BERTicsson: A Recommender System For Troubleshooting

Nuria Marzo I Grimalt[1], Serveh Shalmashi[1], Forough Yaghoubi[1], Leif Jonsson[1], Amir H. Payberah[2]
[1]Ericsson AB, Stockholm, Sweden
[2]KTH Royal Institute of Technology, Stockholm, Sweden
{nuria.marzo.i.grimalt,serveh.shalmashi,forough.yaghoubi,leif.jonsson}@ericsson.com
payberah@kth.se

Abstract

Troubleshooting in the telecommunication industry is a time-consuming task, often involving text understanding, which is challenging to automate due to its domain/company-specific features. This work aims to build a model to retrieve solutions for newly reported problems in automated and quick ways. To this end, we present BERTicsson, a BERT-based model that uses two main stages for (i) retrieving a shortlist of candidate answers for new problems and (ii) raking them accordingly. We study the performance of BERTicsson using Ericsson's troubleshooting dataset and show that it significantly improves the accuracy of the recommended answers compared to non-BERT models, such as BM25.

Introduction

Troubleshooting in modern telecommunication systems is a slow process (Wong et al. 2016). Many failures lead to service downtime or other forms of harm to the customer experience; thus, they must be quickly detected, categorized, and resolved. Usually, when engineers observe a fault in a running system that they cannot solve on site, they create a *Trouble Report (TR)* to track the information regarding the detection, characteristics, and an eventual resolution of the problem (Jonsson 2018).

Figure 1 shows a typical troubleshooting process that has six steps. The process starts by detecting a problem in step 1, followed by reporting it as a TR during steps 2 and 3. Then, this TR is analyzed and corrected in steps 4 and 5, and finally, the proposed solution is verified by engineers in step 6. Steps 4 and 5 often involve understanding textual data in TRs, which can be challenging to automate due to its domain-specific and company-specific features. Moreover, the text may contain many abbreviations, typos, tables, and numerical data, making the process more difficult. However, with today's impressive development in machine learning, specifically in Natural Language Processing (NLP), we can shorten the process by analyzing historical TR data and infer a solution to new problems faster and more accurately.

In this work, we tackle the problem of retrieving solutions in the troubleshooting process in an automated way using NLP models. To this end, we present BERTicsson, a multi-stage model to automate steps 4 and 5 of the process in Figure 1. BERTicsson is a recommender and text-ranking system based on BERT (Devlin et al. 2018) that receives a history of TRs (i.e., old problems and their solutions) and newly reported problems as input and returns a ranked list of recommended solutions to each problem. Both TRs and new problems description are given in natural language.

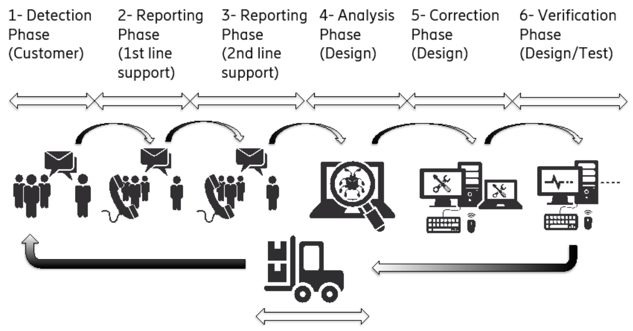

1- Detection Phase (Customer)
2- Reporting Phase (1st line support)
3- Reporting Phase (2nd line support)
4- Analysis Phase (Design)
5- Correction Phase (Design)
6- Verification Phase (Design/Test)

Figure 1: The steps of a troubleshooting process.

Upon receiving input data, BERTicsson initially cleans them at the *pre-processing* stage, and then, through the *Initial Retrieval (IR)* stage, it retrieves a candidate list of answers relevant to the problem. Finally, at the *Re-Ranker (RR)* stage, it ranks the candidate list provided by the IR stage concerning the problem. In the IR stage, BERTicsson uses Sentence-BERT (Reimers et al. 2019), a variant of BERT, to make a representation of the problems and answers, and in the RR stage, it takes advantage of monoBERT (Nogueira et al. 2019a), a two-input classification BERT model.

