DEGREE PROJECT IN COMPUTER SCIENCE AND ENGINEERING,
SECOND CYCLE, 30 CREDITS
STOCKHOLM, SWEDEN 2021

Natural Language Processing Model for Log Analysis to Retrieve Solutions For Troubleshooting Processes

NÚRIA MARZO I GRIMALT

KTH ROYAL INSTITUTE OF TECHNOLOGY
SCHOOL OF ELECTRICAL ENGINEERING AND COMPUTER SCIENCE

Natural Language Processing Model for Log Analysis to Retrieve Solutions For Troubleshooting Processes

Núria Marzo i Grimalt

Master's Programme, Machine Learning, 120 credits
Date: June 16, 2021

Supervisors: Amir H. Payberah, Serveh Shalmashi, Forough Yaghoubi
Examiner: Sarunas Girdzijauskas
School of Electrical Engineering and Computer Science
Host company: Ericsson AB
Swedish title: En NLP-model för analys av loggar för att inhämta lösningar till felsökningsprocesser

Abstract

In the telecommunications industry, one of the most time-consuming tasks is troubleshooting and the resolution of Trouble Report (TR) tickets. This task involves the understanding of textual data which can be challenging due to its domain- and company-specific features. The text contains many abbreviations, typos, tables as well as numerical information. This work tries to solve the issue of retrieving solutions for new troubleshooting reports in an automated way by using a Natural Language Processing (NLP) model, in particular Bidirectional Encoder Representations from Transformers (BERT)-based approaches. It proposes a text ranking model that, given a description of a fault, can rank the best possible solutions to that problem using answers from past TRs. The model tackles the trade-off between accuracy and latency by implementing a multi-stage BERT-based architecture with an initial retrieval stage and a re-ranker stage. Having a model that achieves a desired accuracy under a latency constraint allows it to be suited for industry applications. The experiments to evaluate the latency and the accuracy of the model have been performed on Ericsson's troubleshooting dataset. The evaluation of the proposed model suggest that it is able to retrieve and re-rank solution for TRs with a significant improvement compared to a non-BERT model.

Keywords

Trouble Report, Recommender System, BERT, Information Retrieval, Natural Language Processing, Multi-Stage Ranking

Sammanfattning

En av de mest tidskrävande uppgifterna inom telekommunikationsindustrin är att felsöka och hitta lösningar till felrapporter (TR). Denna uppgift kräver förståelse av textdata, som försvåras as att texten innehåller företags- och domänspecifika attribut. Texten innehåller typiskt sett många förkortningar, felskrivningar och tabeller blandat med numerisk information. Detta examensarbete ämnar att förenkla inhämtningen av lösningar av nya felsökningar på ett automatiserat sätt med hjälp av av naturlig språkbehandling (NLP), specifikt modeller baserade på dubbelriktad kodrepresentation (BERT). Examensarbetet föreslår en textrankningsmodell som, givet en felbeskrivning, kan rangordna de bästa möjliga lösningarna till felet baserat på tidigare felsökningar. Modellen hanterar avvägningen mellan noggrannhet och fördröjning genom att implementera den dubbelriktade kodrepresentationen i två faser: en initial inhämtningsfas och en omordningsfas. För industrianvändning krävs att modellen uppnår en given noggrannhet med en viss tidsbegränsning. Experimenten för att utvärdera noggrannheten och fördröjningen har utförts på Ericssons felsökningsdata. Utvärderingen visar att den föreslagna modellen kan hämta och omordna data för felsökningar med signifikanta förbättringar gentemot modeller utan dubbelriktad kodrepresentation.

Nyckelord

Felrapporter, Rekommendatorsystem, BERT, Informationsinhämtning, Naturlig Språkbehandling, Dubbelriktade Ranking

Acknowledgments

The work presented in this master thesis has been conducted at Ericsson's Global Artificial Intelligence Accelerator (GAIA) department in Stockholm between January and June of 2021.

First and foremost, I want to thank my supervisors Serveh Shalmashi and Forough Yaghoubi (Ericsson), as well as Amir H. Payberah (KTH) for giving me the opportunity to work alongside them. Additionally, I am grateful for their guidance and advice throughout the master thesis and, more generally, on life.

Also from Ericsson, I want to thank Leif Jonsson for his help, as well as his interesting questions and points of view, and Tahar Zanouda for the resources he has provided us. Moreover, I want to acknowledge Frida Svensson and Viveca Lindahl for their cooperation in translating the abstract and for welcoming me to the team. And finally, Caroline Jacobson for making sure I feel welcomed to her team and providing me with any help I need.

Last but not least, I have nothing but words of gratitude to my parents, Hector and Catalina, and my grandparents. They always encouraged me to explore the world and follow my instincts, even when it meant moving to Sweden for almost two years. Additionally, I want to show gratitude for my sister Joana, for being a constant support and a source of entertainment, specially during the first months of the pandemic. I, finally, want to thank my partner Pau, as he has been by my side along this journey and has supported me no matter what.

Stockholm, June 2021
Núria Marzo i Grimalt

Contents

List of Figures

List of Tables

List of acronyms and abbreviations

ANN Artificial Neural Network

AP Average Precision

BERT Bidirectional Encoder Representations from Transformers

BM25 Best Match 25

CNN Convolutional Neural Network

DESM Dual Embeddig Space Model

DRMM Deep Relevance Matching Model

DSSM Deep Structured Semantic Model

ELMo Embeddings from Language Models

HW Hardware

IR Information Retrieval

KNRM Kernel Neural Relevance Model

LDA Latent Dirichlet Allocation

LSTM Long Short-Term Memory

MAP Mean Average Precision

ML Machine Learning

MLM Masked Language Model

MRR Mean Reciprocal Rank

MSMARCO Microsoft Machine Reading Comprehension

nDCG Normalised Discounted Cumulative Gain

NER Name Entity Recognition

NLP Natural Language Processing

NSP Next Sentence Prediction

RNN Recurrent Neural Network

RR Reciprocal Rank

TR Trouble Report

Chapter 1

Introduction

Handling different types of issues that occur in the complex software and hardware infrastructure in modern telecommunication systems is a slow task. As most of these issues lead to service down time or other forms of harm to the customer experience, they must be quickly detected, identified and resolved, which is often done by engineers in network operation centers.

When the engineers observe a problem related to the hardware or software of the running system that they cannot solve on site, they create a TR (also called trouble ticket or commonly, Bug Report) to track information regarding the detection, characteristics and hopefully an eventual resolution of the problem [2]. According to [3], fault localization "is widely recognized to be one of the most tedious, time consuming, and expensive yet equally critical activities in program debugging".

TR routing and analysis is a labor-intensive process in which engineers analyze characteristics of the problems to find possible solutions, and it opens up the question whether this can be effectively automated. The complexity of the problem and the type of data makes it hard to automate using any hard-coded rules. With today's huge development in the field of Machine Learning (ML), specifically in NLP and Information Retrieval (IR), we can benefit from historical data by analyzing previous trouble tickets using ML and infer a solution to a new problem. The aim is finding a resolution to a TR ticket in an automated way thus substantially shortening the lead time to solve TRs.

One of the most important techniques that have appeared recently in the literature is BERT [4], that is a new language representation model based on the transformers architecture [5]. It uses the attention mechanism that

allows the model to process sentences as a whole unlike other techniques where sentences are processed sequentially. As this technique outperforms the state-of-the-art results in many tasks, we use it as the main component of our architecture.

1.1 Problem Statement

The problem that this master thesis will try to tackle is the issue of retrieving solutions for troubleshooting processes in an automated way by using a NLP model.

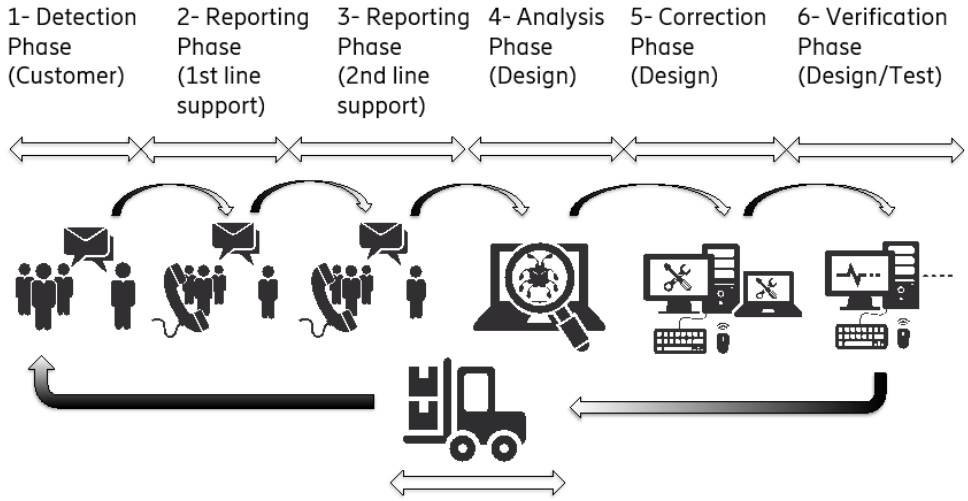


Figure 1.1: Diagram of the the steps in the troubleshooting process.

Figure 1.1 shows the six steps and the complexity of the process of troubleshooting. It starts by detecting a problem in step 1, for example crash in the network. Then, the observations of the problem are reported in a TR in steps 2 and 3. This trouble report will be analyzed and corrected in steps 4 and 5 by engineers that will verify the solution in step 6. This is a time-consuming process, and it requires many steps even to solve minor faults.

To overcome that, this master thesis will design and implement a text ranking model focused on helping automate the steps 4 and 5 of the troubleshooting process. The aim of the model will be to retrieve the most relevant documents with respect to a given query. Particularly, in this master thesis, the query will correspond to an observation of a problem from a TR and the documents to be ranked will be answers from past TRs.

1.2 Research Question and Goals

The research question that the project will tackle is: How can we use BERT-based approaches to find solutions for error reports in the Ericsson logs that improve BM25, the existing non-BERT baseline model at Ericsson? And can this model have a low latency?

Therefore, the goal of this project is to develop a model for text ranking, which aims at retrieving the most relevant answers with regards to a given observation of a trouble report. It will also investigate the trade-off between the computational complexity and latency versus the accuracy of the model. This work has been divided into the following three sub-goals:

1. Analyze the data in which the experiments will be performed using visualization techniques.
2. Study different BERT-based architectures for text ranking and choose the best suited one for our application.
3. Implement the BERT-based chosen architectures and evaluate their accuracy and latency.

