar but not identical words as a complement to the exact match. The aligned question embedding for a token p_i is calculated as $\sum_j a_{i,j}\mathbf{E}(q_j)$. When $\alpha(\cdot)$ signifies a single dense layer with ReLU nonlinearity, $a_{i,j}$ is given by

$$a_{i,j} = \frac{\exp\left(\alpha\left(\mathbf{E}\left(p_i\right)\right) \cdot \alpha\left(\mathbf{E}\left(q_j\right)\right)\right)}{\sum_{j'} \exp\left(\alpha\left(\mathbf{E}\left(p_i\right)\right) \cdot \alpha\left(\mathbf{E}\left(q_{j'}\right)\right)\right)}. \tag{3.1}$$

The question posed to the system is encoded by using the word embeddings of each respective token q_j as input in an RNN. The resulting hidden units $\mathbf{q}_j$ are then combined into one single vector $\mathbf{q}$ with a weighted sum, where the weights b_j represent the importance of each question word. The weight of a specific hidden units vector $\mathbf{q}_j$ is calculated as

$$b_j = \frac{\exp(\mathbf{w} \cdot \mathbf{q}_j)}{\sum_{j'} \exp(\mathbf{w} \cdot \mathbf{q}_{j'})}, \tag{3.2}$$

.
where $\mathbf{w}$ is a learned weight vector.

When the encoding of the paragraph and question is complete, the span of tokens which is most probably the answer will be predicted. To obtain this, the paragraph vectors and the question vector are used as inputs in two classifiers, trained to predict the start respectively the end of the span. A bilinear term is used to capture the similarity between the paragraph token $\mathbf{p}_i$ and the question $\mathbf{q}$. The probability of each token being start and end is computed as:

$$\begin{aligned} P_{\text{start}}(i) &\propto \exp\left(\mathbf{p}_i \mathbf{W}_s \mathbf{q}\right) \\ P_{end}(i) &\propto \exp\left(\mathbf{p}_i \mathbf{W}_e \mathbf{q}\right). \end{aligned} \tag{3.3}$$

The prediction is chosen such that $P_{\text{start}}(i) \times P_{\text{end}}(i')$ is maximized, and the span has a maximum length of 15 tokens. The unnormalized exponential is used to make the scores compatible between paragraphs in different documents. When all paragraphs in the retrieved documents have been processed and each has one predicted answer span, the argmax is calculated over all potential answer spans and one final span is returned.

3.4 CdQA

Closed-domain question answering (cdQA) (Farias, 2019a) is an end-to-end open-source software suite for question answering. The cdQA system was built to be used by anyone who wants to have a closed-domain question answering system (Farias, 2019b). The architecture of cdQA is a retriever-reader, where the retriever is designed just like the one in DrQA (Chen et al., 2017).

The reader is a Pytorch version of BERT that was published by Huggingface (Wolf et al., 2020). To adapt the BERT model for question answering, it was pre-trained on SQuAD1.1 before used in cdQA. The reader outputs the most probable answer it can find in each paragraph that is sent to it. At last, there is a final layer that compares the probable answers through an internal score function and then returns the most likely one.

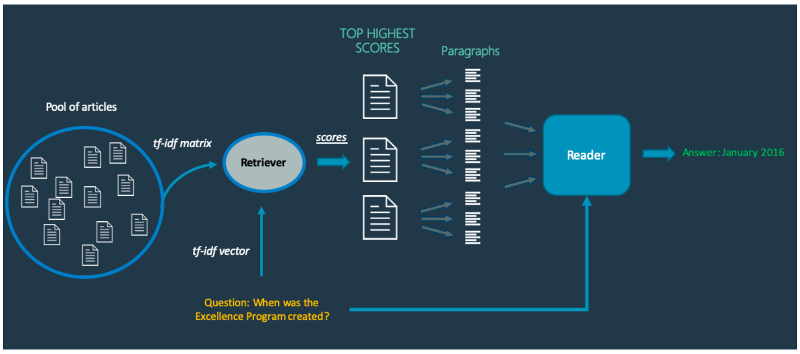


Figure 3.6: Overview of cdQA architecture (Farias, 2019b).

When setting up the cdQA pipeline, a corpus with all the context paragraphs is sent to the retriever. It trains the TF-IDF matrix from the content. There is a possibility to fit the language model to a dataset, i.e to fine-tune the reader. A sketch of the design of cdQA is drawn in figure 3.6.

CdQA has the classic retriever-reader architecture, but with a more elaborate reader than DrQA. Since pre-trained language models have become very popular to use as readers, I found it interesting to investigate a model that included one. With the retriever being similar to DrQA's, my thought was to make some comparisons among them to understand a bit about how much a powerful language model could contribute.

3.5 MDR

Xiong et al. (2021) published a model called *multi-hop dense retrieval* (MDR). The authors pointed out the issue with multi-hop open domain question answering that the search space grows exponentially with each retrieval hop. They wrote that usually, recent work deals with this by constructing a document graph using entity linking or the hyperlink structure in Wikipedia. Even though these models show great results, they may not generalize to new domains very well, since links in documents may not exist. This is a strong reason why I chose this multi-hop model for telecom domain adaptation.

The authors also refer to drawbacks with sparse representations such as term-based information retrieval that GoldEn Retriever (Qi et al., 2019) adopts. It does not capture semantics beyond lexical matching. In a dense representation, on the other hand, words, or tokens, with similar meanings are mapped to vectors close to each other.