We use Ericsson's 4th Generation (4G) and 5th Generation (5G) TR dataset through the experiments and show that BERTicsson provides a high accuracy while keeping the latency low. We compare BERTicsson with BM25 (Robertson et al. 2009), a popular ranking model, and show that BERTicsson improves the Recall of the recommended answers by around 65%.

Background

We define the *text-ranking* problem as generating an ordered set of texts retrieved from a corpus of documents in response to a query for a particular task. This section reviews some of the pre-BERT and BERT techniques for text-ranking.

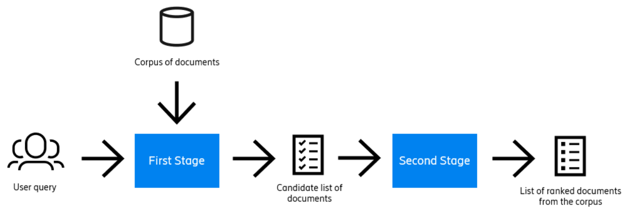

Corpus of documents
User query
First Stage
Candidate list of documents
Second Stage
List of ranked documents from the corpus

Figure 2: A multi-stage architecture.

Pre-BERT Text-Ranking Methods

Before the appearance of BERT (Devlin et al. 2018), the Exact Matching (EM) (Harman et al. 2019) and neural information retrieval techniques (Huang et al. 2013; Mitra et al. 2016; Guo et al. 2016; Xiong et al. 2017) have been the primary text-ranking methods. EM solutions mainly rely on the *term frequency* and *document frequency*, where the former shows the number of times that a term occurs and the latter shows in how many documents it appears. A query and a document have a high score if they both use the same terms. Indeed, this limits the algorithm's applicability and its performance in case of having vocabulary mismatch problem (Furnas et al. 1987), i.e., when the query and the documents use different words to refer to equivalent things.

The pre-BERT neural information retrieval methods are divided into two main architectures: *representation-based* approaches and *interaction-based* approaches. A representation-based model learns a dense vector representation of the query and the documents independently. Then, it computes the similarity between the representations using cosine similarity or inner product and ranks the documents accordingly for the given query (Huang et al. 2013; Mitra et al. 2016). On the other hand, an interaction-based model focuses on the interaction between each query's and document's terms, and a similarity matrix is created. This matrix undergoes further processing to extract a similarity value (Guo et al. 2016; Xiong et al. 2017).

BERT-based Text-Ranking Methods

BERT (Devlin et al. 2018) is a language embedding model that learns contextual representations of words in a sentence. BERT is pre-trained on a large corpus of text in an unsupervised setting, with two different learning objectives: Masked Language Modeling (MLM) and Next Sentence Prediction (NSP). In MLM, some words in the input sentence are hidden from the model, and the model should predict the original word of the masked token based on the context of the other non-masked words in the sentence. In NSP, BERT receives two sentences, A and B, as input, and it should predict whether sentence A is followed by sentence B.

Recently, several variations of BERT have appeared. For example, RoBERTa (Liu et al. 2019) removes the NSP pretraining task and uses dynamic masking by changing the masked tokens during the training epochs, ALBERT (Lan et al. 2019) reduces the number of parameters of the original BERT model by implementing the cross-layer parameter sharing, DistilBERT (Sanh et al. 2019) uses distillation to pre-train BERT, and ELECTRA (Clark et al. 2020) pre-

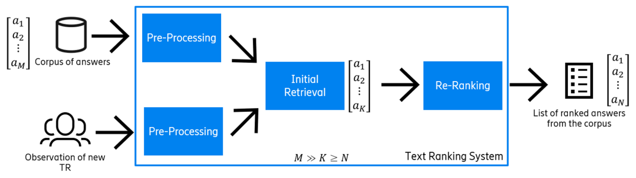


Figure 3: BERTicsson architecture.

trains a discriminator that has to distinguish if the sentences forwarded to the model have a replaced token or not.

