Neural Re-ranking Retrieval System

Abdullah Alarfaj

School of Computing Science

Sir Alwyn Williams Building

University of Glasgow

G12 8RZ

A dissertation presented in part fulfillment of the requirements of the Degree of Master of Science at the University of Glasgow

2nd September 2022

Abstract

Information retrieval (IR) systems have benefited from the recent advancements in neural networks, which can capture the context of words and have a better semantic understanding. One of the shortcomings of neural networks in IR is the cost in terms of efficiency and resources. This can be addressed by using the neural model as a re-ranker which can increase the overall effectiveness of the system without sacrificing efficiency.

In this project, we aim to discover the effect of re-ranking a candidate list of documents using a neural network model. We will also investigate models with different document representations. We will pair these representations with relevance feedback function to improve the results. In other words, we will develop a pipeline of technologies that includes a classical retrieval model, a neural re-ranker, and a pseudo relevance feedback mechanism.

This pipeline has proven to be effective as it outperformed the use of a single classical ranker, the addition of relevance feedback got us significantly better results.

Education Use Consent

I hereby give my permission for this project to be shown to other University of Glasgow students and to be distributed in an electronic form.

Name: Abdullah Alarfaj Signature: Abdullah Alarfaj

Acknowledgements

I would like to express my gratitude to professor Joemon Jose for all support and guidance throughout the dissertation. I also would like to thank my wife and my family for their support and encouragement.

Contents

Chapter 1 Introduction

1.1 Overview

Neural information retrieval systems have achieved great results in the recent years, these systems can be used to rank or re-rank documents, while results are impressive, using these systems for ranking comes at a high cost in terms of efficiency. The other approach is to use these systems to re-rank a list of candidate documents retrieved by an efficient classical ranker; this approach may outperform a classical ranker without sacrificing the efficiency. In this project, we aim to implement an information retrieval system that uses a classical ranker as an initial ranker, candidate documents will be fed to a neural model for re-ranking, and then we will apply a pseudo relevance feedback function [11] for query expansion. We will use MS MARCO document ranking dataset [17] to finetune the neural models and evaluate the system. We found that re-ranking improves the results, and the addition of pseudo relevance feedback led to better results.

1.2 Motivations and Aims

The first step of finding information that users need is to use search engines to look it up, and with the rapid growth of the internet, information retrieval (IR) systems are becoming more important, evermore, retrieving the best available information in a timely manner and presenting it to the user means they can be more productive, and they can make better decisions. Classical ranking models use terms statistics of the query and the document to obtain relevance score, and these models are highly efficient, they can retrieve relevant documents from millions of documents in a fraction of a second. We want to improve the effectiveness by retrieving more relevant documents, especially in the top ranks. Learning to Rank (LTR) [12] models were proposed to address this issue, these models can achieve better results by leveraging hand-crafted features, which include query dependent features such as classical ranking models, query independent features such as PageRank [33], or query features such as presence of entities. Another approach is to use neural networks to improve the effectiveness, unlike LTR, neural models can achieve results by leveraging their capabilities to learn language structure and build representations of text. Transformer-based contextualized word embeddings [32] are the state of the art in natural language processing. These models can generate embeddings that can capture semantic properties of the text, which may lead to better matching between query and document. Although these models can outperform classical models [2], they require higher resources and take longer time to retrieve documents, the main motivation is to combine the advantages of classical models and neural models. This can be achieved in the form of a pipeline that uses the classical model as a first step ranker and uses a neural model to re-rank the candidate list of documents retrieved in the first step, this method takes the higher effectiveness of the neural models and combines it with the higher efficiency of the classical models. This pipeline can be expanded to include a relevance feedback component, which can help reduce the gap between query intention and query representation by expanding the query with terms that hopefully lead to more relevant documents. We aim to construct and evaluate

pipelines that outperform the classical models, in order to do this, we need to have a suitable classical model to retrieve the candidate document list, a suitable neural model to re-rank the candidate document list, and a pseudo relevance feedback method. We also need to address the workflow of the pipeline to ensure it maximizes effectiveness and efficiency. Furthermore, in this project, we aim to answer the following research questions:

- Does using the pipeline approach improve effectiveness when compared to classical ranking model?
- Does the pseudo relevance feedback improve the effectiveness?
- Is just re-ranking using the new expanded query better than retrieving a new candidate list?
- What effect does the number of tokens have in Longformer?
- Does using full document representation instead of just the first passage improve the results?

We will explain and discuss these questions further in chapter 6.

1.3 Methodology

In order to achieve the aims of this project, we will use the following components:

- BM25 [16] to retrieve the initial candidate list of documents, it will be used as the classical ranking component.
- BERT [4] and Longformer [13] representations, in this project we will develop two different pipelines: one uses BERT as a neural re-ranker; and the other uses Longformer. This will allow us to compare two different document representations, the first model creates the representation using the first passage (BERT), and the second model creates the representation using the full document since it can handle long documents (Longformer), we will discuss this further in the next chapter.
- Bo1 from divergence from randomness (DFR) framework [19] as the pseudo relevance feedback function.

These components will be used in a pipeline approach to build the proposed systems. In order to finetune the neural models and evaluate the systems, we will use MS MARCO document dataset [17].

1.4 Outline

The outline of the report is as follows, in chapter 2, will discuss some of the relevant information retrieval research papers, this includes classical models, neural models, and relevance feedback methods. In chapter 3 we will discuss functional and non-functional requirements of the project. In chapter 4, we will discuss the components of the system and their design. In chapter 5, we will discuss the implementation of each component and technologies used in the development. In chapter 6, we will discuss experimental setup and evaluation of the system. In chapter 7, we will summarize the work accomplished and discuss future work.

Chapter 2 Background

2.1 Neural information retrieval

Over the years, neural networks have proven to be an effective alternative to traditional approaches [2] in many fields including information retrieval and natural language processing. Neural information retrieval is the use of neural networks in information retrieval systems. One of the breakthroughs is the use of word embeddings [8][31] to represent text, it allows models to capture words relations which helps with inexact matching, for example searching for documents that contain Scotland may show documents that contain Glasgow because it is a city in Scotland. In general, neural networks follow the architecture in Figure 1, where input can be word embeddings vector, representation can be produced by a neural network like convolutional neural network (CNN) [27], or recurrent neural network (RNN) [27], interaction layer is used to compare the representations, then it outputs matching signals, aggregation layer is used to aggregate the matching signals produced by interaction layer, and finally a fully connected neural network is used to output a matching score.