The retriever in MDR will, given a multi-hop question q and a text corpus, retrieve a sequence of paragraphs $\mathcal{P}_{seq} : \{p_1, p_2, ..., p_n\}$. The number of retrieved paragraphs is fixed to a number k, which should be small enough to process the queries in a reasonable time while maintaining a large enough recall. The probability of selecting a specific paragraph sequence is modeled as:

$$P\left(\mathcal{P}_{seq} \mid q\right) = \prod_{t=1}^{n} P\left(p_t \mid q, p_1, \ldots, p_{t-1}\right). \tag{3.4}$$

For $t = 1$, the probability is only conditioned on the question, since no paragraphs have been retrieved yet. At each retrieval step, a new dense representation of the original question and the retrieved paragraphs are created. The maximum inner product search (MIPS) is used

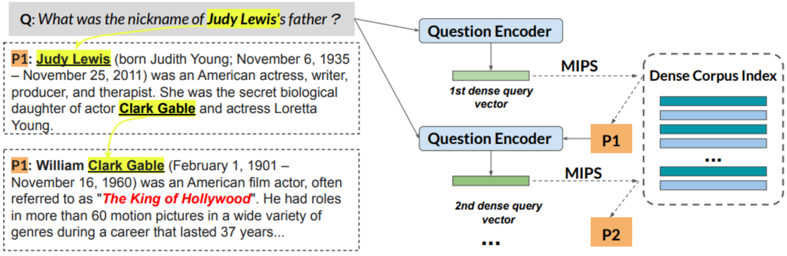

Q: What was the nickname of Judy Lewis's father ?
P1: Judy Lewis (born Judith Young; November 6, 1935 – November 25, 2011) was an American actress, writer, producer, and therapist. She was the secret biological daughter of actor Clark Gable and actress Loretta Young.
P1: William Clark Gable (February 1, 1901 – November 16, 1960) was an American film actor, often referred to as "The King of Hollywood". He had roles in more than 60 motion pictures in a wide variety of genres during a career that lasted 37 years...
Question Encoder
1st dense query vector
MIPS
Dense Corpus Index
...
Question Encoder
P1
MIPS
2nd dense query vector
...
P2

Figure 3.7: Conceptual view of retriever module in MDR (Xiong et al., 2021).

as the retrieval mechanism over the dense representations of the corpus. With $\langle \cdot, \cdot \rangle$ being the inner product between the query and paragraph vector, MIPS is calculated according to

$$P(p_t \mid q, p_1, \ldots, p_{t-1}) = \frac{\exp(\langle \boldsymbol{p}_t, \boldsymbol{q}_t \rangle)}{\sum_{p \in C} \exp(\langle \boldsymbol{p}, \boldsymbol{q}_t \rangle)}, \tag{3.5}$$

where $\boldsymbol{q}_t = g(q, p_1, \ldots, p_{t-1})$and $\boldsymbol{p}_t = h(p_t)$.

$g(\cdot)$ and $h(\cdot)$ are encoders that produce dense representations for the question and retrieved paragraphs respectively the paragraph p_t. The encoder in this case is a RoBERTa-based one (Liu et al., 2019) that is shared for both $g(\cdot)$ and $h(\cdot)$. Layer normalization is applied in MDR over the start token's representation from the encoder to get the final dense vectors.

This retriever module is trainable, which is something that is not possible with sparse retrievers. The goal of the training is to create a vector space, where vector encodings of questions and paragraphs with similar content will have a small distance. The training is set up so that each question is paired with a positive paragraph and a number of negative ones to approximate the softmax over all paragraphs. The positive paragraph is the gold annotated evidence, and the negative paragraphs are both paragraphs corresponding to other questions and ones that are not connected to the dataset.

After training the shared encoder to convergence, a copy of the encoder is saved. The copy is used as a new passage encoder, and a collection of paragraphs from different batches is encoded with it and utilized as negative paragraphs. With this new setup, the retriever is fine-tuned from the last saved checkpoint.

The full retrieval procedure is set up by encoding the whole corpus into an index of paragraph vectors. When a query is given to the system, all paragraphs are scored according to MIPS formulated in Eq. 3.5. Beam search is then used to determine the sequence of paragraphs containing the top k ones.

The pre-trained language model ELECTRA (Clark et al., 2020) is used as a reader in the MDR system. A few different language models were tried out as readers by the authors, both extractive and generative ones, and ELECTRA showed to perform the best.

When investigating possible multi-hop question answering systems to use for my evaluations, I both looked after something that achieved good results as well as would suit to apply to the telecom domain. As described in the previous work section of this work and also mentioned by Xiong et al. (2021), many published models do not fit domain adaptation. I concluded that it would be too difficult to choose a model that depends on some linking structure in its context documents. By the recently published multi-hop QA systems that competed among the best results on HotpotQA, MDR was one of very few that did not rely on links.

3.6 Experiments

In the following section I present how I set up my experiments for the single-hop models DrQA and cdQA respectively the multi-hop one, MDR.

3.6.1 DrQA and cdQA

The evaluations on DrQA and cdQA I made with both TeleQuAD and SQuAD1.1 development (dev) set. I used SQuAD in addition to TeleQuAD to make comparisons between the models' performance on a general domain dataset and a telecom domain one. Since TeleQuAD only includes answerable questions I reasoned that the fairest comparison would be to use SQuAD1.1, which also only includes answerable questions. In addition, DrQA cannot handle unanswerable questions. I used the dev set of SQuAD1.1 instead of the test set since the latter is not publicly available. The dev set includes 10,570 question-answer pairs

When evaluating DrQA with TeleQuAD, a few questions needed to be discarded. This was due to the design of the retriever, which relies on the TF-IDF metric. Some very common words are usually not considered when calculating TF-IDF. These are referred to as *stop words*, and these can for example be *a*, *an*, and *the*. The discarded questions only included stop words and could therefore not be processed. These were: *What is No?, What is AM?, What is RE?, What is M?* and *What is O?*.

My experiments with DrQA and cdQA were done with three different setups: the retriever, the reader, and the full pipeline. Each one of them had a few different settings:

Retriever I created a *paragraph* setting where only the paragraphs with corresponding questions were retrievable and a *distractor* setting where I added approximately ten times more telecom data to the retrieval corpus. Here, I used the full TeleQuAD dataset for evaluations.

I evaluated the retrievers with the metric precision at k, where k was set to 1, 5 respectively 10. k stands for the number of paragraphs retrieved for each question. A paragraph is considered to be retrieved correctly if it contains the answer span. This means that it does not necessarily must be the paragraph that was originally annotated for the current question.