1.3 Scope and Delimitations

The master thesis project will have some delimitations that will determine the scope of the project.

As stated, the data from TRs is very complex and difficult to process as it contains specific language and non-textual information. Moreover, all the TR data from Ericsson concerns many different issues and areas, so we expect each TR to explain very distinct problems. As it is easier to build a model for faults from the same area, we have decided to focus on a particular class of TRs: the 4G and 5G troubleshooting data. This data follows a well known structure and can be parsed easily. Also, we will delimit the processing of the data to 4G and 5G telecom-specific text data. This specific dataset contains TRs related to 4G and 5G radio networks faults from an specific period of time of one year.

Another limitation that we have is the computational resources. For that reason, the training that our method will undergo is only a fine-tuning on a

domain-specific task, as the training of large NLP models is a slow task and needs many computational resources. The fine-tuning task consist of taking the weights of a pre-trained model and use them as the initialization of your model to train it further in a domain-specific task. Therefore, the NLP heavy models like BERT used in this master thesis will already be pre-trained on general-domain data, and will be extracted from open-source platforms such as the Hugging Face Library [6], afterwords they will be fine-tuned on the troubleshooting data.

1.4 Outline of the thesis

This thesis is structured as follows:

- Chapter 1: Introduction where a brief overview is given and the research question is stated.
- Chapter 2: Background explains basics of the text ranking problem as well as the related work.
- Chapter 3: Methods formulates the problem, explains the proposed approaches for each module of the text ranking system.
- Chapter 4: Implementation illustrates the implementation details of the experiments.
- Chapter 5: Results and Discussion focus on analyzing the results of the experiments.
- Chapter 6: Conclusions and Future Work provides the conclusion of this thesis work and discussion for further work.

Chapter 2

Background

This chapter will focus on the technical background of the thesis as well as the existing solutions and related work for the problem stated in Section 1.1.

2.1 The Text Ranking Problem

The text ranking problem can be summarized as generating an ordered set of texts retrieved from a corpus of documents in response to a query for a particular task. There are three main elements that compose this problem:

(i) The query that expresses a need of information.

(ii) The corpus of documents where this information will be retrieved.

(iii) The text ranking model that receives as an input the corpus and the query, and outputs the ordered set of documents.

Figure 2.1 shows a diagram of all the elements that compose the text ranking problem. The most common form of text ranking is search [7]. Other forms of text ranking are: question answering [8], community question answering [9], information filtering [10] or text recommendation [11].

Generally, text ranking is not a trivial problem as the system needs to be able to understand the query and find the relevant results in a large set of documents. Some of the first techniques used for this matter were Exact Term Matching techniques [12]. They rely on two main measures: term frequency and document frequency, the former refers to how many times a term occurs

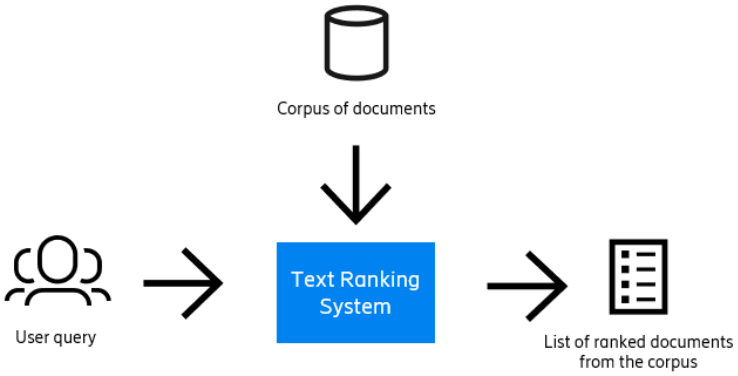


Figure 2.1: Diagram of the text ranking problem.

and the latter shows in how many documents it appears. A query and a document will have a high score if the text from the document and text from the query use the same terms. Indeed, this limits the applicability of the algorithm and its performance when the query and the documents use different words to refer to equivalent things, then there is no exact match. This is referred as the vocabulary mismatch problem [13].

With the revolution of deep learning, the fields of NLP and IR were able to rely less on Exact Term Matching techniques and started using deep neural network models to develop better architectures for text ranking models that focused on semantic matching. In the next sub-sections some of this approaches, relevant to the master thesis, will be presented. First, the main architectures that appeared before BERT will be explained. Then, the BERT methods will be introduced. After that, the state-of-the-art architectures will be commented. Finally, the metrics to evaluate the success of the text ranking model will be stated.

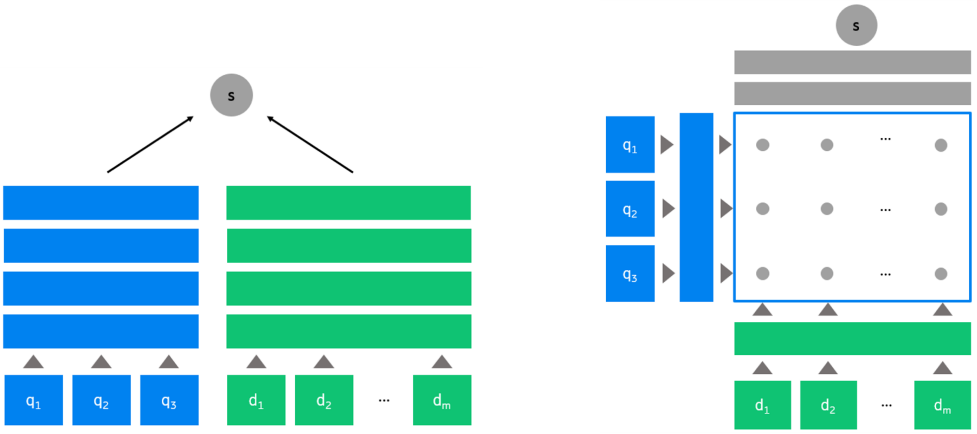
2.1.1 Pre-BERT Methods

Before the appearance of BERT in 2018 [4] the main methods used for text ranking were Exact Matching techniques, as explained in Section 2.1, and neural IR. Neural IR represented the state-of-the-art methods before the BERT model and their main architectures will be explained hereafter.

The pre-BERT neural IR methods can be divided into two main architectures: representation-based approaches and interaction-based approaches.

- A **representation-based** architecture, shown in Figure 2.2a, learns a dense vector representation of the query and the documents independently. Then it computes the similarity between the representations with cosine similarity or inner product. The similarity measure is used as the relevant score to rank the different documents with respect to the query. Some example of this approach are Deep Structured Semantic Model (DSSM) in 2013 [14] and Dual Embeddig Space Model (DESM) in 2016 [15].

- An **interaction-based** architecture, shown in Figure 2.2b, focuses on the interaction between each term of the query with each term of the document and a similarity matrix is created. This matrix undergoes further processing to extract a similarity value. Some examples of this approach are Deep Relevance Matching Model (DRMM) in 2016 [16] and Kernel Neural Relevance Model (KNRM) in 2017 [17].



(a): A generic representation-based neural architecture.

(b): A generic interaction-based neural architecture.

Figure 2.2: Pre-BERT common architectures.

Usually both approaches include neural networks in some of its components. Representation-based approaches use neural models for creating the dense vectors that represent the texts, and interaction-based approaches use neural models to process the similarity matrix and output the similarity value. Each of these approaches has pros and cons, generally studies have shown interaction-based architectures to be more effective but slower than representation-based architectures [18].

2.1.2 BERT Methods

In 2017 transformers were presented by Vaswani et al [5]. A transformer is a sequence to sequence model: it receives a sequence as its input and outputs another sequence. One example of its usage would be machine translation. Before the transformer, those tasks were preformed by Recurrent Neural Network (RNN) and Long Short-Term Memory (LSTM) networks where long-term dependencies were not captured. The transformer was able to solve this issue thanks to its attention mechanism. The attention mechanism is built inside the transformer models with a self-attention module. It is a module that receives a sequence of N inputs and returns N outputs. The inputs interact with each other and as an output we get an aggregate of these interactions.

The transformer is composed by a stack of N encoders and a stack of N decoders. The encoders are focused on building a representation of the input sequence and the decoders are focused on decoding the representation into a sequence of text. The BERT model can be defined as the encoder part of the transformer. BERT was presented in 2018 by Devlin et al [4], and it has revolutionized the NLP and IR fields. It is an architecture built on top of transformers (as explained, it uses the encoder part of the transformer), Embeddings from Language Models (ELMo) [19] as it takes the idea of contextual embeddings and ULMFiT [20] as it uses the proposed pre-training.

Contextual embeddings was an idea introduced by some researchers and it was a step forward of the word embedding used since then. Word embeddings produced the same values independently of the context of the word. For instance, the word *bank* in this sentence: *There is a bank to sit down outside the bank where I opened my account.*, would be expressed with the same values using word embedding but with different values using contextual embeddings as it represents different meanings. This idea was taken by BERT from ELMo and it is one of its main characteristics.

The input of a BERT model usually is a sequence tokenized by the BERT tokenizer. Its aim is to reduce the vocabulary space by splitting words, that way large texts can be modeled by using a small vocabulary (30000 word pieces). The input is divided into two main components: the token embedding done by WordPiece [21] and the position embedding that captures the position of the token in the sequence. The final input is an element-wise summation of all the embeddings. Next, the input is passed through a stack of encoders. Some hyper-parameters that can be fine-tuned are the number of hidden layers, its dimension and the number of attention heads. Finally, the output consist on a

sequence of fixed-size contextual embeddings, one for every token.

The objective used when training a BERT model is Masked Language Model (MLM) and Next Sentence Prediction (NSP). MLM consists of masking a token from the input sequence and asking the model to predict it. NSP consists of giving the model two sentences A and B, and asking the model to determine if sentence B is the follow-up sentence to sentence A.

Variations of the simple BERT model have appeared recently as the field is developing very fast. Some of the new models are:

- **RoBERTa**: It looses the NSP training and, increases the batches of data in the pre-training and pre-trains the model for more epochs [22].
- **AlBERT**: It reduces the number of parameters of the original BERT model by implementing the cross-layer parameter sharing. This allows low memory consumption and limits the overfitting [23].
- **DistilBERT**: It uses distillation to pre-train a general-purpose BERT model that maintains the performance but it is faster and uses less parameters [24].
- **ELECTRA**: It is pre-trained as a discriminator that has to distinguish if the sentences forwarded to the model have a replaced token or not. This model performs better with the minimum amount of pre-training [25].