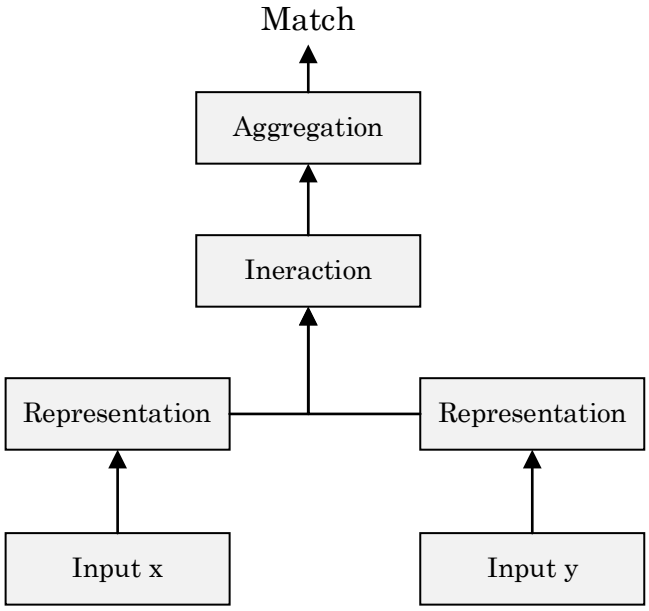

Figure 1 Neural IR

2.2 Transformer

Recurrent neural network (RNN) [27] is a neural network that can handle sequence to sequence tasks like machine translation, text summarization, or voice recognition using encoder-decoder architecture. In the case of machine translation, the encoder takes a sequence of words (the text we want to translate) and creates a context vector, this context vector is then sent to the decoder to generate the output sequence (the translated text). This is possible by using hidden states, the hidden state is the output of the current RNN cell (the word being processed) that we feed to the next RNN cell (Along with next word in the sequence), that is how RNNs capture the context of the text we want to translate. RNNs handle short

sequences with no problems, but when we try to handle longer sequences, the sensitivity decays over time and the network can no longer remember the first inputs, this is known as vanishing gradient problem. Attention mechanism was one of the solutions proposed to deal with this issue [28], it allows the network to focus on parts of the input sequence that are relevant to the current time step. Transformer [3] is a sequence to sequence model that takes the attention mechanism further, it uses "self-attention" to capture the relevant parts of the sequence to the word currently processed, and to make sure it has a better understanding of the words relations, Transformer uses "multi-head attention", in simple terms, it uses multiple "self-attention" heads in every encoder/decoder layer, hence it has multiple representations of word relations combined for a better overall understanding. It uses positional encoding to determine the word position in the sequence, which means we do not need to process the words in order like what we used to do with RNNs, hence we can go parallel to boost the process. Transform enhanced the way we model languages, and Transformer-based models are making breakthroughs in many NLP problems. Many Transformer-based models come pre-trained on large corpora. Pre-training is a form of transfer learning. In natural language processing (NLP), pre-training aims to make the model understand how languages are modeled, this knowledge can be leveraged by finetuning a pretrained model to perform a specific task. This method saves a lot of time and resources, for instance, BERT pretraining took four days and 64 TPU chips, Figure 2 shows Transformer architecture.

2.3 BERT

BERT [4] stands for **B**idirectional **E**ncoder **R**epresentations from **T**ransformers; it is a Transformer-based model that uses the encoder part of Transformer to model languages. Unlike context-free models like Word2vec [8], BERT provides contextualized word embeddings, which means a word can have different embeddings depending on the number of contexts this word can be used in. On the other hand, Word2vec has only a single embedding for every word. BERT was pre-trained on two tasks, the first is masked language modeling task (MLM). Unlike language modelling (predicting the next word in the sentence), BERT predicts the masked word in the sentence, so the objective is to predict the masked word and reconstruct the original sentence which makes it bidirectional (looks at previous words and next words). The second task is next sentence prediction (NSP), given a sentence "a", is sentence "b" coming next? BERT can be finetuned to many downstream tasks with just one additional output layer to suit the need of that task.

2.3.1 BERT Architecture

In the base model, it uses twelve encoder layers, twelve self-attention heads in each layer, and 768 hidden units in the feed forward network (the original Transformer had six encoder layers, eight self-attention heads in each layer, and 512 hidden units in the feed forward network). The first token in every input that BERT takes is the [CLS] token (classification token), this token captures the representation of the whole sequence fed to BERT, a sequence can be a single sentence, or two sentences separated by the special [SEP] token.

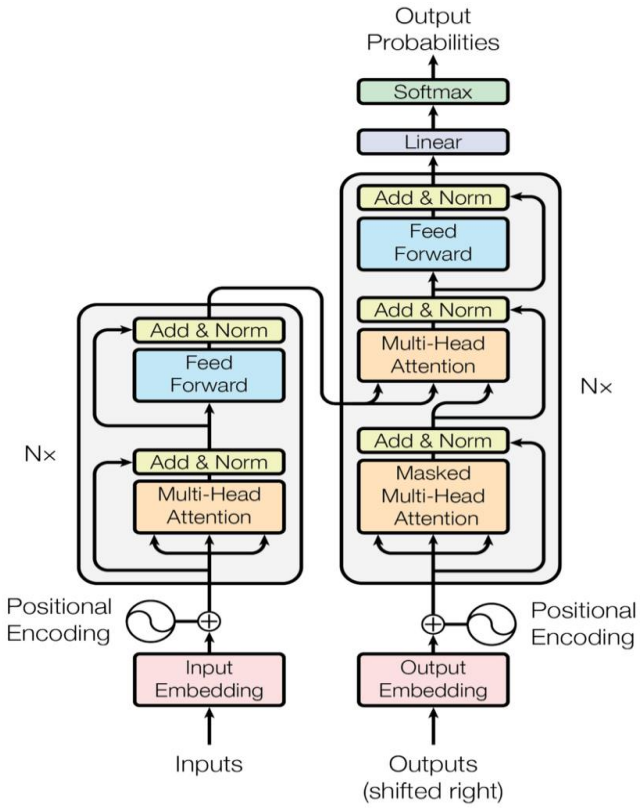

Figure 2 Transformer architecture [3]

2.3.2 BERT for sequence classification

BERT can be finetuned to many NLP tasks including sequence classification, in order to do that, a classification layer has to be added on top of BERT. The classifier leverages the [CLS] token to classify the sequence, for example, it can determine whether the sentence in the input sequence is spam or not, or whether the two sentences in the input sequence are similar or not. In information retrieval systems, we can concatenate the query and the document in one sequence separated by [SEP] token, and feed it to BERT, then we can take the [CLS] token from the output and feed it to the classification layer to determine if the query matches the document or not. We can use the probability of being a match to rank documents, where top ranked documents will have higher probability of being a match. Figure 3 shows the use of BERT for sequence classification.

2.3.3 BERT and long documents

BERT and many other Transformer-based models have an inherent maximum sequence length of 512 tokens, the reason behind this limit is that the cost of self-attention mechanism (which these models inherited from the original Transformer) grow quadratically with sequence length, which makes it infeasible to go beyond this limit. One of the simplest strategies to solve this issue is to take the first passage in the document (FirstP) [7], this method assumes the first passage to be representative of the whole document and scoring can be based on it. Other strategies include splitting the document into passages, one strategy is to score the document based on the best scoring passage (MaxP), the other strategy uses another Transformer to combine the [CLS] token representations of all passages [29].