Reader For the reader I used the *pre-trained* setting where the RNN or language model was kept at their pre-trained setting. In this setting, I used the full TeleQuAD for evaluation. In addition, I made a *fine-tuned* setting where the RNN respectively language-model were fine-tuned with the TeleQuAD dev set, and I used the TeleQuAD test set for evaluation.

Evaluations for the readers I decided to measure with F1 score and exact match (EM) since those are used in basically all openQA evaluations.

Full pipeline In the full pipeline evaluations, I combined the two different settings for both the retriever and the reader in all four possible combinations. For the evaluations with the pre-trained reader, I used the full TeleQuAD, and for the fine-tuned reader I only applied the test set.

The full pipeline evaluations are also measured in F1 and EM. In these experiments, I fixed the number of retrieved documents to 5.

For SQuAD I only ran one evaluation for each of the three setups. This was because I concluded that I would not be able to make fully comparable evaluations. I figured that my evaluations with SQuAD would only work as an overall comparison to a general domain dataset, and therefore I did not put more time into extending the settings for that dataset. So, all experiments that I did with SQuAD are with the corresponding *paragraph* setting for the retriever, meaning that only the paragraphs with annotated questions are included. For the reader part, I have only used the *pre-trained* settings. An additional reason for that is that I did not have access to the test set for SQuAD1.1 as earlier mentioned.

3.6.2 MDR

For the multi-hop model MDR I also wanted to compare evaluation results between the telecom domain and a general domain. Therefore, in addition to run evaluations on mTeleQuAD, I used the results that Xiong et al. (2021), the ones behind MDR, obtained on HotpotQA. I did not achieve any own results for evaluation on HotpotQA due to that the retriever index was too heavy to load on my instance. The HotpotQA development set that was used for evaluation by Xiong et al. (2021) contains 7,405 question-answer pairs, where each pair has two supporting paragraphs.

I ran my evaluations on MDR for three different setups:

Retriever In the retriever evaluation I included all paragraphs that corresponded to questions in the retriever corpus. The results are measured in recall at k. The encoder in the retriever module consisted of a pre-trained RoBERTa model.

Reader When evaluating the reader in MDR the two supporting paragraphs are supplied together with the question. The reader was a pre-trained ELECTRA model. The performance is measured in F1 and EM.

Full pipeline In the full pipeline evaluations there are some different settings for the HotpotQA dataset respectively mTeleQuAD. The accessible results for HotpotQA on MDR is with the *fullwiki* setting, meaning that the first paragraphs from all Wikipedia articles are available for retrieval. The ELECTRA reader was fine-tuned for HotpotQA in advance of the evaluation. For mTeleQuAD, on the other hand, I used a *paragraph* setting for the retriever and the pre-trained ELECTRA reader.

To follow the scores that Xiong et al. (2021) presented for the full pipeline evaluation, I measured the F1 score and EM for both the answer, the supporting paragraphs and calculated a joint score for these two.

Chapter 4

Results

In this chapter I present the results from my evaluations on the DrQA, cdQA respectively MDR models.

4.1 DrQA

Table 4.1 shows the results for the tests on the separate modules. The retriever was evaluated in two settings: *paragraphs* and *distractor* and in the two settings *pre-trained* and *fine-tuned*.

			SQuAD1.1 dev			**TeleQuAD**		
Model	Module	Setting	P@1	P@5	P@10	P@1	P@5	P@10
DrQA	Retriever	Paragraphs	92.1	97.2	97.8	80.9	96.7	98.1
DrQA	Retriever	Distractor				74.5	91.0	94.2
				F1	EM		F1	EM
DrQA	Reader	Pre-trained		73.7	63.4		64.6	40.2
DrQA	Reader	Fine-tuned					65.9	42.4

Table 4.1: Evaluation results for separate modules in DrQA.

For the retriever part in table 4.1, the precision at k increases when k is changed from 1 to 5 and from 5 to 10, which intuitively seems correct. The scores for TeleQuAD on this evaluation are lower than for SQuAD, except for $k = 10$ in the paragraph setting. This is even though the retrieval corpus for SQuAD is significantly larger. The SQuAD1.1 dev set includes 2067 different paragraphs, compared to TeleQuAD's 493 ones. I believe that a contributing reason for this is that there probably is less overlap between documents and paragraphs in SQuAD than in TeleQuAD. Meaning that there are more diverse topics included in the SQuAD corpus. Even though the corpora in open-domain question answering consist of *unstructured* text, you have the fact that the information on Wikipedia is sorted into different articles. In the telecom data that TeleQuAD is built from, on the other hand, the information is focused on a smaller domain. Therefore I find it easy to believe that there is a higher chance of finding the same information in multiple documents in the TeleQuAD corpus than in the SQuAD corpus.

The size of the retrieval corpus probably matters at some point. When the number of retrieved paragraphs is increased with a fixed number, it corresponds to a larger share of a small corpus compared to a bigger one. Therefore, I do not find it very surprising that P@10 is larger for TeleQuAD in the paragraph setting than for SQuAD.

In the *distractor* setting for TeleQuAD, the retriever reaches a 16.5 absolute point higher precision when the number of retrieved documents is increased from 1 to 5. When k is changed from 5 to 10 the increase in absolute points is less, 3.2 points, but still an increase. One easy way to improve the retrieval score is to simply retrieve more documents. But, the more retrieved documents the more work for both the retriever and reader and thus a slower model. The number of retrieved documents to chose from is a balance between having a precise and an efficient model.

The results of the reader module evaluations on DrQA also follow what I expected in advance. The fine-tuning of the reader achieved a higher F1 on TeleQuAD with 1.3 absolute points and a higher EM of 2.2 absolute points. The improvement is not huge, but since the fine-tuning only were conducted with approximately 1,000 examples, I would consider the improvement to be good. These scores are considerably lower than the ones for SQuAD. The evaluations on SQuAD1.1 dev set obtained 73.7 in F1 and 63.4 in EM. This is not surprising, since the reader module is pre-trained on the SQuAD1.1 training set, which one can assume are considerably more alike to the SQuAD1.1 dev set than TeleQuAD.