The first usage of BERT for text ranking was done by Nogueira and Cho in 2019 [26]. They adapted the BERT architecture to be able to ouput a relevance score between the query and a document. As it can be seen in Figure 2.3, the input is the query and the document separated by special tokens: $[CLS]$ and $[SEP]$. The sequence is tokenized using the BERT tokenizer and then is forwarded to the model that outputs a contextual embedding for each of the tokens. Nogueira and Cho [26] used the contextual embedding of the first token $[CLS]$ to derive a similarity score between the query and the document using a linear layer.

BERT-based models present some limitations, as they only accept sequences of less that 512 tokens in the case of BERT base, which limits the length of the text we can input to it. They also present a high complexity and are slow in text ranking tasks.

Figure 2.3: BERT architecture used for text ranking. It includes the special [CLS] and [SEP] tokens to distinguish between the query and the document.

2.1.3 Multi-Stage Architectures

Traditionally proposed approaches to the text ranking problem are one-stage architectures. However, state-of-the-art models follow a multi-stage architecture. These type of models are formed by two main stages, a first stage or initial retrieval phase where a candidate set of text is output, and a second stage or a re-ranker stage where the output is a ranked list of the candidate set.

These type of architectures were developed to tackle the trade-off between effectiveness and efficiency. Effectiveness is how accurate the model is and efficiency is how fast it can output the results. Figure 2.4 shows a diagram of the architecture. The designs of the architectures vary in many different applications as one can decide to completely ignore the ranking scores of the previous stage or add them at every stage, for example. The advantage of having two main stages is that the first stage (initial retrieval) allows the model to discard easy candidates. It is a fast stage where the most obvious non-relevant documents are discarded and a smaller set of documents is passed to the re-ranker stage.

The re-ranker stage can have many different designs: cascade of rankers, ensemble of different models or one large and powerful BERT model above

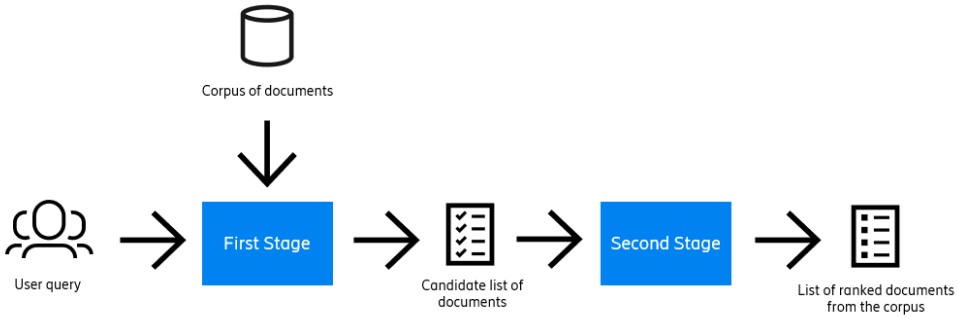


Figure 2.4: General multi-stage architecture.

others. It is a slow stage but as the input of documents is significantly smaller it still maintains the low latency.

The number of document that passes from initial retrieval to the next stage can be computed in different ways. One solution is to keep a certain amount of documents with the highest scores, the second one is to keep a fixed percentage of all document, and the third option is to use a score threshold.

2.1.4 Ranking Metrics

One important aspect of the text ranking models is being able to compare between them to assert which ones works better. In this section some ranking metrics used in the literature will be presented.

To understand the evaluation of text ranking systems better, we need to mathematically define the problem. Given a query q that expresses a need of information and corpus of documents C, the text ranking system purpose is to return a ranked list of K texts from the corpus collection that maximises a particular metric of interest. The parameter K is the retrieval depth.

Some of the most used ranking metrics are:

- **Precision**: given a ranked list of documents R, it is the fraction of documents in R that are relevant. It can be expressed as shown in Equation 2.1:

$$\text{Precision}(R, q) = \frac{\sum_{(i,d)\in R} \text{rel}(q, d)}{|R|} \tag{2.1}$$

where $\text{rel}(q, d)$ is the binary relevance of document d to query q. Precision@K would be the cutoff precision, it can be understood as the fraction of relevant documents from the top-K results. This metric is easy to interpret but only takes into account binary relevance and not the position of the documents in the ranking. For instance, you will get the same Precision@5 if the relevant documents are the number 1 and 2 or if the relevant documents are the number 4 and 5. In both cases Precision@5 = 0.4, but we as users would prefer the first ranked list.

- **Recall**: is fraction of relevant documents for q in the entire corpus C that are retrieved in the ranked list R. It can be expressed as shown in Equation 2.2:

$$\text{Recall}(R, q) = \frac{\sum_{(i,d)\in R} \text{rel}(q, d)}{\sum_{d\in C} \text{rel}(q, d)} \tag{2.2}$$

  It also assumes binary relevance. Recall@K is the measure used if we evaluate the recall at a cutoff K. It doesn't take into account graded relevance (when relevance has more than two values, for example a value between 1 and 5) or the positions in the ranking.

- **Reciprocal Rank (RR)**: can be defined as shown in Equation 2.3:

$$\text{RR}(R, q) = \frac{1}{\text{rank}_i} \tag{2.3}$$

  where rank_i is the smallest rank number of a relevant document. If the first relevant document appears at position 1 then the RR is 1, if it appears at position 3 then RR is 1/3. However, it only captures the appearance of the first relevant results, hence it would be a good metric for question answering problems.

- **Average Precision (AP)**: is a measure that averages the precision scores at different cutoff corresponding to the appearance of the relevant documents. It can be expressed as shown in Equation 2.4:

$$\text{AP}(R, q) = \frac{\sum_{(i,d)\in R} \text{Precision@}i(R, q) \cdot \text{rel}(q, d)}{\sum_{d\in C} \text{rel}(q, d)} \tag{2.4}$$

  where Precision@i is defined in Equation 2.1. This measure captures both precision and recall and favours relevant document appearing at the top of the ranked list.

- **Normalised Discounted Cumulative Gain (nDCG)**: it is a measure designed for graded relevance measures and usually used for web search. It can be expressed as shown in Equation 2.5:

$$\mathrm{DCG}(R,q) = \sum_{(i,d)\in R} \frac{2^{\mathrm{rel}(q,d)} - 1}{\log_2(i+1)} \qquad \mathrm{nDCG}(R,q) = \frac{\mathrm{DCG}(R,q)}{\mathrm{IDCG}(R,q)} \tag{2.5}$$

  where IDCG represents the ideal ranked list.

Some of this statistics can be summarized for many different queries using a simple arithmetic mean. It would consist on adding the values of some of this measures like RR or Precision and dividing it by the number of queries in the test set. The most popular measures used are Mean Average Precision (MAP) and Mean Reciprocal Rank (MRR).

2.2 Related Work

Many BERT-based text ranking models have been developed over the past years. Some examples are: the EPIC model [27], ColBERT [28] or ANCE [29]. They all use the BERT model in diverse ways in order to rank documents with respect to a query and present their results with general-domain data.

Bug analysis and resolution is a topic that the researchers have been interested for a long time. In the literature, there are many papers that propose recommender systems for ticket analysis or bug analysis. The more complex ones use Convolutional Neural Networks (CNNs) or LSTMs like the architecture proposed by [30] but not any BERT-based techniques.

Companies like Ericsson have tried to automate the resolution of faults by using an ensemble of different pre-BERT techniques. Ferland et al. [31] proposed a method for automatically resolving trouble tickets with a hybrid NLP model. They use an ensemble of pre-BERT like LSTM and Latent Dirichlet Allocation (LDA) where the results from each individual model were handled with a stacked ensemble layer. However, the authors of these papers do not consider the latency of the process as part of their performance metric. Another example of an ensemble of pre-BERT techniques is discussed in [32] as well, in their approach they formulate the problem as a non-convex optimization problem which is solved with a heuristic solution for a simplified scenario with focus on accuracy. Consequently, their approach leads to a sub-

optimal solution.

There are many multi-stage BERT-based approaches for text ranking as explained in Section 2.1.3. Some examples are duoBERT [1] or DeeBERT [33], they present their results using general-domain data. However, we are not aware of any that is focused on a domain-specific task like automating the resolution of telecom TRs.

Chapter 3

Methods

In this chapter, the methods used for solving the research question will be explained. First, the general overview of the architecture will be introduced and next, each of the modules of the architecture will be presented.

3.1 Overview of the architecture

The model we use has to be able to:

- Perform text ranking using domain-specific data.
- Output an accurate ranked list of documents with low latency.
- Perform better than a baseline model that does not use BERT.

To fulfill all this characteristics, the proposed model is based on the multi-stage architectures presented in Section 2.1.3, particularly it will be divided into three main stages. It is aimed at retrieving the best possible answers to an observation from a telecom-domain troubleshooting report.

The method processes the data from past trouble reports in order to retrieve and re-rank the best possible solutions from a corpus of past answers, given a new observation of a fault. It achieves a high accuracy while keeping the computational complexity and latency low. As seen in Figure 3.1, it is composed by three main stages: a pre-processing stage, an initial retrieval stage and a re-ranker stage. Each stage has a different purpose, which is explained bellow:

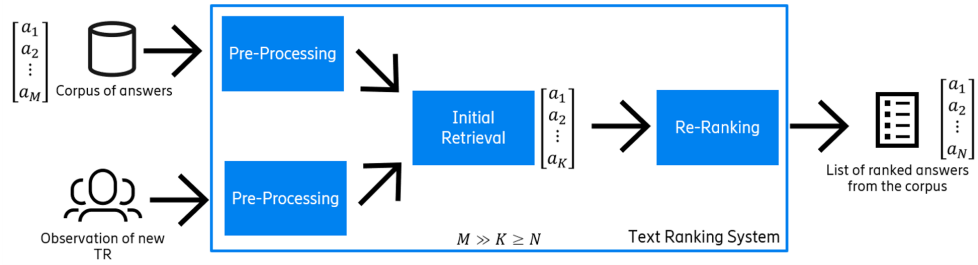


Figure 3.1: Diagram of the proposed solution architecture.