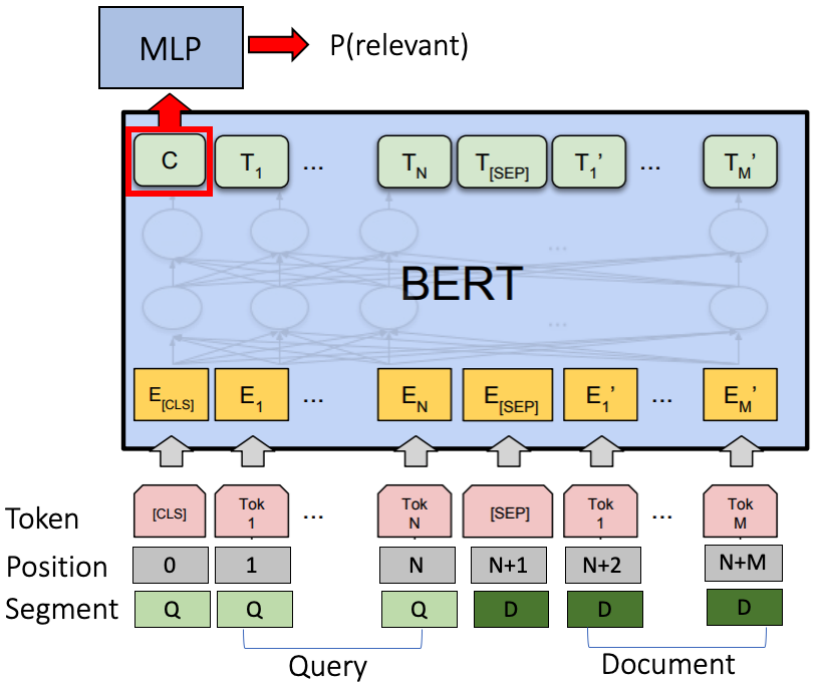

Figure 3 BERT Sequence Classification Illustration [7]

2.4 Longformer

Longformer (long document Transformer) [13] is a BERT-like Transformer-based model that can handle longer sequences. Longformer has a different self-attention mechanism that grows linearly with sequence length. It combines a local windowed self-attention and a global attention to build the representation of longer documents. Tokens with global attention attend to every token in the sequence, while tokens with local attention attend only to tokens that appear in their local window, attention type is selected based on the task. For example, query tokens and the <s> token (classification token) are assigned global attention, and the rest of the document tokens are assigned local attention. Longformer was pretrained on masked language modeling task (MLM).

2.5 BM25

BM25 [16][19][22] is a classical probabilistic information retrieval model based on binary independence retrieval model. BM25 uses statistical approaches to compute the relevance score, it has three main components:

- The first is inverse document frequency component, it computes the term weight based on its occurrences in the collection. For terms that appear very frequently in the collection, weight will be diminished since it is not helping distinguishing documents, and for terms that do not appear frequently, weight will be higher.
- The second is document term frequency component, k_1 parameter is used to smooth the term frequency, because term weight should not increase linearly with repeated occurrences. We also have to account for document length, a longer document may have more term occurrences, this will give longer documents an advantage over shorter documents, K equation is used for document length normalization, it penalizes documents longer than

average and boost documents shorter than average, b parameter is used for the document length normalization.

- The third component is query term frequency, if a term has multiple occurrences in the query, it should have a higher score, k_3 parameter is used for smoothing.

$$Score_{BM25}(Q,D) = \sum_{i=1}^{n} \log\left(\frac{N - n_i + 0.5}{n_i + 0.5}\right) \cdot \frac{(k_1+1)tf_i}{K + tf_i} \cdot \frac{(k_3+1)qtf_i}{k_3 + qtf_i} \qquad (1)$$

$$K = k_1((1-b) + b \cdot \frac{dl}{avdl}$$

Although BM25 does not have the semantic understanding that neural models have, it is very efficient and fast, which makes it favorable as initial ranker. Equation 1 shows BM25

2.6 Pseudo relevance feedback

Information need is the reason people submit queries to information retrieval system, but sometimes the information need might be vague or poorly represented. Relevance feedback can help guide the retrieval process by selecting documents that are relevant to the information need so the system can retrieve similar documents. Instead of asking users to select the relevant documents explicitly, we can do this automatically using pseudo relevance feedback. We assume that topmost retrieved documents are relevant, and we can select terms from these documents to expand the query. Terms from the topmost list are ordered by weight which can be computed using a weighting model, and terms with higher weights are added to the query. In this project, we adopted Bo1 from Divergence from Randomness (DFR) framework [18][19]. Bo1 measures the informativeness of a token using Bose-Einstein statistics. DFR assumes that term occurrences are random in the collection, if term frequency of a term in a document diverge from the collection random distribution, it means that the term is informative in this document. DFR models can measure the degree of divergence and based on that we can sort the tokens in the topmost list. Equation 2 shows Bo1.

$$inf_{Eq}(t) = -\log_2\left(\frac{1}{1+\frac{F_{Eq}}{N}}\right) - F_{Eq} \cdot \left(\frac{\frac{F_{Eq}}{N}}{1+\frac{F_{Eq}}{N}}\right) \qquad (2)$$

Where inf_{Eq} is the informativeness of a term t in the topmost list E_q, F_{Eq} is the number of occurrences of the term in E_q, and N is the number of documents in the collection.

Chapter 3 Requirements

In this chapter, we will discuss functional and non-functional requirements of the proposed system.

3.1 Functional Requirement

Functional requirements specify features the system will have. We will use MoSCoW [5] approach in this project as prioritization method to categorize requirements based on importance. MoSCoW is an acronym that stands for the priority groups that will be used to categorize requirements, M stands for "Must have", S stands for "Should have", and C stands for "Could have". Non-functional requirements will have "NF" label.

3.1.1 Must Have

These requirements must be achieved in this project:

- M.1: The system must be able to handle MS MARCO document dataset.
- M.2: The system must have a classical ranking model to generate the candidate documents to be re-ranked.
- M.3: The project must have two neural models with different document representation approaches to re-rank the initial candidate document list.
- M.4: The system must have a pseudo relevance feedback function to improve the ranking results.
- M.5: The project must have a baseline to compare the proposed system to in experiments.

3.1.2 Should Have

- S.1: The project should have multiple metrics implemented to measure the proposed system effectiveness.
- S.2: The system should accept queries as a string in case of a single query, or a list to run multiple queries at once.

3.1.3 Could Have

- C.1: The system could have an approach to that allows BERT to handle longer documents.
- C.2: We will try to incorporate a neural approach for pseudo relevance feedback to do query expansion instead of the traditional methods.

3.2 Non-functional Requirements

- NF.1: Code should be well-written and well-documented so other researchers and developers can reuse any part of it.
- NF.2: The system should be efficient and reliable.

Chapter 4 Design

In this chapter, we aim to present the system design, and explain the workflow of the system and the design choices.

4.1 Overall Architecture

Information retrieval systems have offline components that prepare parts that are essential to run the system, and online components that does the inference and retrieve information.

Offline components include

- Dataset preparation component: this component is used to extract parts of the dataset to be used to finetune the neural model, and it prepares the dataset for indexing.
- Dataset Indexing component: This component is used to create the inverted index for the classical initial ranker and pseudo relevance feedback.
- Finetuning component: This component is used to finetune the neural models to be used for re-ranking.

Online components include

- Classical ranking component: This component will generate the candidate list of documents to be re-ranked by the neural model
- Neural re-ranking component: This component will be used to re-rank the candidate list of documents generated by the classical ranking component.
- Pseudo relevance feedback component: This component will provide the PRF mechanism to expand the query.
- Retrieval component: This component serves as the pipeline that combines other online components.

4.2 System Diagram

The information retrieval system is divided into two parts, namely offline and online. The offline part takes care of dataset preparation, dataset indexing, and finetuning the neural models. The online part takes care of retrieving the candidate document list, re-ranking, and expanding the query using pseudo relevance feedback. Figure 4 shows the high-level system diagram.

4.3 The workflow of the System

As we can see in figure 4, information retrieval system has two parts with multiple components in each part, the workflow of the system will be explained in this section. Information retrieval system starts with the offline part, dataset preparation component loads the dataset files into memory to help initiate the other two offline components, namely the indexing component and the finetuning component. The indexing component will create the inverted index from the loaded collection file and store the index files to disk. The finetuning component requires the dataset preparation component to extract the training samples from the

dataset before starting the finetuning process. The online part starts with the retrieval component, it will initiate the retrieval by invoking classical ranking component which will retrieve the candidate document list, then it will invoke neural re-ranking component to re-rank the candidate document list, then it will invoke pseudo relevance feedback component to expand the query, the final step for the retrieval component is to use the expanded query to re-rank the document list to show the final results. Requirements M.1, M.2, M.3, M.4, and S.2 are achieved by this design.

