Multi-Stage Document Retrieval and Question Answering Pipeline with BERT
IR Project FSS 21 Report

Florian Rueffer, Marvin Roesel, Mert Ozlutiras
Information Retrieval Project Course (IE681)
Faculty of Business Informatics and Mathematics
University of Mannheim
frueffer@mail.uni-mannheim.de,
maroesel@mail.uni-mannheim.de,
mozlutir@mail.uni-mannheim.de

Abstract

BERT has proven itself as the state-of-the-art base model in natural language processing tasks. It already had a number of successful applications in document retrieval and question answering. This project is done with the aim of exploring BERT's capabilities in developing a holistic document retrieval and question answering pipeline. Our pipeline consists of four sub-models: A probabilistic retrieval model (BM25), a point-wise BERT based document ranking model, a pair-wise BERT based document ranking model and again a BERT based question answering model. Our pipeline is trained and tested on wikipedia dataset.

1 Introduction

In the recent years, NLP tasks ranging from machine translation to question answering are very commonly handled with neural models pre-trained on large corpora with language modeling tasks such as ELMo (Peters et al., 2018) and BERT (Devlin et al., 2019). Many of these models have achieved impressive results on various tasks lately. One of them, BERT, has recently been applied to a multi stage document ranking task (Nogueira et al., 2019). Their paper proposed a multi-stage ranking architecture to tackle the document ranking problem. They introduced two BERT based models called monoBERT and duoBERT. The monoBERT model is a pointwise document ranking model, creating a score for individual candidate documents, while the duoBERT model is a pairwise classification approach that considers pairs of candidate documents. To create an end-to-end document retrieval and question answering pipeline; we implement their architecture in this paper and stack it with a question answering model. The first three models are designed to work one after another (BM25, monoBERT and duoBERT), refining the document retrieval performance synergetically. The last refined output coming from duoBERT is the input to the question answering task to get the predicted answer for the question at hand.

In this paper, we are not using generic BERT as the base model for all models. For the document re-ranking models, we are using a recent variation called RoBERTa that has been shown to outperform generic BERT in many tasks lately. Since we don't have access to vast computational resources, we use a compact version of it, "distilroberta-base". However, given enough computational resources and time; the same pipeline can be replicated with a larger model such as "roberta-base" or "roberta-large". The same applies to the question-answering, where a pre-trained "Distilbert" model was used instead of the larger "Bert-Base" or "Bert-Large"

2 Dataset

TriviaQA dataset has two main parts: question & answers and documents. QA dataset includes 62.000 triplets, each triplet consists of a question, an answer, and the name of the document(s) the answer could be found. Document dataset contains 74.000 text documents scraped from wikipedia.

3 Pipeline

This paper proposes a Pipeline of 3 models stacked on top of each other. First we input BM25 with a question and all available documents. BM25 returns us the most promising 50 documents. These 50 documents are input to our pointwise ranker called monoBERT which returns top 5 candidate documents. Our last document retrieval model, duoBERT, takes these 5 documents and reranks them based on its pairwise scoring algorithm. The

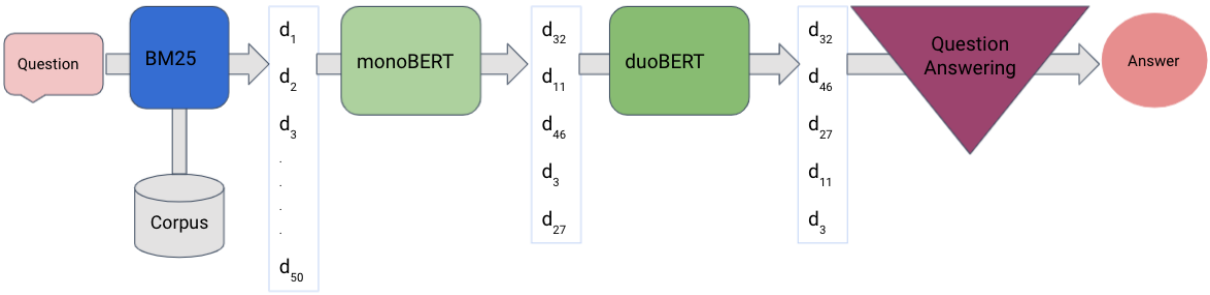

Figure 1: Pipeline

final re-ranked top 5 documents are input to our question answering model to retrieve the predicted answer to our query question in hand.