- The **pre-processing stage** is in charge of cleaning the data and prepare it for the next stage. The input of this stage is the query and the corpus of documents, and the output is the pre-processed and cleaned query and documents.
- The **initial retrieval stage** has the purpose of retrieving a candidate list of documents that are relevant to our query from all the corpus of documents. The input to this stage is the pre-processed query and documents, and the output is a top-K candidate list of documents.
- The **re-ranker stage** is in charge of ranking the candidate list provided by the retriever, and output a final list of ranked documents. The input to this stage is the top-K candidate list of documents and the query, and the output is a final top-N list of ranked documents.

The next sections will analyze the three stages in depth, explaining the characteristics and models chosen for each part.

3.2 Query and Document Pre-Processing Stage

The pre-processing stage is focused on preparing the data for the retrieval and re-ranking stages. As explained, the data we will be working with is telecom-specific data and contains company-specific language. Therefore, it needs some specific pre-processing steps.

Figure 3.2 shows the different steps that the query and the documents will go through to prepare them for the next stages. This module is composed by a Spacy Language Processing pipeline [34] where we add custom cleaning modules. The custom cleaning modules are divided into five steps:

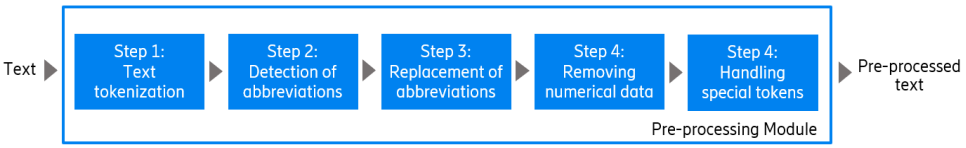


Figure 3.2: Diagram of the the steps in the pre-processing module.

- Step 1, **Text tokenization**: all texts are tokenized with a custom tokenizer that recognizes company- and domain-specific language. For example, it will recognize any name of a company product.
- Step 2, **Detection of abbreviations**: it detects, and tags abbreviations and acronyms with a customized Name Entity Recognition (NER).
- Step 3, **Replacement of abbreviations**: it replaces the detected and tagged telecom abbreviations and acronyms by the complete words. In case of multiple suggestions for a given abbreviation, it picks the suggestion which is most related to radio networks. For example, if one of the acronyms detected is CCS, this module will substitute it for Common Channel Signaling.
- Step 4, **Removing numerical data**: any numerical tokens are removed as they do not provide any useful information to an NLP model. Usually they are part of tables included in the TRs.
- Step 5, **Handling special tokens**: it removes extra spaces, new lines and gaps between words as well as any punctuation signs.

The output of this stage is the pre-processed query and the pre-processed documents ready to be analyzed by the next stages.

3.3 Initial Retrieval Stage

The initial retrieval stage is in charge of receiving the pre-processed query and the pre-processed documents, analyzing them and outputting a top-K candidate list of the most relevant documents to the query.

The characteristics that this stage should fulfill are:

- It needs to have low latency and be able to manage a large amount of data points.

- It needs to output a candidate list of documents that are relevant to the query.
- The list of documents needs to contain all relevant documents, if possible. The documents don't need to be at the first positions but they need to be included in the list.

We will compare two approaches: a BERT-based model and a non BERT-based model that will serve as a baseline model. The BERT-based model is Sentence-BERT [35] and the baseline model is BM25 [36].

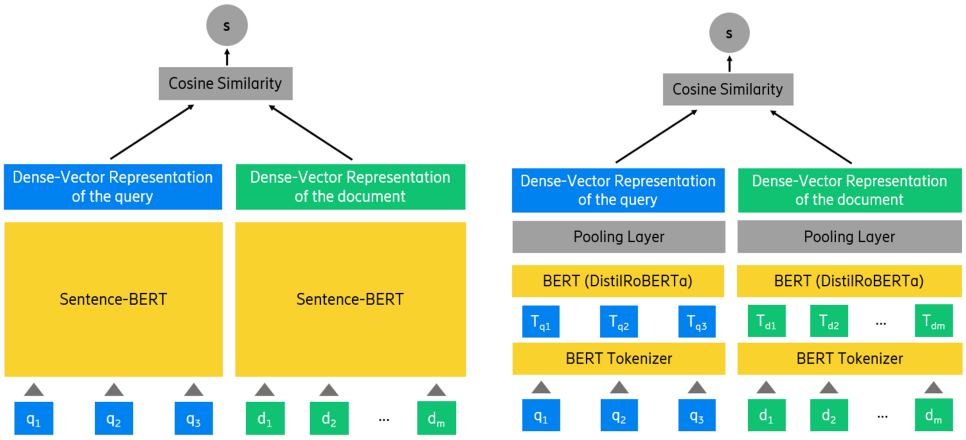
3.3.1 Sentence-BERT

A model that fulfills all the characteristics of the initial retrieval is Sentence-BERT [35]. It is a representation-based architecture (explained on Section 2.1.1), which can be seen in Figure 3.3a. It creates a dense vector representation for the query and a dense vector representation for the documents. These two vector representations are used to compute the similarity value using a similarity measure such as cosine similarity, explained in Equation 3.1. Given two vectors $\mathbf{Q}$ and $\mathbf{D}$, the cosine similarity between them is expressed as:

$$sim(\mathbf{Q}, \mathbf{D}) = \frac{\mathbf{Q} \cdot \mathbf{D}}{\|\mathbf{Q}\| \cdot \|\mathbf{D}\|} = \frac{\sum_i Q_i D_i}{\sqrt{\sum_i Q_i^2}\sqrt{\sum_i D_i^2}} \tag{3.1}$$

Specifically, Sentence-BERT uses a Siamese Network structure. A Siamese Network is a type of Artificial Neural Network (ANN) conformed by two neural networks that share its weights [37]. Each of this two branches is composed by two main layers: first a BERT model, then a mean pooling layer. The mean pooling is performed on the output of the BERT by doing a mean operation of each of the dimensions of the contextual embeddings output by the BERT model. The final output is a fixed-size vector used to compute the cosine similarity. Figure 3.3b shows a detailed diagram of the architecture.

For example, if Sentence-BERT receives two sentences it will first tokenize them, it will forward them to a BERT model that will create a contextual embeding for each of the tokens, and finally a mean pooling will be performed to create the representation of the sentence, which is a fixed-size vector. Once we have the representations of sentence 1 and sentence 2, they will be used to compute the cosine similarity between them using Equation 3.1. The result will be the similarity value between sentence 1 and 2. In the case of text



(a): General architecture of the initial retrieval stage.

(b): Specific architecture of Sentence-BERT.

Figure 3.3: Diagram of the initial retrieval stage.

ranking, sentence 1 will always be the query and sentence 2 will be each of the documents in the corpus. The similarity value will correspond to the ranking score and will be used to create the top-K candidate list.

As explained, Sentence-BERT has a BERT-model inside its architecture. There exist different BERT models, listed in Section 2.1.2, the one used in this approach will be DistilRoBERTa [24]. The main reason is that is has a low complexity as it is a distilled model, and also has the high accuracy of a RoBERTa model [22].

Sentence-BERT behaves differently in the training stages and the inference stages. During the training phase, it receives a batch of pairs of queries and documents relevant to each other. It adjusts its weight in order to make the representations of queries and documents similar if they are relevant to each other. This is a time consuming stage that requires a lot of computational resources. On the other hand, at inference time the model computes the representation of one query and compares it with the representation of all the documents. It is a fast stage. These phases will be explained in more detail in Sections 4.4 and 4.5.

The advantages of this approach are as follows: first it mixes a very fast architecture: representation-based models, with the use of the contextual embeddings from BERT. It is a low latency model that takes advantage of the accuracy of models that use the attention mechanism. Second, Sentence-BERT can be adapted to many different tasks like semantic similarity or

computing sentence embeddings. The idea is to take a pre-trained Sentence-BERT model and fine-tune it on a downstream task. In our case, we will fine-tune it for a text retrieval task.

3.3.2 Baseline: BM25

To compare the results of our proposed solution for initial retrieval (Sentence-BERT [35]), a commonly used baseline model will be implemented: BM25 [36]. It is an Exact Matching technique that ranks the documents based on the query terms that appear in them.

The BM25 score can be computed as stated in Equation 3.2. The term $tf(t,d)$ represents the term t frequency in document d, $df(t)$ is the number of documents that t appears in, N is the total number of documents, l_d is the length of document d and L is the average document length in the corpus. k_1 and b are free parameters.

$$BM25(q,d) = \sum_{t \in q \cap d} \log \frac{N - df(t) + 0.5}{df(t) + 0.5} \frac{tf(t,d) \cdot (k_1 + 1)}{tf(t,d) + k_1 \cdot \left(1 - b + b(\frac{l_d}{L})\right)} \tag{3.2}$$

BM25 is a method that does not include a BERT model and will serve as a baseline to compare the two approaches.

One example of this method can be shown by analyzing the BM25 scores of a query and three documents. The query is: "What do you enjoy to do?" and the three documents are: ["I really enjoy eating outside with my friends.", "I am a student of the Machine Learning master in KTH.", "The Covid-19 vaccinations have started."]. If we compute the BM25 score for each of the documents it will be 0 for the second and third documents as they do not share any terms with the query. And it would be 0.51 for the first one as they share the term "enjoy". In this last case the $tf('\text{enjoy}', d_1) = 1$ as it appears one time in the document, and the $df('\text{enjoy}') = 1$ as it only appears in one document from the corpus. The free parameters used in this example have been $k_1 = 1.5$ and $b = 0.75$.

3.4 Re-Ranker Stage

The re-ranker stage will be in charge of receiving the top-K candidate list of documents, analyzing the K candidate documents and the query, and outputting a final list of top-N ranked documents. The length of the final list of documents will be smaller or equal to the length of the candidate list: $N \leq K$.

The characteristics that this stage should fulfill are:

- It can be a slower or more complex method as it will only need to focus on the candidate list of documents and not the whole corpus.
- It should produce a more accurate ranking score for each document in the candidate list.
- It has to be a high accuracy BERT-based method.

The model chosen for this task is monoBERT [1]. It is a two-input classification BERT model with a linear layer on top. The input of this model is composed by the query, a document and the special tokens: [SEP] and [CLS]. [CLS] refers to classification and [SEP] refers to separation. In the input, first there is the [CLS] token, then the query, then the [SEP] token, then the document, and finally a [SEP] token again. This input sequence can be seen in Figure 3.4. For example, if the query is "What do you enjoy to do?" and the document is "I really enjoy eating outside with my friends", the input to the monoBERT would be: "[CLS] What do you enjoy to do? [SEP] I really enjoy eating outside with my friends [SEP]".