4.4 Further Details of Some Components

4.4.1 Dataset Preparation Component

Dataset preparation component is designed to process MS MARCO document dataset. And it is responsible of providing the data in a suitable format to indexing component and finetuning component. Indexing component requires the full collection file to be loaded to memory. This file contains document id, URL, title, and the document text body. Once loaded, indexing component does not require any additional files to create the inverted index. On the other hand, the finetuning component requires training partition and validation partition in addition to the collection file described above. Each one of these partitions have the following files: the queries file which contains query id and query text of all the queries in the dataset, and the qrels file (query relevance judgment) which contains query id, document id, and relevance judgment. These files will be processed further to suit the need of the neural model to start finetuning. Preparation for finetuning will be explained further in chapter 5. Pseudo relevance feedback component requires an inverted index that includes the first 500 tokens of each document instead of full documents, the processed collection file will be indexed using the indexing component. In total, we will have two inverted indexes for reasons that will be explained in the next chapters. Figure 5 shows the design of the dataset preparation component.

4.4.2 Retrieval Component

Retrieval component takes a query or a list of queries and returns a ranked list of documents. The first step is to send the query to the classical ranking component to get the candidate list of documents, the second step is to send the candidate list to the neural re-ranking component to be re-ranked, the third step is to send the re-ranked list to the pseudo relevance feedback component to expand the query, and the final step is to re-rank the document list again using the expanded query.