My results for the full pipeline evaluation of DrQA are presented in Table 4.2. There, the setting for each experiment is described along with the produced scores.

			SQuAD1.1 dev		**TeleQuAD**	
Model	Retriever	Reader	F1	EM	F1	EM
DrQA	Paragraphs	Pre-trained	47.0	41.1	30.5	20.9
DrQA	Distractor	Pre-trained			34.0	23.2
DrQA	Paragraphs	Fine-tuned			32.3	25.8
DrQA	Distractor	Fine-tuned			36.4	29.1

Table 4.2: Evaluation results for full pipeline of DrQA.

In the full pipeline evaluations on DrQA, no experiment with TeleQuAD outperforms the scores for SQuAD. This comes as expected. What is however a bit surprising is that the evaluations for the distractor settings obtain higher scores on both F1 and EM than for the paragraph settings with the same reader. With the pre-trained reader, the F1 score gains 3.5 absolute points and the EM 2.3 absolute points. For the fine-tuned reader the corresponding numbers are 4.1 for F1 and 3.3 for EM.

This is surprising since the retriever evaluations in Table 4.1 indicated that the distractor setting caused a lower precision in the retrieved documents. When only the retriever setting is changed from paragraphs to distractor in the full pipeline evaluation, I did expect the score to decrease due to this.

4.2 CdQA

Here follows the results from the evaluations on cdQA. In Table 4.3 are the scores for the different evaluations on the separate modules, the retriever respectively the reader.

			SQuAD1.1 dev			TeleQuAD		
Model	Module	Setting	P@1	P@5	P@10	P@1	P@5	P@10
cdQA	Retriever	Paragraphs	84.5	92.2	94.8	97.4	99.0	99.3
cdQA	Retriever	Distractor				97.2	99.0	99.2
				F1	EM		F1	EM
cdQA	Reader	Pre-trained		88.9	81.3		77.8	54.5
cdQA	Reader	Fine-tuned					80.7	59.7

Table 4.3: Evaluation results for separate modules in cdQA.

The evaluation results for cdQA yield some interesting observations. If we first take a look at the results from the retriever performance, I expected to get a larger difference in the scores between the paragraph and the distractor settings. These differences of 0.2 points for P@1 and 0.1 for P@10 are almost insignificant. With the distractor setting of around ten times as many paragraphs, I supposed that the precision would decrease compared to the smaller setting.

The reader results for cdQA, seen in Table 4.3, would I deem quite good. The F1 score of 77.8 and EM of 54.5 with the pre-trained setting are significantly higher than the corresponding ones for DrQA in Table 4.1. Not surprisingly are they much lower than for SQuAD, especially the EM is 26.8 absolute points lower for the pre-trained cdQA reader. Since this is pre-trained with the corresponding training set for SQuAD, it could only be expected.

When the reader is fine-tuned on the development set of TeleQuAD and evaluated on the test share, the results increase with 2.9 absolute F1 points and 5.2 absolute EM points. Considering that the fine-tuning only have been made with around 1,000 question-answer pairs I would say that this is really good. I do not think that it is unlikely that these scores could improve further if the reader would be trained with just a few times more samples.

For the evaluation of cdQA's full pipeline, I combined the two settings for both the retriever and reader. The final results for these experiments are presented in Table 4.4.

			SQuAD1.1 dev		TeleQuAD	
Model	Retriever	Reader	F1	EM	F1	EM
cdQA	Paragraphs	Pre-trained	65.8	59.6	53.8	37.8
cdQA	Distractor	Pre-trained			55.6	39.1
cdQA	Paragraphs	Fine-tuned			65.7	49.6
cdQA	Distractor	Fine-tuned			67.9	51.0

Table 4.4: Results for full pipeline of cdQA.

The full pipeline evaluation of cdQA, with the scores in Table 4.4, is interesting. First, there is a however expected, but still impressively large, improvement between the corresponding pre-trained and fine-tuned settings. With the paragraph retriever setting, the F1 score jumps 11.9 absolute points from 53.8 to 65.7 points. The EM score for the same setting rises from 37.8 to 49.6, which is an 11.8 absolute point increase. For the distractor retriever setting, the F1 increase is 12.3 and 11.9 for EM. I consider these very large, because of the small development set as mentioned before. This indicates that a question answering system can be domain adapted using only a relatively few samples for fine-tuning, supposing that the reader consists of a strong pre-trained language model.

The second thing is that for both reader settings, all the results for the distractor retriever settings are better than for the corresponding paragraph settings. The same thing occurred for DrQA's pipeline evaluations and is hard to explain also for cdQA. In the retriever evaluations for cdQA, there was only a small difference between the two settings. If it is considered significant, I would consider it to bring lower results for the distractor setting compared to the paragraph setting.

4.3 MDR

The results for the evaluations on the multi-hop model MDR are presented here. First, in Table 4.5, are the evaluations on the separate modules in the system.

			HotpotQA			**mTeleQuAD**		
Model	Module	Setting	R@2	R@10	R@20	R@2	R@10	R@20
MDR	Retriever	Pre-trained	65.9	77.5	80.2	52.0	80.0	86.0
				F1	EM		F1	EM
MDR	Reader	Pre-trained		76.2	63.4		80.3	58.0

Table 4.5: Evaluation results for separate modules in MDR. Results for HotpotQA by Xiong et al. (2021).

When looking at the performance of the retriever part of MDR in Table 4.5, there are some interesting differences between the results for HotpotQA and mTeleQuAD. The retriever is pre-trained on the HotpotQA training set, so I would have expected that both recall at 2 and at 10 would have been higher for HotpotQA. But recall at 10 is actually 2.5 absolute points higher for mTeleQuAD. One probable reason for that is the size of the retrieval corpus, which is a lot smaller for mTeleQuAD. The latter includes 100 paragraphs, and the dev set for HotpotQA includes a couple of thousand. The numbers are therefore not entirely comparable.

The reader module in MDR is built on the pre-trained language model ELECTRA. The evaluations on this part have also obtained some notable numbers, see Table 4.5. The general performance of the retriever on mTeleQuAD I consider to be very good, having in mind that the reader only is pre-trained. The F1 score of 80.3 and the EM of 58.0 is 0.5 respectively 3.5 absolute points higher than for the pre-trained cdQA reader evaluation.