3.1 BM25

The first part of our chosen pipeline is described by the BM25 algorithm. BM25 is considered as a state of the art probabilistic retrieval model that ranks documents given a query based on a simple bag of words representation. The goal of the BM25 algorithm is to return a set of ranked documents which are potentially relevant to a query, or question in our case. As BM25 is a bag of words model, we decided to preprocess the queries and the document texts in order to decrease the number of tokens and increase the prediction quality. For preprocessing we used a lowercasing followed by a stopword removal and finally a stemming based on the Porter Stemmer, as all our documents and queries were in english. In our implementation, to represent the tokens efficiently, the BM25 algorithm uses a simple inverted index which contains a set of all tokens in all of our queries, mapping them to the according documents they appear in. However, this inverted index does not only list the documents each query token appears in, but also counts the frequency of appearences of each token in each document to compute term frequencies in a subsequent step.

After preprocessing all the texts and queries each tokenized query is passed through the BM25 method which computes the relevance scores based on the BM25 algorithm and returns the top 50 rated documents for each query in the test set and the top 20 rated documents for each query in the training set.

Even though BM25 is often used as base ranking model for complex tasked there have already been some improvements proposed to this method, as (Trotman et al., 2014) described. They stated that "BM25+ outperforms BM25 in all cases"(Trotman et al., 2014), which is the reason we used BM25+ which provides a rather simple improvement. In the evaluation section the results of BM25, BM25+ and BM11 are evaluated and compared to give evidence of the improved performance of BM25+.

3.2 monoBERT

The goal of re-ranking at this stage is to pointwise predict how relevant a document for the query at hand, query q. In essence, this goal is not different than BM25's goal. However, recent language models a much more deeper an complex neural models which are based on transformers, which are explained and introduced by Vaswani et al. (2017). Transformers are models that combine traditional feed-forward neural networks with self-attention layers. These self-attention layers allow the models to learn semantic relationships between words, which is why they are also often used for sentence completion. BERT models, however, have multiple self-attention and feed-forward layers but do not have decoder that traditional models have. That allows for creating embeddings of input texts without decoding them. We chose monoBERT as the first reranking step and expected the performance to improve over the BM25 results. We can see their working method as BM25 is doing the bulk elimination of documents and monoBERT is refining the top 50 documents and eliminate 45 of them.

monoBERT is a BERT model, described by Nogueira et al. (2019) that takes the query q and document d_i as input. The input is given in a tokenized form - a list of tokens that start with a [CLS]-token. A [SEP] token is then used to seperate the query from the document. However, the model can take a maximum of 512 tokens at once, but we apply no truncation to query. Since queries

in our dataset were fairly short, we give it maximum number of tokens. The rest of the tokens we can use in BERT was allocated to the document. The document was truncated accordingly to meet the maximum number of tokens.

To train the model we defined a training set which was based on the BM25+ outputs as we wanted the model to have some tough decisions about if a document is relevant for a query or not. For that reason we created a training set as follows: For each query we added all the relevant documents to the training set and for each relevant document this query has, we added one irrelevant document from the top of the BM25+ predictions for this query to the dataset. Instead of picking random non-relevant documents, this made the learning task significantly harder and led to a good monoBERT model.

Finally, we used a a pre-trained distil-RoBERTa based model for the backbone of monoBERT, as this model is significantly smaller than the traditional BERT and provides some improvements. Subsequently, it was finetuned on the downstream task of identifying relevant and non-relevant documents for the query q. We finetuned it on our annotated dataset, describe above.

We trained the model with the goal of minimizing the binary cross-entropy loss.