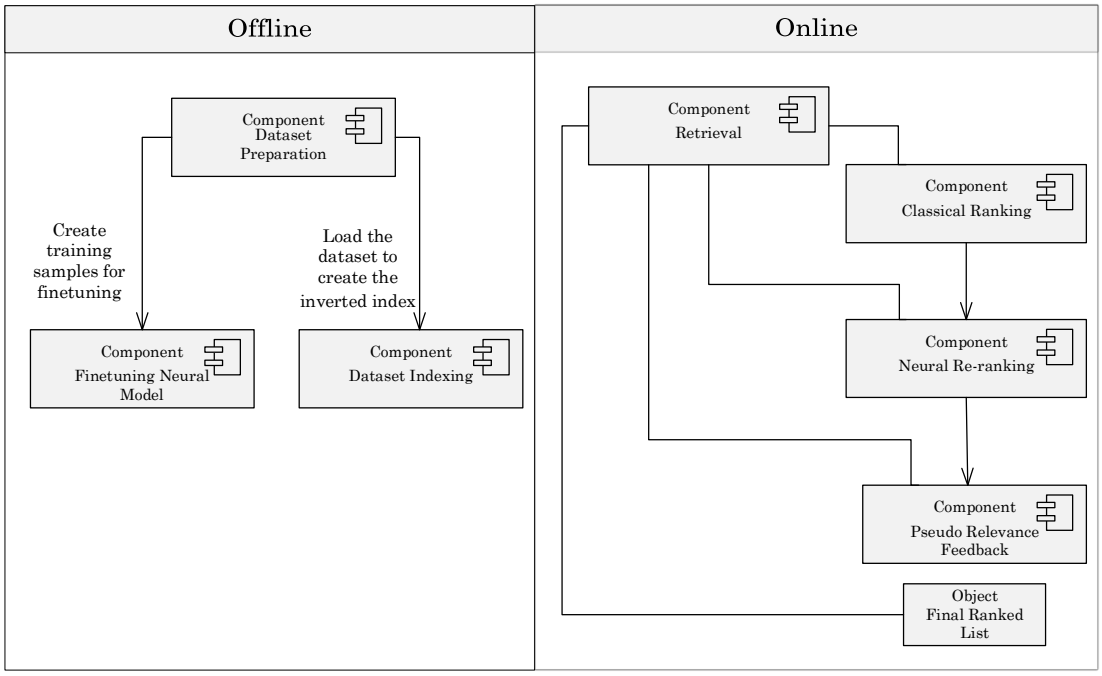

Figure 4 High level design of the system

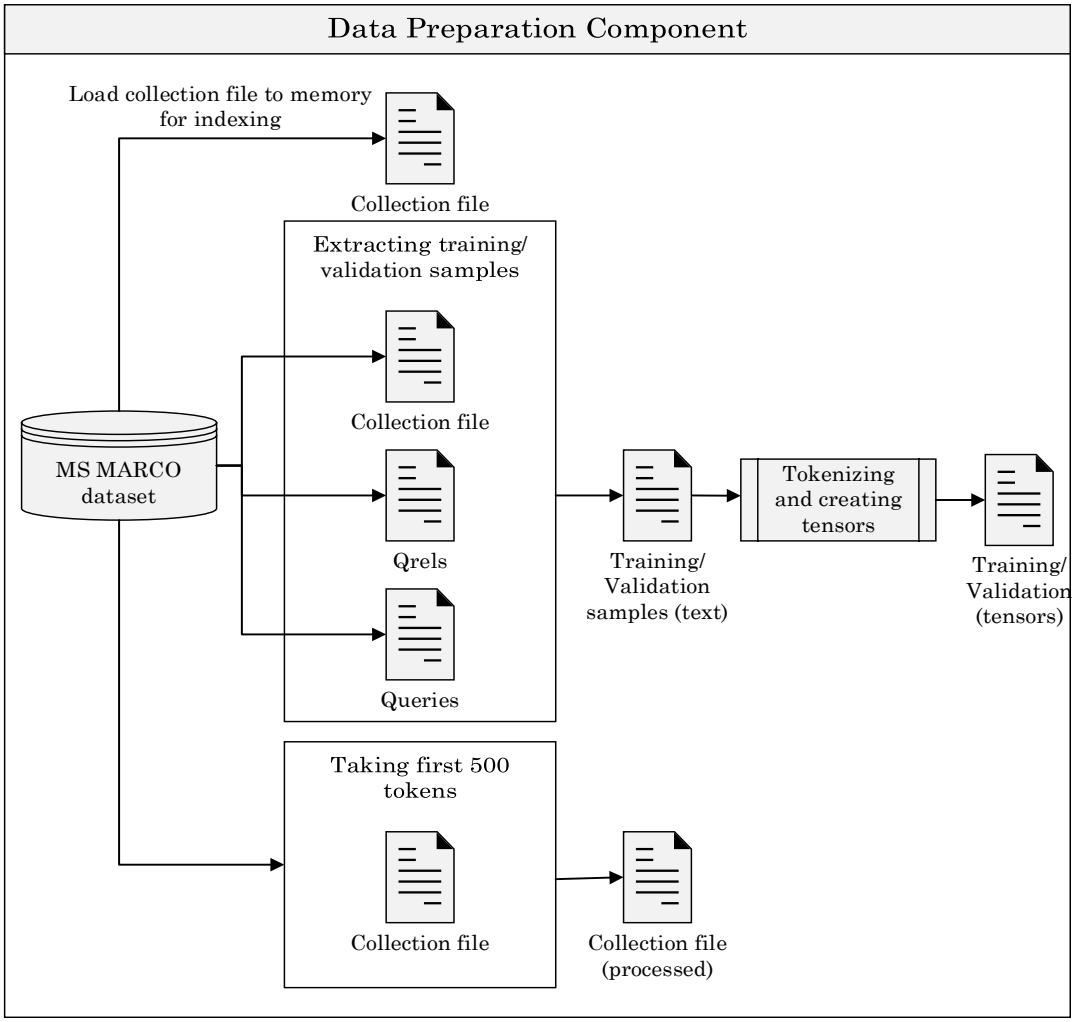

Figure 5 Design of the Data Preparation Component

Chapter 5 Implementation

5.1 Introduction

In this chapter, we will discuss the details of the implementation of the project, we will outline the technologies used in every component and the reasons that led to these choices. This project is implemented using Python programming language, one of the reasons behind this choice is the wide selection of machine learning, natural language processing, and information retrieval libraries available in Python, which made it one of the most popular languages in the field on machine learning [26]. The other reason is the community support, help can be found easily for many problems that may face us. We used Pyterrier IR Platform [20] in multiple parts of the project, this platform was developed by Information Retrieval Group at University of Glasgow. It offers efficient and scalable tools that help build information retrieval systems. We used Pytorch [21] machine learning framework, Pytorch is used in the neural re-ranking component, both BERT and Longformer are implemented using Pytorch and finetuning was implemented using this framework. Pre-trained models are made available by Huggingface [10]. We used Google Colab [9] as the development environment, one of the reasons behind this is that it offers version control to manage changes made to the code base, this is important because it helps if we wanted to check a previous version of the code. The other reason is the availability of cloud GPUs, GPUs are important because neural model training requires hardware that can handle the amount of processing needed in a timely manner, otherwise training time will be unbearable. We used Google Drive to store all project files, including dataset files, finetuned models, and inverted indexes. The reason behind having the project files stored in the cloud is to be safe in case of any failures in local machines, and it is easier to work with Colab if files are stored in Drive.

5.2 Dataset Preparation Component

In this project, dataset preparation component has multiple functionalities. The first is loading the data to create indexes. For the full inverted index, it loads the collection file as explained in the previous chapter, and it sends it to the indexing component to create the inverted index. For the passage inverted index, it loads the collection file and goes through every document in the collection, it takes the first 500 tokens and store that in a new collection file, the new collection file is then sent to indexing component to create the passage inverted index. The second functionality is to create training samples for the finetuning component. MS MARCO dataset only provides positive labels, where for each query there is one document judged relevant, but to finetune the neural models we require negative labels as well. To address this issue, we decided to pick a random document from the collection for each query and assume it is not relevant, now we have a positive label and a negative label for each query which is enough to start the finetuning process. The third functionality is to provide the neural models with the training samples in a suitable format, neural models require the text sequences to be in a specific format and converted to tensors. To address this, we used the tokenizers provided by Huggingface. We will discuss suitable format for BERT and Longformer in finetuning section. We add the tokenized sequences and their corresponding labels to a data structure called TensorDataset, it will be used by

the data loader to feed the neural model batches of the training examples. This component achieves requirement M.1. Table 1 shows MS MARCO document dataset statistics.

File	Number of records	Size
Collection	3,213,835	22 GB
Train queries	367,013	15 MB
Train qrels	367,013	7.6 MB
Validation queries	5,193	216 KB
Validation qrels	5,193	27 MB
Test queries	200	12 KB
Test qrels	16,258	331 KB

Table 1 MS MARCO document dataset statistics

5.3 Neural Re-ranking Component

In this project, we decided to evaluate and discuss two different neural models, namely BERT and Longformer. Since BERT cannot create representations for full documents because of its limitations we discussed in chapter 2, we decided to choose Longformer to address this. We chose these models because we wanted to answer the following research question: does using full document representation instead of just the first passage improve the effectiveness? In some NLP tasks like machine translation, first passage representations are not enough to produce translated text of the full document. In information retrieval, we may not need to use full document representations if the first passage representations can be used to judge the document relevance to a given query accurately. In this section, we will discuss the implementation of the neural models chosen for the project. In this project we used pre-trained models provided by Huggingface. These models were pre-trained on large datasets using masked language modelling (MLM) and next sentence prediction (NSP) as we have discussed in chapter 2. These models have the knowledge of how languages are structured, and this knowledge is leveraged in this project by using the capabilities these models have to judge document relevance. Huggingface provide neural models for many tasks by changing the head (the output layer) on top of the model. For the task at hand, we chose sequence classification head on top of BERT and Longformer, since the task of matching a query to document can be seen as classifying the document as relevant or not relevant to the given query. Huggingface provide tokenizers designed for Transformer-based models, we chose to use these tokenizers to convert text sequences to tensors as we explained in the previous section. Algorithm 1 shows BERT re-ranking process, and Longformer follows the same algorithm with some modifications. This component achieves requiremnt M.3.

5.4 Finetuning Component:

In this project, we finetuned two different neural models, namely BERT and Longformer. The finetuning process starts by loading the training and validation sample sets. From these sample sets we extract input ids, input type ids, and label. Input ids are ids of the tokens in the vocabulary which was obtained from the tokenizer, and input type ids are used to identify which tokens belong to the query and which tokens belong to the document. Next, we feed what we extracted in the

previous step to BERT so we can calculate the loss and optimize the network parameters. We used Adam optimization algorithm [1] as this was recommended in both BERT and Longformer papers. When training is done in the current epoch, we use the validation samples to calculate the average loss and check the progress. Finetuning process of Longformer follows similar workflow. The main difference between finetuning BERT and Longformer is the tokenization process.

BERT Tokenization

- We only take the first 512 tokens of the sequence because of the limitation discussed in chapter 2.
- BERT requires creating a token type ids mask, this mask is used to differentiate tokens from first sequence (the query) and second sequence (the document).
- BERT requires adding the special tokens [CLS] and [SEP].

Longformer Tokenization

- The sequence length is up to 4096 because of the different attention mechanism implemented in Longformer as we have discussed in chapter 2.
- Longformer attention mechanism requires creating a mask to determine which tokens attend to other tokens globally and which tokens attend locally, this can be done by creating a tensor of the same size as the input sequence, for tokens attending globally we assign 1 in the corresponding position, for tokens attending locally we assign 0, query tokens will attend to all other tokens (global) and document tokens will attend to tokens in their window (local).
- Longformer requires adding the special tokens <s> and </s>

Figure 6 shows the structure of the BERT tokenized sequence, and Figure 7 shows Longformer. Algorithms 2 and 3 show the workflow of the finetuning process for BERT and Longformer, respectively.

5.5 Retrieval Component

This component acts as the pipeline of the system. It accepts user queries in a text format for single queries, or a list for multiple queries. It starts by invoking the classical ranking component (BM25) to get the candidate document list, it will then invoke the neural re-ranking component (BERT or Longformer) to re-rank the candidate document list, it will then invoke the pseudo relevance feedback component to expand the query, finally it will re-rank the document list using the expanded query and output the final ranked list. This component achieves requirement S.2

5.6 Classical Ranking Component

This component provides the candidate document list to be re-ranked by the neural re-ranking component. This component is the only component in the retrieval pipeline that interacts with the collection directly, other components only build up on what this component retrieve. We used BM25 as we have discussed in chapter 2. Since BM25 implementation is available in Pyterrier platform, we decided to adopt it in our project. The reason behind that is classical ranking models do not

require finetuning like neural models, so there is no benefit from implementing it again. Pyterrier's implementation follows equation 1 listed in chapter 2. This component achieves requirement M.2.