Since the overall setting in the MDR case is slightly more difficult due to that there are two supporting paragraphs to reason over, it is impressive that the MDR reader still performs better. ELECTRA has performed better as a reader than BERT-like models in several cases, and my results also indicate that it has a better capacity when it comes to this task.

If these numbers are compared to those for HotpotQA, we see that the F1 score actually is higher for mTeleQuAD. I believe that a strong reason for that is the much smaller length of the mTeleQuAD paragraphs, almost ten times as short as for HotpotQA, see Table 3.2. Reasonably, it becomes easier to select the correct span in a given paragraph if it is shorter.

			Answer		Support		Joint	
Dataset	Retriever	Reader	F1	EM	F1	EM	F1	EM
HotpotQA	Fullwiki	Fine-tuned	75.3	62.3	80.9	57.5	66.6	41.8
mTeleQuAD	Paragraphs	Pre-trained	49.8	30.0	54.1	26.0	35.9	14.0

Table 4.6: Evaluation results for MDR. Results for HotpotQA by Xiong et al. (2021).

I only did the full pipeline evaluation of MDR in one setting, and the results from this are shown in table 4.6. The scores for HotpotQA and mTeleQuAD are not very comparable, since the settings are very different. For example, it is much expected that HotpotQA would return a lot higher scores since both the retriever and reader are trained on the training part of this dataset. I would expect this even though the evaluation is made with the fullwiki setting.

It is interesting to look at the scores that the evaluations on mTeleQuAD got since it indicates how difficult this actual dataset is. Would MDR perform almost as good on mTeleQuAD as on HotpotQA, my conclusion would be that mTeleQuAD is a too easy QA dataset, rather than that MDR is a great QA system. Therefore, I see it as something positive that the evaluation scores for mTeleQuAD were not higher than this. At the same time, I did expect even lower results for the answer F1 and EM, since I personally found the questions that I annotated to be really hard.

Chapter 5

Discussion

5.1 Retrievers

When I compared the results from the retriever evaluations for DrQA and cdQA, I expected to get almost the same results. This is since the cdQA retriever was implemented according to DrQA's. But, as can be observed when comparing the numbers between Table 4.1 and 4.3, they differ. The precision for the TeleQuAD evaluations is throughout higher for cdQA. Notably is P@1 16.5 absolute points higher for the paragraph setting and 22.7 absolute points higher for the distractor setting. But, the precision for SQuAD is instead higher for DrQA with a 7.6 absolute points difference.

I can come up with several reasons why these numbers differ, but I am not sure of the correct reason. The two systems have different ways of creating and handling databases for context documents. One possible reason for the difference could be that the documents are divided into paragraphs in different ways. If a system creates fewer paragraphs for the same corpus, it would reasonably be easier for it to retrieve the correct one, at least if you generalize over a large number of retrievals.

The retriever modules themselves are also implemented differently, but if that affects anything is hard to say. CdQA uses functions from Python's scikit-learn package in order to generate the TF-IDF vectors. DrQA's retriever, on the other hand, includes more functions that Chen et al. (2017) implemented themselves. However, since these retrievers adopt the same technique, these differences should reasonably not cause any differences. One more probable reason is the fact that I implemented the evaluation function for cdQA's retriever myself since there were no such in the original repository. There is a risk that I did not make it work exactly like the retriever evaluation script for DrQA. I did, however, make use of functions from the DrQA repository when implementing the script to make them as alike as I could.

Regardless of the difference in the performance, these non-neural, TF-IDF-based retrievers did generate good results. One strength of this technique is that it easily can be applied to a new domain of a corpus, without requiring any training.

For MDR, the retriever performance is measured in the recall at k instead of precision at k since two paragraphs are required to be retrieved. The scores for this module, which are presented in Table 4.5, are not fully comparable to the corresponding ones for DrQA and cdQA. But without drawing any certain conclusion, the retriever in MDR seem to perform worse than the other two on their respective telecom dataset. However, MDR's retriever has a completely different design with its dense representations and trainable architecture, which I have not been taking advance of. When counting in that the MDR retriever presumably would perform better if fine-tuned on the telecom domain, I consider these results as quite good.

In my project, I have both experimented with sparse representation retrievers and dense representation ones. My impression from the latest research within open-domain question answering is that there is no consensus concerning whether sparse or dense retrieval techniques are the best ones. Models containing both types are still being released, pointing at different advantages with the respective one.

For all three models that I have experimented with, the number of retrieved documents is fixed. A solution that has been proposed by some scientists is to have an adaptive number of retrieved documents. Depending on the implementation, an architecture like that could probably increase the overall retriever performance but to a lower computational cost than just retrieve a higher, fixed, number of documents.

5.2 Readers

When I compare the results of the evaluations on TeleQuAD respectively mTeleQuAD on all three models' reader parts it is clear that the readers that utilize a pre-trained language model achieve significantly higher scores. DrQA's reader, which is constructed from an LSTM, achieves an F1 score of 64.6 and an EM of 40.2 on the pre-trained setting. The comparable numbers for cdQA's reader, which is an implementation of BERT, are an F1 score of 77.8 and an EM of 54.5 when only pre-trained. This indicated that these pre-trained lganguage models really are powerful, which has been claimed by many sources. I find it also impressive that the reader module on MDR, an ELECTRA implementation, achieves even higher scores than cdQA, even though it has a multi-hop setting. The scores for MDR's pre-trained ELECTRA reader are 80.3 for F1 and 58.0 on EM; a small but significant improvement compared to cdQA.

This indicates that the pre-trained language model ELECTRA has some properties that make it better suitable for the machine reading comprehension task than BERT. Interesting continuations of my project would be to include ELECTRA in some single-hop openQA system like cdQA and to fine-tune it on the telecom domain. If we have in mind that BERT and ELECTRA were released in 2018 respectively 2020, I think that it is very likely that we soon will see even stronger pre-trained language models.