3.3 duoBERT

The output we had from the previous model is used as input to the pairwise re-ranker called duoBERT. Within the framework of "learning to rank", duoBERT can be characterized as a "pairwise" approach, while monoBERT can be characterized as a "pointwise" approach (Liu, 2009). In this pairwise approach, the re-ranker estimates the probability $p_{i,j}$ of the candidate d_i being more relevant than d_j (Nogueira et al., 2019) This second re-ranker model is also BERT based and it takes as input the question, candidate d_i and candidate d_j. Again similar to what we did in monoBERT, we don't truncate the question, we allocate the number of tokens required. The rest of the tokens (512 - tokens used for question) is shared between candidate documents d_i and d_j equally. The candidate documents are truncated accordingly. The entire sequence have exactly 512 tokens when concatenated with the [CLS] token and the three separator tokens. In our dataset, all the documents were long, therefore truncated.

We use the last hidden output of the BERT model as input to a two layer neural network to obtain the probability $p_{i,j}$. We also use a dropout layer with 0.2 probability to have a better training. We train the model with binary cross-entropy loss.

Since there are 5 candidate documents at inference time, we calculate we the pairwise score for every combination. In total, there are 20 pairwise scores calculated. We aggregate these scores into one single score S_i per candidate document. Even though there could be different methods to use for this aggregation, we used a simple sum method. At the end, documents were ranked based on their aggregation scores S_i.

3.4 Question Answering

For question-answering, a pre-trained BERT model was used and further trained on the given data set. Among three main pre-trained BERT models available ("Bert-Large" with 24 layers, "Bert-Base" with 12 layers and "Distilbert" with 6 layers) the "Distilbert" model was chosen due to computational restrictions of Google Colab. It can be assumed that, with enough resources, the same code used for training the "Distilbert" model can be run on either of the two other models. Since both the "Bert-Large" and "Bert-Base" can handle more complex structures, they will, when given enough time and data to properly train, likely outperform the "Distilbert" model.

In our wikipedia dataset there are roughly 62.000 triples given. Each triple consists of a question, an answer to that question and one or multiple filenames of .txt files, which reference a Wikipedia article inside the "wikipedia.zip" archive. However, not every triple can be used for training the "Distilbert" model, as either the given .txt file was not found in the .zip archive or the correct answer was not found in the given .txt file. In the original data set, aliases can be found for each answer, but for this approach a stricter restriction was set. Because of this, the text files that did not include the exact answer (but rather an alias) were excluded from both the training and the test set. This resulted in about 20.000 entries that had to be removed in order to assure that for the remaining data the answer to the question can be found somewhere in the given text file. The remaining 42.000 entries should be sufficient to see an increase of the question-answering performance of the model after training.

Contrary to the previous models, the maximum length of 512 tokens is problematic for answering the questions. Some tokenized Wikipedia articles (paragraphs) consist of more than 20.000 tokens, which results in more than 39 buckets / containers with 512 tokens each. Using all these 39 containers to train the model could at worst result in 38 cases where the answer is impossible to find in these 512 tokens and only one case where the model can theoretically find the answer. In order to still be able to train the "Distilbert" model in these cases and achieve a positive outcome, the bucket with the answer somewhere in these 512 tokens has to be extracted and then given to the model to train. In order to find the right bucket for each question, the question and the text are tokenized in three different ways for each question:

1. "train_encodings" contains one single container with at most 512 tokens. This variable will be used to store the correct pair of question and text with the answer. "train_encodings" will later be used to train the model as the answer is within these 512 tokens for each question.

2. The "train_encodings_non_truncated" variable is used to generate one container for each question, regardless of the amount of tokens. In the example above this variable would have more than 20.000 tokens for one question and could not be used in any Bert model. This variable is used to get the position of the complete answer inside all these tokens which is then used in the following variable.