Once the input sequence is tokenized (using the BERT tokenizer explained in Section 2.1.2), it is forwarded to a BERT model, which creates contextual embeddings for all the tokens. Next, the model takes the contextual embedding of the [CLS] token and forwards it to a single linear layer that outputs a scalar value indicating the probability of the document being relevant to the query. These probabilities are used as the similarity score to re-rank the documents in the candidate list and output the top-N final list of ranked documents. After this final stage, the results of the top-N final ranked list will be presented to the engineers in charge of solving the specific TR that the query is about.

The monoBERT model is chosen as it provides a high accuracy as a re-ranker. The main drawback of monoBERT is latency, as explained before.

Our proposed method will be able to handle this by the limiting the length of the candidate list. Figure 3.4 shows a diagram of the re-ranker with each of its parts. Moreover, the monoBERT includes a BERT model in its architecture. The specific BERT model used will be ELECTRA [25] in most of the experiments, as this model provides a high accuracy.

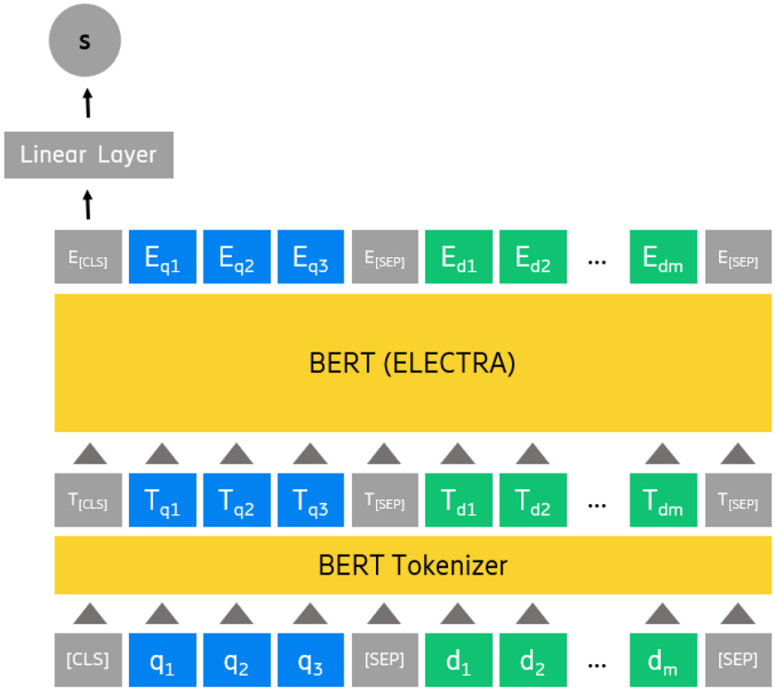


Figure 3.4: Diagram of the monoBERT model.

The monoBERT model also behaves differently in the training and inference stages. The training phase is a time-consuming stage that receives pairs of queries and documents and modifies its weights to classify correctly relevant pairs and non-relevant pairs. In the inference stage it receives a limited set of documents and a query and classifies the pairs formed by the query and each document in the set. This stage is less time-consuming than the training stage. More details on the training and inference will be given in Sections 4.4 and 4.5.

Chapter 4

Implementation

In this chapter the details of the implementation of the method are presented. First the data specifics will be explained. Then, the input and outputs of the methods will be specified. And finally, the training and inference phases will be discussed.

4.1 Data

The data used to train and test the method proposed will be Ericsson trouble-shooting data. It consist of finished TRs from the past year that are from the area of 4G and 5G radio networks. The language of these TRs is telecom-specific as well as company-specific and very different from a normal general-domain text such as Wikipedia. That is why it presents some challenges.

The general layout of a TR (as well as of bug reports in general) is that it has some structured fields. The main fields are:

- **Heading/Subject**: A short sentence that gives a summary overview description of the problem.
- **Observation**: A longer text describing the observed behavior of a probelm. Any useful information for its solution is provided (logs, configuration, Hardware (HW) versions, etc...). The observation text is typically guided by a template that divides the observation text in sections.
- **Answer**: A longer text that is filled in when the TR is solved, and the

solution is known. The answer contains the resolution given to the fault as well as the reason for the fault. The answer section is also guided by a template that divides the answer text in sections.

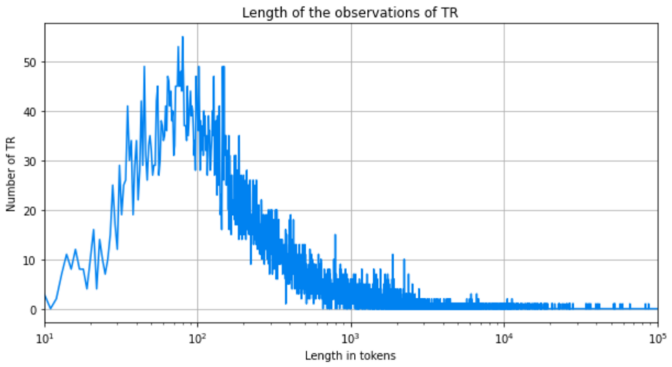
- **Faulty Product**: It is the specific code of the product on which the fault is reported. We propose creating a derived field from the product names which we call "Faulty Area". The "Faulty Products" will be mapped to a "Faulty Are"a which we will use in the input as it will be explained in Section 4.2.

All these sections will be extracted from the TRs and used in the input of our method.

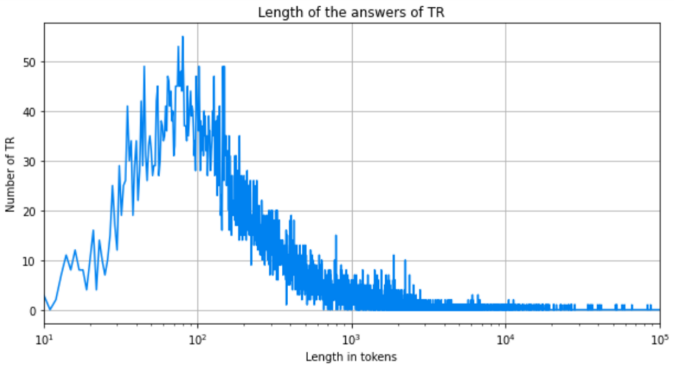
Furthermore, there are many challenges with the data, for example:

- Not all sections follow the same template, and not all organizations use the same templates (even within the same company).
- The length of each TR can differ, this means that the system must be able to cope with variable length input. As explained in Section 2.1.2, BERT models have a limitation in the length of the input of 512 tokens. The length of the two main sections in the dataset (observation and answer) after being tokenized can be seen in Figure 4.1. As shown, most of the TRs have an observation and an answer length of less that 512 tokens. The ones that are longer will be truncated.
- The TRs might have lots of non-textual data such as punctuation, links, software code, configuration information, machine generated logging information in various non-standardized formats. This information will need to be removed in the pre-processing stage.
- The language is not only domain-specific, but also company-specific, containing:
  - Company-specific product names.
  - Variable names specific to company products.
  - Company-specific nomenclature and abbreviations.
  - Adresses.

To emphasize the challenges that the data presents, an example of an observation of a TR can be seen bellow in Listing 4.1. It contains, acronyms,



(a): Length in BERT tokens of the observation section of the TRs.



(b): Length in BERT tokens of the answer section of the TRs.

Figure 4.1: Length of the TRs.

numerical data and punctuation signs, as well as company-specific language. This type of data will be pre-processed in the first stage of the method, as explained in Section 3.2.

```
1.1 Summary of the trouble
-------------------------------------------------
A crash in a node has been detected during a
RCC test.

1.2 Observations of the impact
-------------------------------------------------
The crashes was produced during a process
related to
RCC: 0x20050482
Time: 17-03-20 13:49:02
```

Listing 4.1: Example of an observation section of a TR.

4.2 Input and Output

The inputs of the text ranking system are the query and the corpus of documents as showed in Figure 3.1.

In this implementation, the corpus of documents will correspond to the corpus of past answers from finished TRs. The query will be formed by concatenating different sections of the TRs presented in Section 4.1. The main one will be the observation section of the TR, the other will be the Heading/Subject and the last one will be the Faulty Area, which is a synthetic field. A diagram of the input can be seen in Figure 4.2.

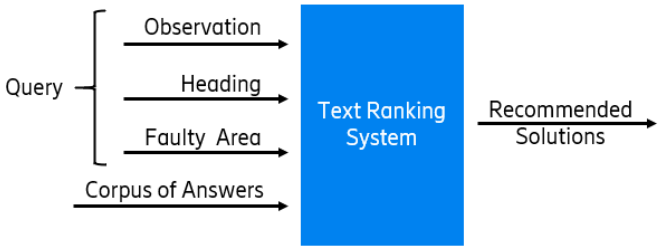


Figure 4.2: Diagram of the input to the text ranking system.

The Faulty Area is a synthetization of the Faulty Product which includes different products. By studying historical TRs, we can create a set of Faulty Areas, then we can map the products to these Faulty Areas. That way, the hundreds of products are reduced to a few areas which can add extra valuable information to the query. They can be viewed as an error area tag.

Specifically, the query will be a concatenation of different fields. Two different ways of forming the query will be compared in the experiments:

(i) We will concatenate the **Heading** and the **Observation**.

(ii) We will concatenate the **Faulty Area**, the **Heading** and the **Observation**.

By comparing these two approaches we will be able to see the added value of the Faulty Area in the performance.

Given a new TR that is input into the system, the output is a top-N ranked list of answers relevant to the TR. The list will be given to the support engineers in charge of solving the TR.

4.3 Candidate list

As explained in Section 3.3, the output of the initial retrieval stage is the top-K candidate list of documents. This list can be created in different ways. For example, one could use a threshold score and include in the list all the answers that had a score above this threshold. In this implementation we will use a candidate list with a fixed size of K. This parameter will be tuned by performing experiments with different values of K and comparing the latency of the method with the accuracy.

State-of-the-art multi-stage methods, explained in Section 2.1.3, usually use a candidate list of 100 to 1000 documents but this amount of documents increases the latency of the method a lot. Our aim is to reduce it as much as possible while keeping the performance acceptable and that is why the candidate list will be formed by less than 100 documents.

4.4 Training

Both the initial retriever and the re-ranker need training to achieve good performance on domain- and company-specific data. Each stage is trained separately as each model requires a different type of training.