5.3 Single-hop and multi-hop models

In the full pipeline evaluations for the single-hop models, cdQA outperformed DrQA in all different settings. This was as I expected since cdQA includes a much more elaborate reader than DrQA. In the experiments with the pre-trained readers and the paragraphs setting for the retrievers, the F1 score was 13.3 absolute points higher for cdQA and the EM was 16.9 points higher. Almost the same difference had the distractor setting experiments for the pre-trained reader: 11.6 points difference for the F1 and 15.9 for the EM.

cdQA outperforms DrQA, even more, when the respective readers are fine-tuned. This gives a hint of how much and how fast a pre-trained language model can learn. With the paragraph setting in this context, the F1 for cdQA is 23.4 absolute points higher than for DrQA, and the EM differs with 23.8 absolute points. With the distractor retrieval, the F1 rose by 21.5 points and the EM with 21.9 points from DrQA to cdQA. I did expect cdQA to perform better than DrQA throughout, but I was a bit surprised at how much better its final results were.

The full pipeline evaluation for MDR I only made with a paragraph setting for the retriever and a pre-trained reader. The results for this evaluation are shown in Table 4.6. The presented scores for HotpotQA in the same table are produced under a different setting, and therefore it is hardly surprising that they are a lot higher than the ones for mTeleQuAD. A comparison that is perhaps more valid to make is to compare the F1 score and EM for the answers in mTeleQuAD with the scores from cdQA and DrQA that were received with the pre-trained, paragraph setting.

MDR obtained 49.8 in answer F1 and 30.0 in answer EM for mTeleQuAD. With TeleQuAD, cdQA got an F1 of 53.8 and an EM of 37.8 with the named settings. Corresponding numbers for DrQA are an F1 of 30.5 and an EM of 20.9. This means that the multi-hop model performed almost as well as cdQA, even though the former dealt with more complex questions. In my opinion, this is a good result of MDR's capability to answer complex questions. Also, it makes me believe that with some fine-tuning, MDR could perform very well on questions from the telecom domain.

5.4 Datasets

The creation of TeleQuAD and mTeleQuAD were important steps to evaluate my selected models in the telecom domain. Also, they enabled training neural parts of the systems on in-domain data, which adapts them further to the specific domain. TeleQuAD, with its approximately 2,000 question-answer pairs, are not large enough to train any module from scratch. However, I could show that using half of it to fine-tune a neural-based reader did increase its performance on telecom domain questions. This is applicable both for the LSTM based reader in DrQA and the BERT-based one in cdQA. This finding is in line with what earlier research has found, as accounted for in the section regarding previous work in domain adaptation. The fact that a robust, pre-trained system can be adjusted towards new domains with just a couple of thousand question-answer samples is very useful. Since building a dataset consumes a lot of resources, a lot of time and money can be saved if just development and test sets are needed and no training set.

In the multi-hop case, the dataset annotation took considerably more time per annotated question and answer. The main reason why mTeleQuAD only contains 50 question-answer pairs is that I did not have time to annotate more. But the findings from the evaluations on the single-hop models could reasonably be transferred to this setting. Meaning that a relatively small development set for fine-tuning can result in considerably higher performance for QA models.

One possible difficulty with both TeleQuAD and mTeleQuAD is the narrow domain that the questions and answers cover. The documents that the datasets were annotated from could possibly be redundant, meaning that information related to a specific question can be retrieved from several paragraphs, and not just only in the annotated one. In the case of SQuAD and HotpotQA, which both are annotated from Wikipedia, the source information is more structured, bringing some guarantee that the same information is not in several different articles.

This is not an issue when only evaluating machine reading comprehension, i.e. the performance of the reader modules when the context paragraph is provided along with a question. Instead, when investigating open-domain question answering, it can become troublesome to have similar information in many different documents. The retrieval criteria might not work as wanted in these cases. But on the other hand, relevant information might be stated in more places than initially planned. As in the case with the full pipeline evaluation for both DrQA and cdQA, the distractor documents might possibly include answers to the questions in TeleQuAD. In cases like these, it becomes less interesting to look at the performance of the retriever since it does not necessarily need to be the annotated paragraph that is retrieved in order to answer a question correctly.

The overall evaluation results for both TeleQuAD and mTeleQuAD show that they both include quite hard questions. To make a reasonable evaluation of how well a question answering system can handle queries from a specific domain, it is important that the questions are not too easy. If they are too easy, the evaluation would not yields any valuable information on how well the system can handle the in-domain questions. My conviction is that these two datasets fulfill their purpose of evaluating models' performance in the telecom domain.

5.5 Evaluation metrics

The evaluation metrics used, EM and F1 score, only focuses on the lexical matching and gives no credit for, for example, synonyms. Since many questions could have some kind of ambiguity, these metrics could possibly punish models that practically provide the correct answers. With the inclusion of language models in question answering systems, language understanding is improved. These openQA systems have shown to be able to handle more complex questions, and some of them are able to generate free-form answers. This motivates an introduction of some new metric, that can measure the similarity between predicted answers and the correct ones.

When an openQA system gets access to a large corpus of possible context to a given question, the answer can possibly be phrased in several places. However, it might not be formulated exactly as the annotated answer to the question. How the similarity between answers should be measured is however not easily decided. One possibility is to make use of the trained dense representations included in some models, for example, MDR. If a module like this is trained on relevant data it should be able to express similarities between expressions.

But the way of calculating the similarity between the dense vector representations and some threshold value for how similar answers are required probably needs to be manually decided.

5.6 Future work

The existing large pre-trained language models could probably be better adapted to the telecom domain than what I have achieved in this project. Instead of just fine-tuning the model with a small QA dataset, the language model could be trained from scratch with in-domain data. Using telecom data in this training would hopefully make the model better suited for language understanding within this area. From the experiments that I have made, and from the literature that I have studied, the ELECTRA model seems to be the better choice over some BERT model. A smaller, but probably also contributing improvement, would be to fine-tune a model with more samples. This requires a larger dataset than the one I have been using.