3. In "train_encodings_overflow" the maximum number of tokens inside each bucket is limited to 512 tokens. However, the amount of buckets is not limited (contrary to train_encodings). With the information where the answer is in the total text, it's relative position (i.e. in which bucket is the answer) can be calculated. Each bucket consists of the following parts: One "[CLS]" token, the tokenized question, one "[SEP]" token, the tokenized text until 511 tokens are reached and one final "[SEP]" token. This leaves space for $512 - 3 - (NumberOfTokensForQuestion)$ tokens to fit the actual text. The position of the answer is then divided by the amount of text tokens that fit in each bucket to find the container in which the answer is located. This container is then stored in the "train_encodings" to train the model with.

It may happen that the answer starts in one bucket but ends in another bucket when the 512 token limit is reached in between. In that case only a part of the answer is contained when training the model with that specific token container. This could be changed by actively switching the order of the tokens inside these cases, however, this scenario may also happen when using the model later on when the correct answer is not present and the buckets cannot be reorganized to fit perfectly. Therefore, these cases are explicitly not changed when training the model. For the same reasoning, one can argue that choosing the bucket which definitely contains the right answer does not represent a real world scenario and should not be changed as well. For this case a solution is presented later in this paper on how to answer questions when more than 512 tokens are needed to tokenize the question plus the associated text.

As an alternative to the described approach on how to select the correct answer tokens, it would have also been possible to loop over each bucket and search for the answer inside. This would have likely been more computationally expensive, but the variable "train_encodings_non_truncated" would not have been needed and therefore some memory could have been saved during runtime. Because the memory usage was no problem at this stage, the approach described first was used.

In order to use backpropagation to improve the model step by step, the beginning and the end token of the answer are added to "train_encodings". With the start and end positions the loss can be computed while training the model. For training the model the Adam optimizer was used. The training and test data set was split with a 95 to 5 ratio. This leads to roughly 40.000 entries to train the model. It takes on average 30 minutes to train 5.000 questions and Google Colab allows the user to use a TPU for about four hours per day. With that it was most of the time possible to train the model for one epoch each day.

After training the model, it can be used to answer questions given a text that contains the answer. Similar to the training part, a solution has to be found for texts with a length of more than 512 tokens. When answering a question, the Bert

model assigns each token a probability that this token is the start of the answer and that the same token is the end of the answer in another variable. In order to work with texts that possess more than 512 tokens in their tokenized version, the whole text is tokenized and then split up into buckets with 512 tokens each. Each container then contains, similar to the training part, 1 "[CLS"] token, 2"[SEP]" tokens, the tokenized question and the corresponding tokenized text. The answer and the end probabilities are then calculated for each bucket that belong to one specific question. These scores are then stored in an ongoing array, which results in one large array with all the scores for all tokens that correspond to one specific question. From this the maximum values are extracted one for the start score and once for the end score. These positions are then traced back to the correct bucket and the correct position inside the bucket to retrieve the predicted answer from the model. Due to performance reasons, the number of buckets that shall be calculated, can be restricted with the parameter "numOverflow", which is set to three by default. Similar to the training part, this approach does not work when the answer is split between two buckets. However, most of the time the answer consists of only one, at most three to four tokens, which makes it less probable that the answer is split between two buckets. Especially in the Wikipedia articles that contain a lot of words, it is more probable that the answer is not between the first 512 tokens and would therefore be impossible to find for the model. While testing with one and multiple buckets, the approach that used multiple buckets consistently performed better, which is why this method will be used going further.

The approach above described how to handle large text data files. However, it may be possible that multiple files exist that are likely to contain the answer to the question. In a single file given from the previous models it may happen that the answer is not present in the text or that the answer is not correctly found. Therefore, having multiple files to search through can be an advantage. This approach is implemented rather similar to handling large text data, but instead of comparing all tokens, only the maximum values for each file are compared. The answer from the file with the highest score is then chosen as the final answer. Using multiple files per question, they achieved on average a better score than using one file per question.

4 Evaluation

In this section, we provide insights about the performance of our chosen models. For the document retrieval pipeline we mainly examined the probabilistic retrieval and monoBERT as we observed the largest differences between different models here. As we ranked all the documents we use P@K, AP@K, R-Precision and Average R-Precision as evaluation measures. We evaluated BM25 on the whole test set and monoBERT on a subsample due to capacity restrictions. We used mean values to average the scores over multiple queries.