The first attempt of using BERT for text-ranking adopts the BERT architecture to make a relevant score between the query and a document (Nogueira et al. 2019b). The input to this model is a query and a document, and the output is a contextual embedding of them, which is used for deriving the similarity score between them. BERT-based models, however, present some limitations, as they only accept sequences of less than 512 tokens, which limits the length of input text. They also present high complexity and are slow in text-ranking tasks.

Traditional text-ranking approaches have one-stage architecture. However, the state-of-the-art models follow a multistage architecture (Figure 2), usually formed by two main stages: (i) an Initial Retrieval (IR) stage that extracts a candidate set of text, and (ii) a Re-Ranker (RR) stage that creates a ranked list of the candidate set. The IR stage is fast, allows the model to discard easy candidates, and passes a smaller set of documents to the RR stage. The RR stage is slow, but as the input of documents is significantly smaller, it maintains low latency. The number of documents that pass from IR to RR can be defined in different ways, such as keeping a certain number of documents with the highest scores or keeping a fixed percentage of all documents.

BERTicsson

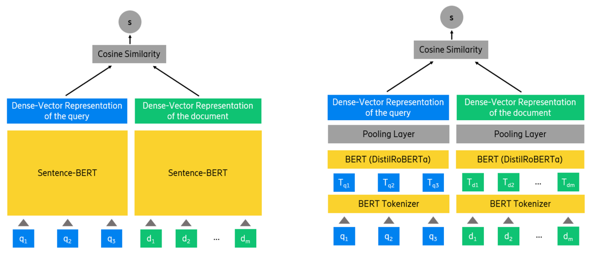
We aim to make a model for retrieving accurate solutions to newly reported problems in telecommunication with low latency. To this end, we present BERTicsson (Figure 3), a recommender and text-ranking model that is composed of three stages: (i) the pre-processing stage that cleans the input TRs (i.e., the new problems (queries) and the corpus of documents), (ii) the Initial Retrieval (IR) stage that retrieves a top-K candidate list of documents with answers relevant to the query, and (iii) the Re-Ranker (RR) stage that ranks the top-K candidate list provided by the IR and outputs the final list of ranked documents. In the rest of this section, we present these three stages in depth.

Pre-Processing Stage

The pre-processing stage prepares the data for the IR and RR stages. The input TR is a newly reported problem and corpus of documents containing telecom/company-specific language, and the output is the *query* and the documents ready to be given to the IR and RR stages.

The pre-processing stage has five steps:

1. *Tokenizing text*: it tokenizes input text using a customized tokenizer that recognizes company-specific and domain-specific language.



(a) The IR stage architecture. (b) Sentence-BERT architecture.

Figure 4: The IR stage.

2. *Detecting abbreviations*: it detects and tags abbreviations and acronyms.
3. *Replacing abbreviations*: it replaces the tagged telecom abbreviations and acronyms with the complete words. In the case of multiple suggestions for a given abbreviation, it picks the suggestion which is most related to radio networks, which have been manually labeled.
4. *Removing numerical data*: it removes any numerical tokens as they do not provide any helpful information to the model.
5. *Handling special tokens*: it removes extra spaces, new lines, and gaps between words as well as any punctuation signs.

Initial Retrieval Stage

The IR stage gets the pre-processed query and documents, and after analyzing them, creates a top-K candidate list of the most relevant documents (answers) to the query. The candidate list should have all relevant documents to the query (if possible); however, the order of the documents in the candidate list is not important at this stage, meaning that the documents with the most relevant answers do not need to be at the top positions of the list. The IR stage should have low latency to quickly manage a large amount of data. We use Sentence-BERT (Reimers et al. 2019) to build this stage. Sentence-BERT is a representation-based model that creates dense vectors Q and D as embeddings for the query q and the document d, respectively (Figure 4a). It then uses cosine similarity to compute the similarity between the Q and D as below:

$$S(Q, D) = \frac{Q \cdot D}{\|Q\| \cdot \|D\|} = \frac{\sum_i Q_i D_i}{\sqrt{\sum_i Q_i^2}\sqrt{\sum_i D_i^2}} \qquad (1)$$