5.7 Pseudo Relevance Feedback Component

This component is used for query expansion. It takes the re-ranked document list and picks top K documents, from those documents it takes top k tokens based on their informativeness, finally it adds those tokens to the original query. We implemented Bo1 from Divergence from Randomness framework (DFR) as we have discussed in chapter 2. Values for K and k are 5 and 10, respectively. This component achieves requirement M.4.

5.7.1 Passage inverted index

As we have mentioned in previous sections, we decided to create another inverted index using only the first 500 tokens. The reason behind that is BERT number of tokens limitation, because of this limitation we discussed earlier BERT cannot accept more than 512 tokens. So, if we expand the query with tokens that appear in later passages BERT may not benefit from them and it may hurt the results. By using this method, we are only adding tokens that will appear within the 512 tokens window. This index is only used for query expansion, as we are using the full inverted index for retrieval with BM25. More about results and benefits will be discussed in the next chapter.

Algorithm 1 BERT Re-ranking

Input: Candidate list of document **CD**
Output: Re-ranked list of document **RD**
P = [ ] (Empty list)
Batches **B** = Tokenize(**CD**)
for B = 1, *len*(**B**) **do**
  Extract input ids **input**, and input type ids **type** from **B**
  Result = **BERT**(**input, type**)
  P.append(**Result.logits**)
end for
Predictions = Sigmoid(**P**)
RD = Sort(**Predictions**)
return **RD**

Input Sequence												
[CLS]	q1	q2	q3	q4	…	[SEP]	d1	D2	D3	D4	…	[SEP]
Token Type IDs												
0	0	0	0	0	0	0	1	1	1	1	1	1

Figure 6 BERT tokenized structure

Input Sequence												
<s>	q1	q2	q3	q4	...	</s>	d1	D2	D3	D4	...	</s>
Global attention mask												
0	0	0	0	0	0	0	1	1	1	1	1	1

Figure 7 Longformer tokenized structure

Algorithm 2 Finetuning BERT

Input: training sample batches **TB**, validation sample batches **VB**, and number of epochs **E**
Output: Finetuned BERT checkpoints **BERT**
for $Epoch = 1, E$ **do**
  for $TB = 1, len(TB)$ **do**
    Extract input ids **input**, input type ids **type**, and labels **L** from **TB**
    Result = **BERT**(**input, type, L**)
    Loss = **Result.loss**
    Loss.backward
    optimizer.step (AdamW opimizer)
    scheduler.step (Linear scheduler with warmup)
  end for
  for $VB = 1, len(VB)$ **do**
    Extract input ids **input**, input type ids **type**, and labels **L** from **VB**
    Result = **BERT**(**input, type, L**)
    Loss = **Result.loss**
    TotalLoss = **TotalLoss** + **Loss**
  end for
  AVGLoss = **TotalLoss** / len(**VB**)
end for
return **BERT**

Algorithm 3 Finetuning Longformer

Input: training sample batches **TB**, validation sample batches **VB**, and number of epochs **E**
Output: Finetuned Longformer checkpoints **Longformer**
for $Epoch = 1, E$ **do**
  for $TB = 1, len(TB)$ **do**
    Extract input ids **input**, global attention mask **G**, and labels **L** from **TB**
    Result = **Longformer**(**input, G, L**)
    Loss = **Result.loss**
    Loss.backward
    optimizer.step (AdamW opimizer)
    scheduler.step (Linear scheduler with warmup)
  end for
  for $VB = 1, len(VB)$ **do**
    Extract input ids **input**, global attention mask **G**,and labels **L** from **VB**
    Result = **Longformer**(**input, G, L**)
    Loss = **Result.loss**
    TotalLoss = **TotalLoss** + **Loss**
  end for
  AVGLoss = **TotalLoss** / len(**VB**)
end for
return **Longformer**

Chapter 6 Evaluation

6.1 Introduction

In this chapter, we will evaluate the system, discuss the results, and answer the research questions. We will list some of the hyperparameters we used in implementation.

6.1.1 BERT Hyperparameters

- **Optimizer**: AdamW (Adaptive learning rate)
- **Learning rate**: 3e-6
- **Number of warmup steps**: 10000
- **Batch size**: 16
- **Number of epochs:** 2
- **Number of tokens:** 512

6.1.2 Longformer Hyperparameters

- **Optimizer**: AdamW (Adaptive learning rate)
- **Learning rate**: 3e-5
- **Number of warmup steps**: 5000
- **Batch size**: 16
- **Number of epochs:** 2
- **Number of tokens:** 1024
- **Local attention window:** 512

We will discuss the effect of Longformer's number of tokens in the evaluations.

6.2 Experimental Setup

In this section, we will discuss the testing dataset, evaluation metrics, and baselines.

6.2.1 Dataset

We will use the same dataset we used for finetuning and validating the neural models, namely MS MARCO document dataset. This dataset has been partitioned into training, validation, and test partitions. We used the training and validation partitions as we demonstrated in the previous chapter. For the purpose of evaluating the performance of our proposed system, we will use the test partition. This partition has 43 queries that has been judged using a scale of four labels: [3] Perfectly relevant, [2] Highly relevant, [1] Related, and [0] Irrelevant.

6.2.2 Evaluation metrics

In this subsection, we will discuss the evaluation metrics we chose to evaluate the system. These metrics can help us measure the quality of the retrieved documents and whether the proposed system outperform the baseline. We will use the following five metrics:

- **Precision (P) [11]:** is defined as "the fraction of retrieved documents that are relevant". In simpler words, this metric measures how many retrieved documents are actually relevant, regardless of if it had to miss some relevant documents. The aim here is for accuracy.

$$Presicion = \frac{\text{nmuber of relevant documents retrieved}}{\text{number of retrieved documents}}$$

- **Recall (R) [11]:** is defined as "the fraction of relevant documents that are retrieved". In simpler words, this metric measures how many relevant documents the system was able to retrieve, regardless of if it had to retrieve some non-relevant documents. The aim here is to get more relevant documents even if it affects the accuracy.

$$Recall = \frac{\text{nmuber of relevant documents retrieved}}{\text{number of relevant docuemts}}$$

- **Mean Average Precision (MAP) [11]:** Average precision is calculated by taking the precision value at the rank where a relevant document is retrieved. These values are then summed and divided by the number of relevant documents. Mean average precision is calculated by calculating average precision for all queries and then dividing by the number of queries. For this metric to measure the performance accurately, the test partition has to have many relevance judgments per query, which is the case in MS MARCO document dataset.

$$MAP = \frac{1}{|Q|}\sum_{q=1}^{|Q|} \mathrm{AP(q)}$$

- **Mean Reciprocal Rank (MRR) [14]:** Reciprocal rank is reciprocal of the rank at which the first relevant document is retrieved. Mean reciprocal rank is calculating the reciprocal rank for all queries and then dividing by the number of queries. This metric is highly sensitive to rank position.

$$MRR = \frac{1}{|Q|}\sum_{q=1}^{|Q|} \frac{1}{rank_{qi}}$$

- **Normalized Discounted Cumulative Gain (NDCG) [11]:** Discounted gain (DG) requires multiple relevance labels to be calculated. It is calculated by dividing the label numerical value by log of the rank. This metric assumes that the gain of seeing a relevant document in a lower rank is discounted. Discounted cumulative gain (DCG) is calculated by summing the discounted gain value for all ranks. And normalized discounted cumulative gain is calculated by dividing the discounted cumulative gain by the ideal discounted cumulative gain (IDCG), which is the DCG if document ranking was perfect.

$$NDCG = \frac{DCG}{IDCG}$$

Each metric can be evaluated at a cutoff point, for example we can calculate the precision at rank 10 (P@10), which means we only consider the top ten documents for evaluation. These metrics achieve requirement S.1.