4.1 BM25

As described above, we used different models to showcase the improvements BM25+ has compared to other probabilistic models. The respective Mean precision scores are represented in the table below:

	BM11		BM25		BM25+	
k	MPK	MAPK	MPK	MAPK	MPK	MAPK
1	0.47	0.47	0.49	0.49	0.52	0.52
2	0.34	0.54	0.36	0.56	0.38	0.59
3	0.28	0.56	0.28	0.58	0.30	0.60
4	0.23	0.56	0.23	0.58	0.25	0.61
5	0.20	0.57	0.20	0.58	0.21	0.61
6	0.17	0.56	0.17	0.58	0.18	0.61
7	0.15	0.56	0.16	0.58	0.16	0.61

Table 1: Evaluating BM11/25/25+

	BM11	BM25	BM25+
Mean R-Precision	0.40	0.42	0.44
Mean Average R-Precision	0.51	0.53	0.56

Table 2: Precision BM11/25/25+

As we can clearly see, the model performance improves from BM11 to BM25+. BM25+ outperforms BM11 and BM25 considering the mean precision and mean average precision at k-values between one and seven. Respectively, it outperforms the other models in mean R-Precision and mean average R-Precision. With a mean r-precision of 0.44 and a mean average r-precision of 0.56 the model already delivers some good results.

4.2 monoBERT

After the BM25+ model the monoBERT reranking was mostly able to refine and improve the BM25+

ranking. As the inference with monoBERT needed a large amount of GPU and RAM capacities we sampled a subset of the testset and evaluated BM25+ against monoBERT on that sub-sample.

	BM25+		monoBERT	
k	MPK	MAPK	MPK	MAPK
1	0.52	0.52	0.57	0.57
2	0.38	0.59	0.49	0.67
3	0.30	0.60	0.37	0.67
4	0.25	0.61	0.32	0.67
5	0.21	0.61	0.27	0.67
6	0.18	0.61	0.23	0.67
7	0.16	0.61	0.20	0.66

Table 3: Evaluation BM25+ / monoBERT

	BM25+	monoBERT
Mean R-Precision	0.44	0.46
Mean Average R-Precision	0.56	0.61

Table 4: Precision BM25+ / monoBERT

monoBERT was able to improve the document retrieval considering the mean precision matrices we used. The mean precision at 2 increased by 0.11, which means, that instead of ranking about returning a fraction of 38% of relevant documents considering the top two ranked documents, monoBERT is able to retrieve nearly 50% relevant documents among the top 2 ratings. The mean average precision score also increased by 0.08 which leads to a score of 0.67 for k-values between 2 and 6. In mean r-precision and mean average r-precision the model perfomance, compared to BM25+ increased by 0.02 and 0.05 respectively. It is important to note here, that, in order to evaluate monoBERT, we returned more than 5 documents from monoBERT to calculate precision scores for higher k-values and r-precision for many relevant documents.

4.3 duoBERT

duoBERT, however, was mostly not able to further improve the monoBERT reranking. This could possibly be an issue on how we created the dataset or simply an issue with BERT traditionally not created for such tasks, as it does not know how to interpret additional [SEP]-tokens between the documents. We also had some ideas to improve the duoBERT, but were not able to make the model converge very good, such that the performance did not increase, which is why we did not mention these solutions in our pipeline above.

4.4 Question Answering

In order to be able to evaluate a given answer from the model, some regulations have been set on when to classify an answer as correct and when not to. Since an uncased Bert model is used, the answers are given as lowercase letters. Therefore the correct answers has to be lowercased as well. An answer is seen as correct when the given answer corresponds completely to the correct answer. Since the aliases to an answer are not used as described earlier, some conditions are set on when to count an answer as partially correct. A partially correct answer is given when either:

1. The given answer can be found inside the correct answer. For example: the correct answer to the question ”In what movie does Anakin become Darth Vader” is ”star wars: revenge of the sith (episode iii)”, but the models predicts just ”revenge of the sith (episode iii)” as answer. As the answer is still correct when viewing it from a human perspective, this is counted as a partially correct answer

2. The reversed case: The correct answer can be found in the given answer Example: The correct answer to the question “Which golfer Jack was nicknamed ‘The Golden Bear’?” is ”jack nicklaus”. The model however predicts his full name ”jack william nicklaus”. As the answer is still correct when viewed when a human perspective, this is counted as a partially correct answer.