To extend TeleQuAD and mTeleQuAD would be a great way to further investigate how well models can be adapted to the telecom domain. A larger TeleQuAD would make better fine-tuning of readers possible. To annotate additional samples to mTeleQuAD could make it possible to fine-tune both the retriever and the reader module in MDR, since they both are trainable. That would be a very interesting experiment, to see how much the scores could be improved in multi-hop question answering.

Wiese et al. (2017) argues that creating datasets for a specific domain requires domain experts. That is probably valid in many cases, but for me, it has worked very well to annotate questions and answers without almost any telecom knowledge. With the data that I used to annotate, I was able to understand the overall meaning of a paragraph by just reading it. From this, I found it often possible to write a question regarding the context. Therefore I believe for anyone who has a developed reading comprehension to annotate more samples for the two telecom datasets.

The three models that I have evaluated in this thesis have been adjusted for either single-hop or multi-hop questions. In order to make openQA system more available for use cases, it would be great if they could handle several types of questions. There are a few recently published models that are more flexible in what input they can deal with, which would be interesting to look into if their code will be made public. I also believe that it would be possible to make adjustments to multi-hop question answering models such as MDR to make it able to answer single-hop questions too.

Chapter 6
Conclusion

This Master's thesis has focused on the adaptation of neural open-domain question answering systems to the telecommunication domain. By annotating a single-hop telecom question answering dataset, TeleQuAD, with around 2,000 question-answer pairs, I was able to fine-tune an openQA system. This fine-tuning resulted in an improved F1 score as well as exact match. This indicates that further enhancement of the performance might be possible without requiring more than a few thousand extra samples.

I also annotated a small multi-hop question answering dataset for the telecom domain, mTeleQuAD. I then applied a multi-hop openQA system with pre-trained language models as both retriever and reader. Even though the mTeleQuAD questions were complex, I was able to obtain an F1 score of 50 for the answers. This indicates that this corpus can serve as a reference point for evaluating question answering performance in the telecom domain.

mTeleQuAD's size is still limited and further annotation work to extend this corpus would certainly be beneficial. A larger corpus would enable a better fine-tuning of the retriever and reader and presumably increase the scores considerably.

References

Asai, A., Hashimoto, K., Hajishirzi, H., Socher, R., and Xiong, C. (2019). Learning to retrieve reasoning paths over wikipedia graph for question answering. *CoRR*, abs/1911.10470.

Chen, D., Fisch, A., Weston, J., and Bordes, A. (2017). Reading wikipedia to answer open-domain questions. *CoRR*, abs/1704.00051.

Clark, C. and Gardner, M. (2017). Simple and effective multi-paragraph reading comprehension. *CoRR*, abs/1710.10723.

Clark, K., Luong, M., Le, Q. V., and Manning, C. D. (2020). ELECTRA: pre-training text encoders as discriminators rather than generators. *CoRR*, abs/2003.10555.

Das, R., Dhuliawala, S., Zaheer, M., and McCallum, A. (2019). Multi-step retriever-reader interaction for scalable open-domain question answering. *CoRR*, abs/1905.05733.

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

Ding, M., Zhou, C., Chen, Q., Yang, H., and Tang, J. (2019). Cognitive graph for multi-hop reading comprehension at scale. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2694–2703, Florence, Italy. Association for Computational Linguistics.

Dunn, M., Sagun, L., Higgins, M., Güney, V. U., Cirik, V., and Cho, K. (2017). Searchqa: A new q&a dataset augmented with context from a search engine. *CoRR*, abs/1704.05179.

Farias, A. (2019a). Cdqa. `https://github.com/cdqa-suite/cdQA`.

Farias, A. (2019b). How to create your own question-answering system easily with python.

Feldman, Y. and El-Yaniv, R. (2019). Multi-hop paragraph retrieval for open-domain question answering. In *Proceedings of the 57th Annual Meeting of the Association for Computational Linguistics*, pages 2296–2309, Florence, Italy. Association for Computational Linguistics.

Green, B. F., Wolf, A. K., Chomsky, C., and Laughery, K. (1961). Baseball: An automatic question-answerer. In *Papers Presented at the May 9-11, 1961, Western Joint IRE-AIEE-ACM Computer Conference*, IRE-AIEE-ACM '61 (Western), page 219–224, New York, NY, USA. Association for Computing Machinery.

Hazen, T. J., Dhuliawala, S., and Boies, D. (2019). Towards domain adaptation from limited data for question answering using deep neural networks. *CoRR*, abs/1911.02655.

Hermann, K. M., Kociský, T., Grefenstette, E., Espeholt, L., Kay, W., Suleyman, M., and Blunsom, P. (2015). Teaching machines to read and comprehend. *CoRR*, abs/1506.03340.

Jiang, Y., Bordia, S., Zhong, Z., Dognin, C., Singh, M., and Bansal, M. (2020). Hover: A dataset for many-hop fact extraction and claim verification. *CoRR*, abs/2011.03088.

Karpukhin, V., Oguz, B., Min, S., Lewis, P., Wu, L., Edunov, S., Chen, D., and Yih, W.-t. (2020). Dense passage retrieval for open-domain question answering. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 6769–6781, Online. Association for Computational Linguistics.

Khattab, O., Potts, C., and Zaharia, M. (2021). Baleen: Robust multi-hop reasoning at scale via condensed retrieval. *CoRR*, abs/2101.00436.

Kratzwald, B. and Feuerriegel, S. (2018). Adaptive document retrieval for deep question answering. In *Proceedings of the 2018 Conference on Empirical Methods in Natural Language Processing*, pages 576–581, Brussels, Belgium. Association for Computational Linguistics.

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., 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.

Lee, K., Chang, M., and Toutanova, K. (2019). Latent retrieval for weakly supervised open domain question answering. *CoRR*, abs/1906.00300.

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

Min, S., Zhong, V., Zettlemoyer, L., and Hajishirzi, H. (2019). Multi-hop reading comprehension through question decomposition and rescoring. *CoRR*, abs/1906.02916.

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.

Perez, E., Lewis, P. S. H., Yih, W., Cho, K., and Kiela, D. (2020). Unsupervised question decomposition for question answering. *CoRR*, abs/2002.09758.