Both stages contain a BERT model in their structure. Sentence-BERT [35] is composed by a BERT model and a pooling layer (as seen in Figure 3.3b) and monoBERT is composed by a BERT model and a linear layer (as seen in Figure 3.4). The BERT models inside those architectures have been pre-trained. The pre-training is done with general-domain data using the objectives explained in Section 2.1.2.

Then, the full structures of the Sentence-BERT and monoBERT are trained. The training is done in a similar task to ours (text ranking) but using general-domain data: the Microsoft Machine Reading Comprehension (MSMARCO) dataset [38]. This dataset includes search queries and passages from a search engine. Finally, both models are fine-tuned for our task (domain-specific text ranking) and each stage is be trained separately.

In the first stage (Sentence-BERT [35]) we take a model that has already been trained using the MSMARCO dataset [38] and can be found in the Hugging Face open-source transformers library [6]. Then, we fine-tuned it to work with telecom domain-specific data using our training set of troubleshooting

data. We have used a training set composed of queries and answer pairs, which are input into the model in batches. The loss used is the Multiple Negative Ranking Loss [39]. The model is trained for eight epochs, using a learning rate of $6 \cdot 10^{-5}$ with a linear warm up as shown in Table 4.1.

In the second stage (monoBERT [1]), we take a model that has been trained using the MSMARCO dataset [38] and can be found in the Hugging Face open-source transformers library [6]. It is fine-tuned using the training set of our troubleshooting data with negative samples. It is composed queries and answer pairs, as well as queries and non-relevant answer pairs, which are input into the model in batches. The loss used is the Cross-Entropy Loss. The model is trained for four epochs using a learning rate of $2 \cdot 10^{-5}$ and a linear warm up as shown in Table 4.1.

	Loss	**L. Rate**	**Epochs**	**Warm-up**
Initial Retrieval	Multiple Negative Ranking	$6 \cdot 10^{-5}$	8	Linear
Re-Ranking	Cross-Entropy	$2 \cdot 10^{-5}$	4	Linear

Table 4.1: Parameters of the training.

A validation set composed by 15% of the data points has been used in order to do hyper-parameter tuning to find the configuration with the highest accuracy.

4.5 Inference

Inference time corresponds to the time when the model receives a query and needs to output a ranked list of documents. The model needs to already be trained in this stage.

The proposed method at inference time is as follows: a fault is detected by a customer or during internal testing and a TR is submitted. From the TR, the observation, the heading and the Faulty Product are extracted. The Faulty Product is then mapped to the corresponding Faulty Area. Then, we then concatenate these fields to form the query.

In order to reduce the latency of the process, in the initial retrieval an already pre-computed representation of the answers in the corpus is used.

That way, at inference time the only representation needed to compute is the representation of the new query.

Once the representation of the new query is computed, the initial retriever computes its similarity with the different answers, and a candidate list of top-K answers is generated for the next phase.

In the re-ranker stage, the top-K candidate list is received. The re-ranker processes the query and the K answers and outputs a final ranked list. This list is used to recommend N possible answers to an engineer that has received a new TR. The number of proposed solutions at the end of the system is a design parameter that is equal or smaller to the candidate list length $N \leq K$. This number needs to be a reasonable number of answers to show to a support engineer, one example is $N = 5$.

At inference time, it is important that we limit the computations of the re-ranker as much as possible, as it is a time-consuming stage. Having an initial retrieval that outputs a candidate list allows this to happen, as the re-ranker only processes K answers instead of whole corpus.

If M is the total size of the corpus of answers then we need a candidate list that has a length $K \ll M$. That way, by having the two stages and pre-saved representations of the M answers, we only need to do a forward pass through a BERT model one time (at initial retrieval for the query) and K times (at re-ranker for each candidate), a total of $1+K$. If we did not use an initial retrieval stage the number of forward passes through BERT would be M which is much greater than $1 + K$.

We cannot pre-save the calculations of the second stage as the input for a two-input classification BERT model is: query + answer_1, query + answer_2, ... , query + answer_K as shown in Figure 3.4. While at initial retrieval the query and answers are input separately.

Chapter 5

Results and Discussion

In this chapter, the experiments and results of each of the stages and the whole model will be explained.

The proposed method has been tested using the dataset of troubleshooting reports. Therefore, for each query in the dataset, we know the correct answer and we can check if this answer is placed in a high position in the resulting recommended list. The metrics used to evaluate the method are the Recall@K, the MRR and the nDCG. In our setup Precision@K is not a valid metric for performance evaluation as we have only one correct answer for each query. All of those metrics are presented in Section 2.1.4.

We will present the results of each of the stages, the results of the latency measures, and finally we will evaluate how well our model can recognize similar TRs. The experiments will be performed on the test set of the troubleshooting data, which contains 15% of all data points.

5.1 Initial Retrieval Results

As explained in Section 3.3, for the initial retrieval stage the BERT-based model used has been Sentence-BERT [35]. It is composed by a BERT model and a pooling layer.

Three main experiments have been performed in this stage: the first one compares three BERT models in Sentence-BERT: DistilBERT [24], RoBERTa [22] and DistilRoBERTa [24]. The second one evaluates the how much value the Faulty Area adds to the query. And the third one compares the BERT-

based model with a baseline approach: BM25 [36]. To evaluate them we will use the measure of Recall@K, explained in Section 2.1.4.

Comparison of different BERT models in Sentence-BERT

The first experiment is a comparison between three different BERT models. As explained in Section 3.3, Sentence-BERT is composed by a BERT model and a pooling layer. This experiment shows the differences in performance if we use different types of BERT models: DistilBERT [24], RoBERTa [22] and DistilRoBERTa [24]. The difference between this models is stated in Section 2.1.2. The results of the experiment can be seen in Table 5.1. The difference in performance is minor but the model that performs best is DistilRoBERTa. For that reason it has been the preferred model for all the experiments.

BERT model	DistilBERT	RoBERTa	DistilRoBERTa
Recall@1	27.2%	26.5%	28.3%
Recall@3	39.3%	37.8%	39.7%
Recall@5	45.8%	44.0%	46.2%
Recall@10	54.4%	53.6%	55.2%
Recall@15	59.4%	58.8%	60.5%

Table 5.1: Comparison of different BERT models in Sentence-BERT.

Added value of the Faulty Area to the query

The second experiment performed in the initial retrieval is about the input of the model. As explained in Section 4.2, we want to see the added value of the Faulty Area to the query. For that, we want to investigate the performance of our first stage in case of whether we use the Faulty Area in the query or not. By seeing the difference in the results we will determine if value is added or not.

The results of this experiment can be seen in Table 5.2. To evaluate performance of the initial retrieval stage the metric used has been Recall@K. Depending on what information we input the model as the query, two results are stated. The first approach uses the heading and the observation as the

query, and the second approach uses the same fields as the first one but also uses the Faulty Area at the beginning. As it can be seen, the results of adding the Faulty Area in the query are significantly better.

Query	Heading + Observation	Faulty Area + Heading + Observation
Recall@1	28.3%	30.2%
Recall@3	39.7%	43.1%
Recall@5	46.2%	49.0%
Recall@10	55.2%	58.2%
Recall@15	60.5%	64.0%

Table 5.2: Results of Sentence-BERT using different information in the query.

Comparison of Sentence-BERT with a baseline model

We have also compared the results of our proposed solution (Sentence-BERT [35]) with a baseline model commonly used in initial retrieval tasks: BM25 [36]. With this comparison we can evaluate the difference of a BERT-based approach with an Exact Term approach.

The results of this comparison can be seen in Table 5.3. The score of each document has been computed using the Equation 3.2 presented in Section 3.3.2. The values of the free parameters used are $k_1 = 1.5$ and $b = 0.75$. As showed in the table, there is a big difference between using an Exact Matching technique compared to a BERT-based technique.

5.2 Re-Ranker Results

As explained in Section 3.4, for the re-ranker stage, the model used has been monoBERT [1]. It is a a two-input classification BERT model with a linear layer on top.

Four experiments have been performed in this stage: the first experiment compares different BERT models in the monoBERT to see which one performs

Initial Retriever	BM25	Sentence-BERT
Recall@1	18.2%	30.2%
Recall@3	23.7%	43.1%
Recall@5	26.6%	49.0%
Recall@10	31.0%	58.2%
Recall@15	33.3%	64.0%

Table 5.3: Comparison of the results of an Exact Matching technique (BM25) and a BERT-based technique (Sentence-BERT) for the initial retrieval.

better. The second experiment compares the results of the initial retrieval with the results of the multi stage model (initial retrieval and re-ranker) to see how much the performance improves by using a re-ranker stage. The third experiment shows an example of an ensemble of two monoBERT models. These experiments have been performed on the test set of the troubleshooting data. And the forth experiment is a cross-validation of the results of the initial retrieval and the whole model to check the generalization of our model. To evaluate them we will use the Recall@K measure as well as the MRR and the nDCG, all explained in Section 2.1.4.

Comparison between different BERT models

In the first experiment there is a comparison between two different BERT models: ELECTRA [25] and DistilRoBERTa [24]. As explained in Section 3.4, monoBERT [1] is composed by a BERT model and a linear layer on top. This experiment shows the differences in performance if we use different types of BERT models inside the monoBERT. The results can be seen in Table 5.4. It is very clear that ELECTRA has a better performance than DistilRoBERTa and that is why, for the following experiments, the preferred model in this method is ELECTRA. DistilRoBERTa is a less complex model (as explained in Section 2.1.2), but in this stage we are more interested in effectiveness rather than in efficiency.

BERT models	ELECTRA	DistilRoBERTa
Recall@1	36.6%	34.1%
Recall@3	48.5%	47.3%
Recall@5	53.5%	52.3%
Recall@10	59.8%	59.6%
Recall@15	64.0%	64.0%
MRR	0.44	0.42

Table 5.4: Comparison between different BERT models in monoBERT [1].

Comparison of the initial retrieval and the multi stage model

This experiment compares the performance of a single-stage model (only using the initial retriever) and a multi-satge model (using the initial retriever and re-ranker). The results of this experiment can be seen in Table 5.5. As explained, the re-ranker receives a candidate list of K documents (for this experiments $K = 15$), then it will re-rank those candidates more accurately so the correct answer climbs to the top of the list. This improvement can be seen in the Table 5.5 as the MRR and nDCG get higher, and the recall at small K improves. This means that we are retrieving the correct answer with the initial retriever and that the re-ranker is placing it at top positions.