Rarely the model outputs the whole text as an answer to a given question. As the answer is most likely somewhere in the answer predicted by the model, this would count as a partially correct answer by the above described method. If this would be true, then the model could just output the complete text all the time and would at least achieve mostly partially correct answers. To prevent this an additional constraint was set: If the number of characters in the answer extends 50, then the answer is automatically classified as wrong. Most of the time the answer consist of one, at most three words. For reference: The answer ”Star Wars: Revenge of the Sith (Episode III)” belongs to the longest answers in the whole data set and consists of 44 letters. Therefore it can still be answered

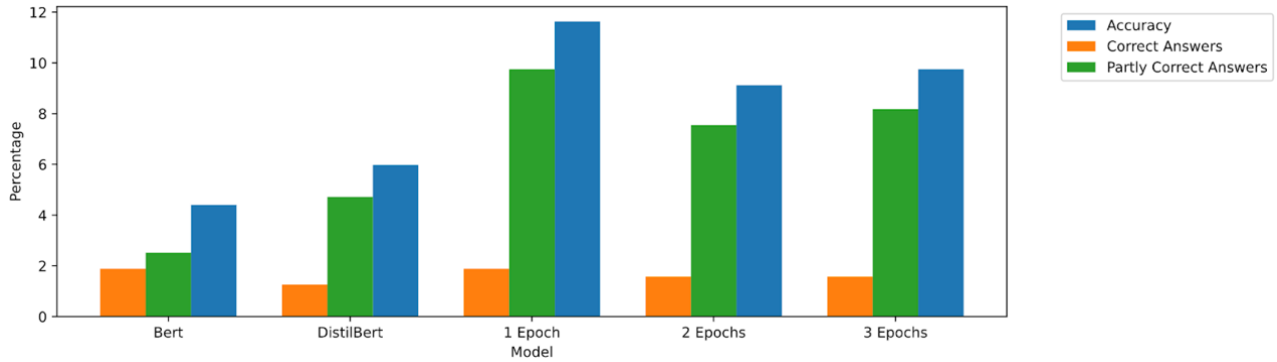

Figure 2: QA Performance Validation Set

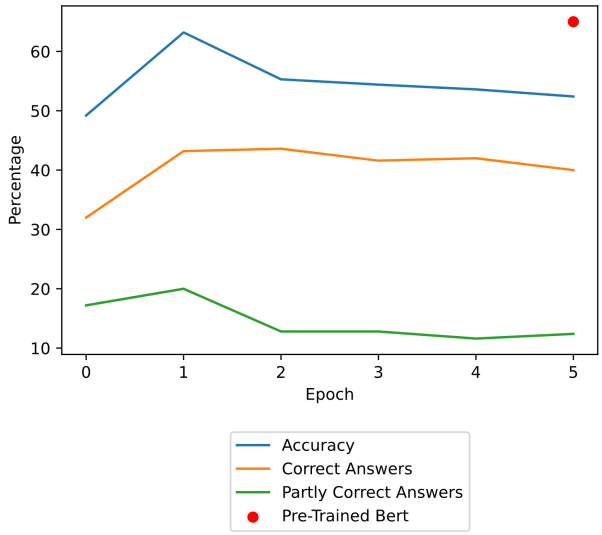

Figure 3: QA Performance per Epoch in Test Set

correctly by the model and be recognized as a correctly given answer. If none of the above mentioned cases fits for a given answer, this prediction is classified as a wrong answer. The model was trained for 10 epochs with about 40.000 training data in each epoch. In order to evaluate each epoch, the accuracy was calculated. For the accuracy, both the correct and the partially correct answers were added up and divided by the total number of questions. The data set used to calculate the accuracy consist of 250 images, that were split from the training beforehand. Each epoch was tested on the same data set to ensure comparability. The results of the first five epochs can be seen in figure 3. While the pre-trained model achieves an accuracy of about 49%, this value grows to 63% after the first epoch. However, for every following epoch the accuracy starts decreasing again, as the model starts to overfit on the training data. Each of these epochs still performs slightly better than the original pre-trained model, however the first epoch shows a significant increase in performance. It is also notable that the percentage of correct answers given in each epoch stays roughly the same and above the accuracy of the baseline model. Only the number of partly correct answers decreases with each epoch after the first one, performing even worse than the pre-trained model. A reason to this might be that the model is very confident in giving answers due to the overfitting and therefore returns mostly answers that are either completely correct or wrong. The pre-trained "Bert-Large" model achieves an accuracy on the same data set of roughly 65%. This almost matches with the accuracy of the best epoch, however, the "Distilbert" model answers question almost 40% faster than the pre-trained "Bert-Large" model. When the time to compute an answer is an important factor, this trained model achieves comparable results in a smaller time period. Therefore, the best approach in this scenario is to use the model that was trained only for one epoch.