6.2.3 Baseline

To be able to measure improvements, we need to set a baseline and compare the proposed system to it. We chose BM25 as our baseline. BM25 is a classical statistical ranking model that we used as the first component in our pipeline. Being one of the most popular classical models, we want to see if our proposed system can outperform it. This baseline achieves requirement M.5.

6.3 Results and Evaluation

We evaluated the system using the metrics mentioned in subsection 6.2.2, but we applied cutoffs to them, we used P@10, R@100, MAP@100, MRR@10, and NDCG@10. In precision, MRR, and NDCG, we used cutoff 10 because these three metrics are more suitable to measure the quality of the top-ranking documents which most people usually look at. On the other hand, in recall and MAP, we used cutoff 100 because these two metrics are more suitable to measure the quality of a broader range.

In table 2, we reported the results achieved by the systems proposed compared to the baseline. Before discussing the results, we want to clarify the abbreviations used, FI and PI specify the inverted index used for query expansion, FI stands for full index and PI stands for passage index. The letter J in FIJ and PIJ stands for just re-rank, which means instead of retrieving a new candidate list using the expanded query, we use the expanded query to re-rank the initial candidate list. Based on the results shown in table 2, we will try to answer the following research questions:

- **Does using the pipeline approach improve effectiveness when compared to classical ranking model?**

  Results have shown improvement compared to BM25. BERT outperformed BM25 is all metrics with significant improvement in precision, Longformer on the other hand did not have a significant improvement in any metric but it performed better overall. This improvement is due to the neural models' ability to build contextual representations to get better matching results. BM25 on the other hand, relies on statical analysis, which cannot capture the semantics.

- **Does the pseudo relevance feedback improve the effectiveness?**

  According to the results, pseudo relevance feedback using the full index is not improving the effectiveness of BERT's pipeline, in fact it got surprisingly worse. We were surprised because we assumed that PRF would improve the effectiveness of BERT like it is improving the effectiveness of classical models, but that assumption was incorrect. After investigation we produced two explanations for this; the first is the number of tokens that BERT can handle compared to the length of the document. What if the added terms from PRF do not come from the first passage? That

would confuse BERT because these terms do not appear in the window that BERT can see. This was the reason we built a second inverted index that covers only the first passage of every document, while there is no significant improvement compared to BM25, it is performing better than PRF with full index. The second explanation is about the way that PRF works. PRF adds terms to the end of the original query ordered by score which may look random, while this has no effect on classical models, BERT and contextualized neural models expect a complete sentence that have meaning. To show an example, this is a query from the test dataset "what slows down the flow of blood" and this is the expanded query "what slows down the flow of blood flow slow blood narrow atherosclerosi doctor heart occur increas vessel". The expanded query is clearly not a complete sentence and words are added in a random order. We will try to investigate this issue further in future research. Longformer also did not perform well with PRF, results from full index were not reported in the table but it performed worse. When we used the passage index the results got better as shown in the table except for MRR.

- **Is just re-ranking using the new expanded query better than retrieving a new candidate list?**

  Results has shown that just re-ranking was better in in both BERT and Longformer. The reason behind that is we are not introducing new documents in the case of just re-rank, we are just re-ranking the list that was already retrieved, which is the same list that was used to expand the query. On the other, retrieving a new list means that BM25 will retrieve new documents based on the expanded query, those documents may have the elite (distinguishing) terms in later passages which BERT cannot see.

- **What effect does the number of tokens have in Longformer?**

  Results suggest that there is a point where increasing the number of tokens does not improve the results, using 2048 or even 4096 tokens did not improve the results when compared to 1024 tokens, this may be specific to this dataset, this can be investigated further in future research.

- **Does using full document representation instead of just the first passage improve the effectiveness?**

  Based on the results, we do not see any benefit from using full document representations when compared to using first passage representation, the reason behind this can be that the elite terms are usually found in the introduction or the abstract which is usually the first passage of the document.

Model	Number of tokens	P@10	R@100	MAP@100	MRR@10	NDCG@10
BM25	N/A	0.6116	0.3937	0.2461	0.8722	0.5400
BM25 + BERT	512	**0.6814**	0.3939	0.2607	0.9138	**0.5858**
BM25 + BERT + PRF (FI)	512	0.6488	0.4002	0.2547	0.8236	0.5423
BM25 + BERT + PRF (PI)	512	0.6698	0.4270	0.2719	0.8527	0.5759
BM25 + BERT + PRF (PIJ)	512	**0.6884**	**0.4142**	0.2655	0.8561	**0.5909**
BM25 + Longformer	512	0.6581	0.3892	0.2427	0.8064	0.5452
BM25 + Longformer	1024	0.6628	0.4061	0.2596	0.8917	0.5737
BM25 + Longformer	2048	0.6628	0.4053	0.2588	0.9033	0.5748
BM25 + Longformer	4096	0.6628	0.4053	0.2588	0.9033	0.5748
BM25 + Longformer + PRF (PI)	1024	0.6465	**0.4306**	0.2677	0.7985	0.5524
BM25 + Longformer + PRF (PIJ)	1024	0.6767	**0.4181**	0.2670	0.8521	0.5762

Table 2 Results, bold marks significant improvement p-value <0.05

Chapter 7 Conclusion

7.1 Summary

In conclusion, we designed and implemented an information retrieval system for the document ranking task using a pipeline of three components, namely the classical model for initial ranking (BM25), the neural model for re-ranking (BERT or Longformer) and the relevance feedback function (Bo1). The neural models were finetuned to MS MARCO document dataset, and we used the same dataset for evaluation. The project has met all "Must have" and "Should have" requirements. We showed that BERT pipeline has improved the effectiveness of the system significantly. Moreover, we introduced the use of the passage index for pseudo relevance feedback which expands the query using the first 500 tokens of the documents, and we showed its effectiveness. Furthermore, we showed that using the expanded query to re-rank the same retrieved document list is better than gathering more document. Moreover, we showed that full document representations are not essential for matching, and first passage is enough to match a query to a document. In addition, we discussed the significance of the number of tokens in Longformer, we showed that creating a representation based on 2048, or 4096 tokens does not improve the effectiveness, and we found that 1024 tokens to be suitable for matching.

7.2 Future Work

In future work, we plan to investigate the following:

- Methods to expand the query in a way that Transformer-based models can fully leverage. Because we assume that adding words to the end of the query in order of score is not optimum, and we want to investigate a way to do that properly.
- We want to further investigate the effect of full document representations in information retrieval. Is there a need for full document representations to match a query to a document effectively? We will use more datasets to get more generalizable results.
- Methods that allow BERT to go beyond the number of tokens limit efficiently. We want to investigate this because it may lead to creating better representations without sacrificing efficiency.

Bibliography

[1] D. P. Kingma and J. Ba, “Adam: A Method for Stochastic Optimization.” arXiv, Jan. 29, 2017. Accessed: Aug. 14, 2022. [Online]. Available: http://arxiv.org/abs/1412.6980

[2] B. Mitra and N. Craswell, “An Introduction to Neural Information Retrieval,” Foundations and Trends® in Information Retrieval, vol. 13, no. 1, pp. 1–126, Dec. 2018.