	Initial Retrieval	In. Ret. + Re-Ranker
Recall@1	30.2%	36.6%
Recall@3	43.1%	48.5%
Recall@5	49.0%	53.5%
Recall@10	58.2%	59.8%
Recall@15	64.0%	64.0%
MRR	0.39	0.44
nDCG	0.44	0.48

Table 5.5: Results of the re-ranker compared to the initial retrieval.

Combination of two monoBERT models

In this experiment, two monoBERT [1] models are combined in an ensemble in order to see if together they improve the performance of the text ranking system. The first monoBERT model uses ELECTRA [25] as its BERT model, and the second monoBERT model uses DistilRoBERTa [24] as the BERT model. The scores that each model outputs are combined linearly by doing an mean operation, so the score of each monoBERT in the ensemble has the same importance. The results can be seen in Table 5.6, if we compare these results with Table 5.4 (using a single monoBERT model) we can see that the improvement is not significant enough while requiring more computational resources.

	Ensemble of two monoBERT models
Recall@1	37.1%
Recall@3	48.6%
Recall@5	54.0%
Recall@10	60.3%
Recall@15	64.0%
MRR	0.45

Table 5.6: Results of an ensemble of two monoBERT models [1].

Cross-validation of the results of initial retrieval and the multi stage model

This experiment is done in order to see if the results are generalizable. For that, a cross-validation is performed, in particular, a k-fold cross-validation. A k-fold cross-validation consists of dividing the dataset into k equal parts. Then, the training and testing is repeated k times, and each time one of the parts is used as the test set and the remaining are used as the training set. The results of each of the testings are averaged.

For this implementation we have used $k = 5$, and the results can be seen in Table 5.7. We can see that the results after the cross-validation are similar to the ones in Table 5.5. The performance is mantained and the

improvements between using only an initial retrieval stage and using a multi-stage architecture (initial retrieval and re-ranker) are also asserted.

	Initial Retrieval	**In. Ret. + Re-Ranker**
Recall@1	28.5%	32.6%
Recall@3	40.9%	46.2%
Recall@5	46.9%	51.1%
Recall@10	55.6%	57.5%
Recall@15	61.4%	61.4%
MRR	0.37	0.41
nDCG	0.42	0.46

Table 5.7: Results of the re-ranker compared to the initial retrieval.

5.3 Latency Results

We are interested in the latency of the model, for that, some experiments will be performed. The first one will be a comparison between the latency of the the two stages (initial retrieval and re-ranker). The second one will compare the latency of the model with the accuracy when using different lengths in the candidate list of documents. All the results have been performed on the test set of the troubleshooting dataset and using an NVIDIA Tesla T4.

Latency between the two stages

The latency of the different stages is analyzed by computing the average time it takes the system to find the top relevant answers to a query in each of the stages. We have used a candidate list in-between stages was of 15 documents. The representations of the corpus of answers have been pre-computed so that at initial retrieval the only representation needed to compute is the one from the query.

Table 5.8 shows the results of this experiment. As we can see, it takes 28 ms on average for the model to output a candidate list and 550 ms on average

for the re-ranker to output the final list of ranked answers. As explained in Section 4.5, if we just used the re-ranker without the initial retrieval (the re-ranker needs to process all documents instead of just the top K), the latency of the model would increase to minutes just for one query. The latency of the re-ranker increases proportionally to the length of the candidate list. By keeping the candidate list small, we are able to maintain this low latency.

Also, if we compare the two initial retrieval approaches we can see that using a non BERT-based approach does not contribute to less latency.

Stage	**Initial Retrieval**		**Re-Ranker**
Model	**Sentence-BERT**	**BM25**	**monoBERT**
milliseconds/query	28	125	550

Table 5.8: Latency of the two stages. The candidate list was of 15 documents.

Latency vs Accuracy using different length in the candidate list

In this experiment, the latency results have been compared using a different number of documents in the candidate list. We will compare a measure of accuracy like MRR with the latency of the whole model (initial retrieval and re-ranking) to see how much those values change if we use a different length of documents in the candidate list.

The results of this experiment can be seen in Table 5.9. As stated, the value $K = 15$ is optimal as it keeps the latency low while maintaining a good performance. For larger values, the increase in accuracy is very minor compared to the increase in latency, which goes from 0.6 seconds with $K = 15$ to 4 seconds with $K = 100$.

Candidate list length	$K = 15$	$K = 50$	$K = 100$
milliseconds/query	578	1940	3820
MRR	0.44	0.455	0.46

Table 5.9: Latency and accuracy of the method by using different length of candidate lists.

5.4 Consistency of Similar TRs

The last experiment consist of evaluating if our model is able to produce similar ranking lists for similar TRs.

There is a high probability that different customers individually raise different TRs for the same underlying problem. The problems are written in different words and length but talk about a similar or equal problem. If it is not identified early, different teams might work on the same underlying issue but on different TRs. As said, we want to check if the proposed method is able to recommend the same answers to similar TR even though they might be written using different words or described in other ways.

Of all the TRs, some engineers have identified TRs that could be considered similar. As we have the information of which TRs are similar to each other, we can produce the ranked lists of each TR and compare them in order to see if they are similar.

In the first experiment, we have computed the ranked lists for all the TRs that have "duplicates" and we have divide them between the ones were we perform well (the correct answer is retrieved in the top-15 list) and the once where the model fails. If we only pick the TRs where the model performs well and we analyze their ranking lists we can see that the correct answer is ranked at a similar position in 70% of them.

Another experiment performed has been to identify duplicates by checking which answer they rank at the top of the list regardless if it is the correct one or not. If we take all TRs that have one or more similar TRs in our test set, we can check if the answer they rank on top is the same as their "duplicates". This strategy is a possible method to detect similar TRs. Of all the TRs that have similar TRs in the test set we are able to identify at least one duplicate in 40% of the samples.

We have to keep in mind that we have not given the model any indication that there might be similar TRs. So, if the model can find similar TRs without the explicit training, it means that it has learned the domain-specific company language and can make inferences about it.

5.5 Discussion

The results presented suggest that we are able to recommend correct solutions to TRs. To show that, in this section, the results of all the experiments will be discussed.

In Section 5.1 we present the results of the initial retrieval using different fields in the query, comparing a baseline approach with a BERT-based aproach and using different BERT models. To evaluate those, we are interested in the recall. The measure of Recall@K indicates in which percentage of the queries from the test set we are able to retrieve the correct answer. Therefore, with a candidate list of 15 documents we are retrieving the correct answer in 64.0% of the queries using Sentence-BERT (Table 5.3).

The results in Table 5.2 suggest that as more information is added to the query, the best it can retrieve the solution. This effect can be seen in the difference between Recall@15, where if the Faulty Area is not used the value drops four points. By adding a word in the beginning of the query, Sentence-BERT is able to understand the information and use it to better find a correct solution. An interesting extra analysis of the system could be to remove the observation from the query and see if, by only using the Faulty Area and the Heading, the model performs better. However, the TRs not always contain a heading or a faulty product but they always contain the observation and that is why this experiment has not been done.

Furthermore, the results of Sentence-BERT are compared with BM25 to emphasize the difference in results of using an Exact Matching technique compared to a BERT-based technique. The BERT-based technique is able to understand semantically the queries and the answers while an Exact Matching technique has the problem of the vocabulary mismatch explained in Section 2.1. This can be seen in Table 5.3 and it suggests that using a BERT-based model in initial stages of a ranking system improves the quality of results and enables us to forward less documents to the next (more complex) stage. This effect maintains the latency of the process low at inference time.

Moreover, the results in Table 5.1 show how the DistilRoBERTa [24] model achieves a better performance compared to other BERT models and why it is the preferred model for this method.

In Section 5.2 we are interested in seeing an improvement in the results from the first stage to the second. This is demonstrated by the change in the MRR from 0.39 to 0.44 in Table 5.5, it means that the correct answer has

climbed-up the ranking thanks to the second stage. We can say that the re-ranker fulfills its purpose of improving the candidate list. This effect can also be seen if we compare the values of Recall@1 between the two stages. In this section we also compare the results of different BERT models and an ensemble and show that the model with the highest performance is ELECTRA [25].

In addition, one of the metrics most interesting for the research question is the latency of the method. We want to keep it as low as possible, while maintaining a reasonable accuracy. With the results from Table 5.8, we can demonstrate that by using a candidate list of 15 answers we are able to keep the time it takes to the system to recommend answers to less that 1 second.

Also, by analyzing the performance of using different candidate list lengths (Table 5.9) we are able to demonstrate that using more documents in the candidate list does not improve the accuracy as much. However, the latency is incremented a lot. This is the main reason the choice of length $K = 15$ is made. A visualization of how the similarity scores change along the corpus could be useful as well for determining how long this list should be.

An interesting point to add to the discussion is to evaluate the strategy of truncation of the texts when they are longer than 512 tokens, which is the BERT limit explained in Section 2.1.2. By looking at Figures 4.1 we can see that approximately 20% of TRs will be truncated. An interesting analysis to do here as future work would be to compute the percentage of text that is truncate from the TRs on average to see if we are loosing a lot or few information and how much it affects the performance.

Finally, an important result is the consistency similar the duplicates. Section 5.4 suggest that we are recommending the same answers to similar queries. The information that there are similar queries has not been used to train the model so this is a consequence of using BERT-based approach and its contextual embeddings.

Chapter 6

Conclusions and Future work

In this chapter the conclusions of the master thesis will be presented, as well as the future work.

6.1 Conclusions

In this master thesis, an NLP method for log analysis to retrieve solutions for troubleshooting processes has been presented, developed and tested.

First, we have introduced the problem of the troubleshooting process and the need for automation of the analysis and correction phase to improve the customer experience. We have also covered the background and the exiting solutions. Next, the a multi-satge BERT-based text ranking model for retrieving solutions to TRs has been proposed. Finally, the model has been tested and its results have been discussed.

We have been able answer the research question of: *How can we use BERT-based approaches to find solutions for error reports in the Ericsson logs that improve BM25, the existing non-BERT baseline model at Ericsson? And can this model have a low latency?* The model proposed is able to recommend the best possible solutions to a new error report from Ericsson with a high accuracy. It is composed by a pre-processing stage and two main stages, both being BERT-based. It has majorly increased the performance of a baseline non-BERT model like BM25.