A similar behaviour can be seen in figure 2 when comparing the models on the validation set. However, in this validation set all models perform significantly worse than in the test set. A possible explanation to this might be that the questions are more complex and therefore harder for each of the models to answer. In this approach the model that was trained for only one epoch performed the best out of all "Distilbert" models. Contrary to the test set, this trained model also performed significantly better than the pre-trained "Bert-Large" model, especially when it comes to the party correct answers. Still all the models performed rather bad on this validation set. In order to achieve better results on this set, more questions like these are needed in the training set. When training the

model on significantly harder questions, the results in the validation set will very likely also increase.

5 Conclusion

Several methods were used to retrieve a small subset of documents, which were given to the question-answering model to predict an answer. For the document retrieval, BM25+ performed better than standard BM25. Stacking monoBERT on top of BM25+ resulted in a higher accuracy compared to BM25 or BM25+ used alone. Adding duoBERT on top of these two well performing models did not increase the performance. Even though several attempts like adding a CNN or increasing the number of hidden layers were explored to improve the model, duoBERT didn't yield a remarkable improvement. Overall, our document retrieval pipeline achieved a performance that is comparable to other state-of-the-art solutions. Another remark from our study is that using "Distil" versions of the models not only led to faster training and inference, but also enabled us to train our models on free GoogleColab GPUs. Using "Distilbert" for the question answering reduced the time required to answer a question by approximately 40% compared to using "BERT-large". Given enough computational resources, in case the number of the correct answers is more important than the execution time; we would still recommend training a "Bert-Large" instead of the "Distilbert". Even though we could not test how much of an improvement can be achieved when training on "Bert-Large" due to the restricted resources given by Google Colab, we would expect some improvement over "Distilbert" since it's a deeper model. However, the predicted performance gain should be somewhat similar to the increase of accuracy that could be achieved when using "Distilbert".

The goal of the project, to build a pipeline that can answer a question given a wikipedia archive, was therefore fulfilled and all of the models are fitted to their respective task.

References

Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of deep bidirectional transformers for language understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, Volume 1 (Long and Short Papers)*, pages 4171–4186, Minneapolis, Minnesota. Association for Computational Linguistics.

Tie-Yan Liu. 2009. Learning to rank for information retrieval. *Foundations and Trends® in Information Retrieval*, 3(3):225–331.

Rodrigo Nogueira, Wei Yang, Kyunghyun Cho, and Jimmy Lin. 2019. Multi-stage document ranking with bert.

Matthew E. Peters, Mark Neumann, Mohit Iyyer, Matt Gardner, Christopher Clark, Kenton Lee, and Luke Zettlemoyer. 2018. Deep contextualized word representations.

Andrew Trotman, Antti Puurula, and Blake Burgees. 2014. Improvements to bm25 and language models examined.

Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N. Gomez, Lukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need.