[3] A. Vaswani et al., “Attention Is All You Need.” arXiv, Dec. 05, 2017. Accessed: Aug. 14, 2022. [Online]. Available: http://arxiv.org/abs/1706.03762

[4] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding.” arXiv, May 24, 2019. Accessed: Aug. 14, 2022. [Online]. Available: http://arxiv.org/abs/1810.04805

[5] S. Hatton, “Choosing the Right Prioritisation Method,” in 19th Australian Conference on Software Engineering (aswec 2008), Perth, Australia, Mar. 2008, pp. 517–526. doi: 10.1109/ASWEC.2008.4483241.

[6] J. Xu, X. He, and H. Li, “Deep Learning for Matching in Search and Recommendation,” in The 41st International ACM SIGIR Conference on Research & Development in Information Retrieval, Ann Arbor MI USA, Jun. 2018, pp. 1365–1368. doi: 10.1145/3209978.3210181.

[7] Z. Dai and J. Callan, “Deeper Text Understanding for IR with Contextual Neural Language Modeling,” in Proceedings of the 42nd International ACM SIGIR Conference on Research and Development in Information Retrieval, Jul. 2019, pp. 985–988. doi: 10.1145/3331184.3331303.

[8] T. Mikolov, K. Chen, G. Corrado, and J. Dean, “Efficient Estimation of Word Representations in Vector Space.” arXiv, Sep. 06, 2013. Accessed: Aug. 14, 2022. [Online]. Available: http://arxiv.org/abs/1301.3781

[9] E. Bisong, “Google Colaboratory,” in Building Machine Learning and Deep Learning Models on Google Cloud Platform: A Comprehensive Guide for Beginners, Berkeley, CA: Apress, 2019, pp. 59–64. doi: 10.1007/978-1-4842-4470-8_7.

[10] T. Wolf et al., “HuggingFace’s Transformers: State-of-the-art Natural Language Processing.” arXiv, Jul. 13, 2020. Accessed: Aug. 14, 2022. [Online]. Available: http://arxiv.org/abs/1910.03771

[11] H. S. Christopher D. Manning, Prabhakar Raghavan, Introduction to Information Retrieval. 2008. [Online]. Available: http://nlp.stanford.edu/IR-book/

[12] T.-Y. Liu, “Learning to Rank for Information Retrieval,” FNT in Information Retrieval, vol. 3, no. 3, pp. 225–331, 2007, doi: 10.1561/1500000016.

[13] I. Beltagy, M. E. Peters, and A. Cohan, “Longformer: The Long-Document Transformer.” arXiv, Dec. 02, 2020. Accessed: Aug. 14, 2022. [Online]. Available: http://arxiv.org/abs/2004.05150

[14] N. Craswell, "Mean Reciprocal Rank," in Encyclopedia of Database Systems, L. LIU and M. T. ÖZSU, Eds. Boston, MA: Springer US, 2009, pp. 1703–1703. doi: 10.1007/978-0-387-39940-9_488.

[15] B. Mitra and N. Craswell, "Neural Models for Information Retrieval." arXiv, May 03, 2017. Accessed: Aug. 14, 2022. [Online]. Available: http://arxiv.org/abs/1705.01509

[16] S. Robertson, S. Walker, S. Jones, M. M. Hancock-Beaulieu, and M. Gatford, "Okapi at TREC-3," in Overview of the Third Text REtrieval Conference (TREC-3), Jan. 1995, Overview of the Third Text REtrieval Conference (TREC–3)., pp. 109–126. [Online]. Available: https://www.microsoft.com/en-us/research/publication/okapi-at-trec-3/

[17] N. Craswell, B. Mitra, E. Yilmaz, D. Campos, and E. M. Voorhees, "Overview of the TREC 2019 deep learning track." arXiv, Mar. 18, 2020. Accessed: Aug. 14, 2022. [Online]. Available: http://arxiv.org/abs/2003.07820

[18] G. Amati and C. J. Van Rijsbergen, "Probabilistic models of information retrieval based on measuring the divergence from randomness," ACM Trans. Inf. Syst., vol. 20, no. 4, pp. 357–389, Oct. 2002, doi: 10.1145/582415.582416.

[19] G. Amati, "Probability models for information retrieval based on divergence from randomness," Thesis (PhD), University of Glasgow, 2003. [Online]. Available: https://theses.gla.ac.uk/1570/

[20] C. Macdonald, N. Tonellotto, S. MacAvaney, and I. Ounis, "PyTerrier: Declarative Experimentation in Python from BM25 to Dense Retrieval," in Proceedings of the 30th ACM International Conference on Information & Knowledge Management, Virtual Event Queensland Australia, Oct. 2021, pp. 4526–4533. doi: 10.1145/3459637.3482013.

[21] A. Paszke et al., "PyTorch: An Imperative Style, High-Performance Deep Learning Library," in Advances in Neural Information Processing Systems 32, Curran Associates, Inc., 2019, pp. 8024–8035. [Online]. Available: http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

[22] S. E. Robertson and S. Walker, "Some Simple Effective Approximations to the 2-Poisson Model for Probabilistic Weighted Retrieval," in SIGIR '94, B. W. Croft and C. J. van Rijsbergen, Eds. London: Springer London, 1994, pp. 232–241. doi: 10.1007/978-1-4471-2099-5_24.

[23] J. Alammar, "The Illustrated BERT, ELMo, and co. (How NLP Cracked Transfer Learning)." https://jalammar.github.io/illustrated-bert/ (accessed Jul. 15, 2022).

[24] Y. Mingfeng, "Neural Search Engine," Thesis (Honors), University of Glasgow, 2021.

[25] J. Alammar, "Visualizing A Neural Machine Translation Model (Mechanics of Seq2seq Models With Attention)." https://jalammar.github.io/visualizing-neural-machine-translation-mechanics-of-seq2seq-models-with-attention/ (accessed Jul. 13, 2022).

[26] C. Voskoglou, "What is the best programming language for Machine Learning?" https://towardsdatascience.com/what-is-the-best-programming-language-for-machine-learning-a745c156d6b7 (accessed Mar. 08, 2022).

[27] I. Goodfellow, Y. Bengio, and A. Courville, Deep Learning. MIT Press, 2016. [Online]. Available: http://www.deeplearningbook.org

[28] M.-T. Luong, H. Pham, and C. D. Manning, “Effective Approaches to Attention-based Neural Machine Translation.” arXiv, Sep. 20, 2015. Accessed: Aug. 14, 2022. [Online]. Available: http://arxiv.org/abs/1508.04025

[29] C. Li, A. Yates, S. MacAvaney, B. He, and Y. Sun, “PARADE: Passage Representation Aggregation for Document Reranking.” arXiv, Jun. 10, 2021. Accessed: Aug. 15, 2022. [Online]. Available: http://arxiv.org/abs/2008.09093

[30] J. Alammar, “The Illustrated Transformer.” https://jalammar.github.io/illustrated-transformer/ (accessed Jul. 14, 2022).

[31] J. Pennington, R. Socher, and C. D. Manning, “GloVe: Global Vectors for Word Representation,” in Empirical Methods in Natural Language Processing (EMNLP), 2014, pp. 1532–1543. [Online]. Available: http://www.aclweb.org/anthology/D14-1162

[32] Q. Liu, M. J. Kusner, and P. Blunsom, “A Survey on Contextual Embeddings.” arXiv, Apr. 13, 2020. Accessed: Aug. 22, 2022. [Online]. Available: http://arxiv.org/abs/2003.07278

[33] L. Page, S. Brin, R. Motwani, and T. Winograd, “The PageRank Citation Ranking: Bringing Order to the Web.,” Stanford InfoLab, Technical Report 1999–66, Nov. 1999. [Online]. Available: http://ilpubs.stanford.edu:8090/422/