The main conclusion drawn from this work are:

- Using a BERT-based approach instead of simpler techniques like BM25 in the first stage of initial retrieval improves the performance of the text ranking system.
- Creating a very good small candidate list (such as top-15) is the key to reducing the latency of the model, as the time-consuming stage (re-ranker) only need to process and re-rank this small list.
- Including a re-ranker in the architecture allows us to improve the list and increase the accuracy from the first stage to the second. We are able to recommend the best quality ranked lists to the support engineers.
- Using as much information as possible in the query helps the method produce a better list of recommended solutions. When we use the Faulty Area in the query the results are significantly better and we can say that it adds a lot of value.
- The model is consistent as it is able to recommend the same answers to similar TRs written in different words.

To sum up, the work of the master thesis has been able to investigate a problem, propose a solution that answers the research question, and provide the experiments and results needed to validate the solution.

6.2 Sustainability and ethics

Many NLP models raise ethical and sustainability questions. One of the main examples is the GPT-3 OpenAI model [40], a generative model that uses the transformers architecture [5]. However, the project has used text ranking BERT-based models, which do not raise the same amount of ethical concerns. However, they may still have biases that need to be accounted.

Also, it is important to add that the training of large models can be considered as not sustainable, that is why for this project only a fine-tuning of the models has been done and already pre-trained models have been taken from open-source repositories.

6.3 Future Work

The work developed in this master thesis is a good baseline for a BERT-based system for retrieving solutions for an error report. Some future work can be done in order to build upon the proposed method.

One possible track to take would be to improve the training of the BERT models. As explained in Section 2.1.2, the BERT models go through an extensive pre-training with general-domain data. Afterwards, they can be trained and fine-tuned for a specific task, like text ranking. One possibility to improve the results of the model would be to pre-train from scratch a BERT model using telecom-domain data instead of the general-domain data. This could improve the understanding of the TRs which usually contain a lot of telecom language.

Another possible improvement is the refining of the pre-processing. Even though the architecture has a pre-processing stage with many components, this step could be expanded to try to remove any non-textual information that may have not been removed by our cleaning modules. The observation text of TRs, usually contains many tables and code text that, sometimes, avoids the cleaning and introduces noise to the data. A better and meticulous treatment of this text is a possible future track.

Finally, a possible improvement of the architecture would be to re-think the re-ranker in terms of the latency. The proposed design is composed only by one model (monoBERT [1]), this choice has been made trying to reduce the latency at the inference time as much as possible. If we could be more flexible with the latency constrains, we could rethink the second stage and use an ensemble of different re-rankers in order to improve the accuracy and performance of the method.

References

[1] R. Nogueira, W. Yang, K. Cho, and J. Lin, "Multi-stage document ranking with bert," *arXiv preprint arXiv:1910.14424*, 2019.

[2] L. Jonsson, *Machine Learning-Based Bug Handling in Large-Scale Software Development*. Linköping University Electronic Press, 2018, vol. 1936.

[3] W. E. Wong, R. Gao, Y. Li, R. Abreu, and F. Wotawa, "A survey on software fault localization," *IEEE Transactions on Software Engineering*, vol. 42, no. 8, pp. 707–740, 2016.

[4] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[5] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," *arXiv preprint arXiv:1706.03762*, 2017.

[6] T. Wolf, L. Debut, V. Sanh, J. Chaumond, C. Delangue, A. Moi, P. Cistac, T. Rault, R. Louf, M. Funtowicz, J. Davison, S. Shleifer, P. von Platen, C. Ma, Y. Jernite, J. Plu, C. Xu, T. L. Scao, S. Gugger, M. Drame, Q. Lhoest, and A. M. Rush, "Transformers: State-of-the-art natural language processing," in *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*. Online: Association for Computational Linguistics, oct 2020, pp. 38–45. [Online]. Available: https://www.aclweb.org/anthology/2020.emnlp-demos.6

[7] C. Buckley, "Implementation of the smart information retrieval system," Cornell University, Tech. Rep., 1985.

[8] E. M. Voorhees, "Overview of the trec 2001 question answering track," in *In Proceedings of the Tenth Text REtrieval Conference (TREC*. Citeseer, 2001.

[9] I. Srba and M. Bielikova, "A comprehensive survey and classification of approaches for community question answering," *ACM Transactions on the Web (TWEB)*, vol. 10, no. 3, pp. 1–63, 2016.

[10] J. Lin, A. Roegiest, L. Tan, R. McCreadie, E. M. Voorhees, and F. Diaz, "Overview of the trec 2016 real-time summarization track." in *TREC*, 2016.

[11] I. Soboroff, S. Huang, and D. Harman, "Trec 2018 news track overview." in *TREC*, 2018.

[12] D. Harman *et al.*, "Information retrieval: the early years," *Foundations and Trends® in Information Retrieval*, vol. 13, no. 5, pp. 425–577, 2019.

[13] G. W. Furnas, T. K. Landauer, L. M. Gomez, and S. T. Dumais, "The vocabulary problem in human-system communication," *Communications of the ACM*, vol. 30, no. 11, pp. 964–971, 1987.

[14] P.-S. Huang, X. He, J. Gao, L. Deng, A. Acero, and L. Heck, "Learning deep structured semantic models for web search using clickthrough data," in *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, 2013, pp. 2333–2338.

[15] B. Mitra, E. Nalisnick, N. Craswell, and R. Caruana, "A dual embedding space model for document ranking," *arXiv preprint arXiv:1602.01137*, 2016.

[16] J. Guo, Y. Fan, Q. Ai, and W. B. Croft, "A deep relevance matching model for ad-hoc retrieval," in *Proceedings of the 25th ACM international on conference on information and knowledge management*, 2016, pp. 55–64.

[17] C. Xiong, Z. Dai, J. Callan, Z. Liu, and R. Power, "End-to-end neural ad-hoc ranking with kernel pooling," in *Proceedings of the 40th International ACM SIGIR conference on research and development in information retrieval*, 2017, pp. 55–64.

[18] J. Guo, Y. Fan, L. Pang, L. Yang, Q. Ai, H. Zamani, C. Wu, W. B. Croft, and X. Cheng, "A deep look into neural ranking models for information

retrieval," *Information Processing & Management*, vol. 57, no. 6, p. 102067, 2020.

[19] M. E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations," *arXiv preprint arXiv:1802.05365*, 2018.

[20] J. Howard and S. Ruder, "Universal language model fine-tuning for text classification," *arXiv preprint arXiv:1801.06146*, 2018.

[21] Y. Wu, M. Schuster, Z. Chen, Q. V. Le, M. Norouzi, W. Macherey, M. Krikun, Y. Cao, Q. Gao, K. Macherey *et al.*, "Google's neural machine translation system: Bridging the gap between human and machine translation," *arXiv preprint arXiv:1609.08144*, 2016.

[22] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "Roberta: A robustly optimized bert pretraining approach," *arXiv preprint arXiv:1907.11692*, 2019.

[23] Z. Lan, M. Chen, S. Goodman, K. Gimpel, P. Sharma, and R. Soricut, "Albert: A lite bert for self-supervised learning of language representations," *arXiv preprint arXiv:1909.11942*, 2019.

[24] V. Sanh, L. Debut, J. Chaumond, and T. Wolf, "Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter," *arXiv preprint arXiv:1910.01108*, 2019.

[25] K. Clark, M.-T. Luong, Q. V. Le, and C. D. Manning, "Electra: Pre-training text encoders as discriminators rather than generators," *arXiv preprint arXiv:2003.10555*, 2020.

[26] R. Nogueira and K. Cho, "Passage re-ranking with bert," *arXiv preprint arXiv:1901.04085*, 2019.

[27] S. MacAvaney, F. M. Nardini, R. Perego, N. Tonellotto, N. Goharian, and O. Frieder, "Expansion via prediction of importance with contextualization," in *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 2020, pp. 1573–1576.

[28] O. Khattab and M. Zaharia, "Colbert: Efficient and effective passage search via contextualized late interaction over bert," in *Proceedings*

of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval, 2020, pp. 39–48.

[29] L. Xiong, C. Xiong, Y. Li, K.-F. Tang, J. Liu, P. Bennett, J. Ahmed, and A. Overwijk, "Approximate nearest neighbor negative contrastive learning for dense text retrieval," *arXiv preprint arXiv:2007.00808*, 2020.

[30] J. Jiang, W. Lu, J. Chen, Q. Lin, P. Zhao, Y. Kang, H. Zhang, Y. Xiong, F. Gao, Z. Xu *et al.*, "How to mitigate the incident? an effective troubleshooting guide recommendation technique for online service systems," in *Proceedings of the 28th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, 2020, pp. 1410–1420.

[31] N. Ferland, W. Sun, X. Fan, L. Yu, and J. Yang, "Automatically resolve trouble tickets with hybrid nlp," in *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 2020, pp. 1334–1340.

[32] Y. Wang, I.-C. Choi, and H. Liu, "Generalized ensemble model for document ranking in information retrieval," *arXiv preprint arXiv:1507.08586*, 2015.

[33] J. Xin, R. Tang, J. Lee, Y. Yu, and J. Lin, "Deebert: Dynamic early exiting for accelerating bert inference," *arXiv preprint arXiv:2004.12993*, 2020.

[34] Spacy, "Language processing pipelines," https://spacy.io/usage/processing-pipelines, accessed 2021-05-12.

[35] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," *arXiv preprint arXiv:1908.10084*, 2019.

[36] S. Robertson and H. Zaragoza, *The probabilistic relevance framework: BM25 and beyond*. Now Publishers Inc, 2009.

[37] D. Chicco, "Siamese neural networks: An overview," *Artificial Neural Networks*, pp. 73–94, 2021.

[38] T. Nguyen, M. Rosenberg, X. Song, J. Gao, S. Tiwary, R. Majumder, and L. Deng, "Ms marco: A human generated machine reading comprehension dataset," in *CoCo@ NIPS*, 2016.

[39] M. Henderson, R. Al-Rfou, B. Strope, Y.-H. Sung, L. Lukács, R. Guo, S. Kumar, B. Miklos, and R. Kurzweil, "Efficient natural language response suggestion for smart reply," *arXiv preprint arXiv:1705.00652*, 2017.

[40] T. B. Brown, B. Mann, N. Ryder, M. Subbiah, J. Kaplan, P. Dhariwal, A. Neelakantan, P. Shyam, G. Sastry, A. Askell *et al.*, "Language models are few-shot learners," *arXiv preprint arXiv:2005.14165*, 2020.