Qi, P., Lee, H., Sido, O. T., and Manning, C. D. (2020). Retrieve, rerank, read, then iterate: Answering open-domain questions of arbitrary complexity from text. *CoRR*, abs/2010.12527.

Qi, P., Lin, X., Mehr, L., Wang, Z., and Manning, C. D. (2019). Answering complex open-domain questions through iterative query generation. *CoRR*, abs/1910.07000.

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

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

Talmor, A. and Berant, J. (2018). The web as a knowledge-base for answering complex questions. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 641–651, New Orleans, Louisiana. Association for Computational Linguistics.

Thorne, J., Vlachos, A., Christodoulopoulos, C., and Mittal, A. (2018). FEVER: a large-scale dataset for fact extraction and VERification. In *Proceedings of the 2018 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long Papers)*, pages 809–819, New Orleans, Louisiana. Association for Computational Linguistics.

Trischler, A., Wang, T., Yuan, X., Harris, J., Sordoni, A., Bachman, P., and Suleman, K. (2016). Newsqa: A machine comprehension dataset. *CoRR*, abs/1611.09830.

Wang, S., Yu, M., Guo, X., Wang, Z., Klinger, T., Zhang, W., Chang, S., Tesauro, G., Zhou, B., and Jiang, J. (2017). R^3: Reinforced reader-ranker for open-domain question answering. *CoRR*, abs/1709.00023.

Welbl, J., Stenetorp, P., and Riedel, S. (2018). Constructing datasets for multi-hop reading comprehension across documents. *Transactions of the Association for Computational Linguistics*, 6:287–302.

Wiese, G., Weissenborn, D., and Neves, M. (2017). Neural domain adaptation for biomedical question answering. In *Proceedings of the 21st Conference on Computational Natural Language Learning (CoNLL 2017)*, pages 281–289, Vancouver, Canada. Association for Computational Linguistics.

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., Scao, T. L., Gugger, S., Drame, M., Lhoest, Q., and Rush, A. M. (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.

Xiong, W., Li, X. L., Iyer, S., Du, J., Lewis, P., Wang, W. Y., Mehdad, Y., Yih, W.-t., Riedel, S., Kiela, D., and Oğuz, B. (2021). Answering complex open-domain questions with multi-hop dense retrieval. *International Conference on Learning Representations*.

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

Zhang, Z., Yang, J., and Zhao, H. (2020). Retrospective reader for machine reading comprehension. *CoRR*, abs/2001.09694.

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.

Appendices

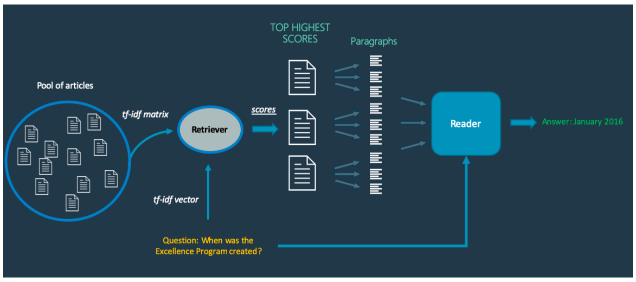
INSTITUTIONEN FÖR DATAVETENSKAP | LUNDS TEKNISKA HÖGSKOLA | PRESENTERAD 2021-06-09

EXAMENSARBETE Multi-hop Neural Question Answering for the Telecom Domain
STUDENT Maria Gunnarsson
HANDLEDARE Pierre Nugues (LTH), Fitsum Gaim Gebre (Ericsson)
EXAMINATOR Jacek Malec (LTH)

Neurala fråge-svarssystem tillämpade inom telekommunikation

POPULÄRVETENSKAPLIG SAMMANFATTNING **Maria Gunnarsson**

Tack vare storskaliga fråge-svarsdataset har neurala tekniker de senaste åren tillämpats i modeller för automatisk besvaring av frågor. Genom att ta fram dataset specifika för telekommunikation har vi i detta projekt finjusterat system för att bli specialiserade på just detta område.



När sökmotorer på internet används för att ställa en fråga, får man nuförtiden ibland upp ett färdigformulerat svar utan att behöva läsa igenom sökträffarna. Detta är exempel på användning av fråge-svarssystem, som med maskininlärning har lärt sig att lokalisera svar på frågor som formulerats i mänskligt språk. Utformningen av dataset som används i maskininlärning styr vad systemen blir bra på. I de flesta fallen av fråge-svarssystem så används dataset med allmän kunskap. Inom specialiserad verksamhet kan det dock vara mer användbart med ett system som kan besvara frågor inom det berörda området.

I mitt examensarbete har jag undersökt sätt att anpassa system för frågor och svar gällande telekommunikation. Tre olika publicerade fråge-svarssystem har använts; två som hanterar enklare frågor och ett som klarar av mer komplext ställda. För att finjustera dessa för det aktuella ämnesområdet har projektet innefattat att annotera två dataset; ett för de mer enkla frågorna och ett för de komplicerade. Vardera dataset innehöll frågor och tillhörande svar som rör telekom.

Först testade jag hur bra de tre systemen svarade på frågorna inom telekom och såg att deras resultat blev lägre jämfört med för frågor kring allmän kunskap. Sedan gjordes ytterligare undersökningar med de två systemen för enklare frågor. Halva datasetet med simpla frågor användes för att finjustera delarna som tränats med maskininlärning i dessa system. Detta i syfte att *lära* dem mer om telekom. Därefter användes den andra hälften för att igen avgöra hur väl systemen kunde hantera telekomfrågorna.

Resultaten visade att genom finjusteringen kunde en betydande förbättring av systemens kapacitet att svara rätt på telekomfrågor skapas. Andelen frågor som besvarades helt korrekt av det bättre av de två systemen ökade med över 10 procentenheter efter finjusteringen, vilket i sammanhanget kan anses vara väldigt bra. Dessa resultat tyder på att kraftfulla fråge-svarssystem kan anpassas för nya kunskapsområden med relativt små arbetsinsatser.