Sentence-BERT internally uses a Siamese network (Chicco 2021) that consists of two neural networks with shared weights. Each of these two branches has two layers: a BERT layer and a mean pooling layer (Figure 4b). The Sentence-BERT receives two sentences as input: query q and document d. First, it tokenizes q and d and forwards the tokens to the BERT layer to create a contextual embedding of each token. Next, the BERT layer output is given to the mean pooling layer to create the fixed-size representation of each sentence. Once we have the representations of q and d, denoted by Q and D, respectively, we use Equation 1 to measure their similarity value. The similarity value will correspond to the ranking score and be used to create the top-K candidate list.

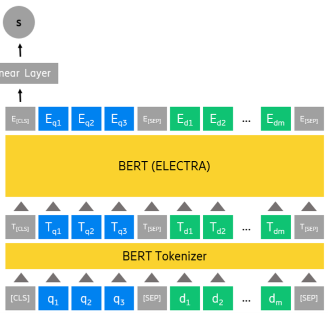

s
Linear Layer
E[CLS] Eq1 Eq2 Eq3 E[SEP] Ed1 Ed2 ... Edm E[SEP]
BERT (ELECTRA)
T[CLS] Tq1 Tq2 Tq3 T[SEP] Td1 Td2 ... Tdm T[SEP]
BERT Tokenizer
[CLS] q1 q2 q3 [SEP] d1 d2 ... dm [SEP]

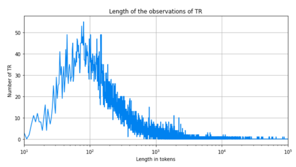
Figure 5: The RR stage and the monoBERT model.

We can use any of the BERT models in Sentence-BERT; however, we use DistilRoBERTa (Sanh et al. 2019) in our implementation. Sentence-BERT behaves differently in the training and inference phases. During the training phase, it receives a batch of pairs of relevant queries and documents, and it then adjusts its weight to represent queries and documents similar if they are relevant to each other. However, this is a time-consuming task; thus, after the training phase, we store the representation of all the documents (only the documents, not the queries). Therefore, at the inference phase, we only need to compute each query's representation and compare it with the stored representation of the documents, which is fast.

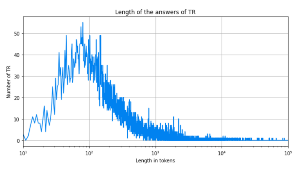
Re-Ranker Stage

The RR stage receives a query q and the top-K candidate documents relevant to q and returns the final list of top-N ranked documents, such that $N \leq K$ (Figure 3). In this stage, we use monoBERT (Nogueira et al. 2019a) (Figure 5). The input to monoBERT is composed of a query q and a document d, and the special tokens [SEP] and [CLS] in form of "[CLS]q[SEP]d[SEP]". Once monoBERT tokenizes the input sequence, it forwards them to the internal BERT to create the contextual embeddings for all the tokens. Next, it forwards the embedding of the [CLS] token to a single linear layer that outputs a scalar value indicating the probability of the document d being relevant to the query q.

We choose monoBERT for the RR stage due to its high accuracy. However, the main drawback of monoBERT is its latency. To overcome this issue, we limit the length of the candidate list passed through the IR stage (it just sends the top-K candidates of documents to the RR rather than the whole list of documents). Moreover, as Figure 5 shows, we use ELECTRA (Clark et al. 2020) inside monoBERT in our implementation. The training phase of RR is time-consuming, as monoBERT receives pairs of queries and documents from the whole corpus of the documents to update its weights to classify correctly relevant pairs and non-relevant pairs. However, monoBERT receives a query in the inference phase and only the top-K relevant candidates. Thus, it classifies the pairs formed by the query and each document in the candidate list.



(a) The Observation field.



(b) The Answer field.

Figure 6: Num. of tokens of the Observation and Answer field of the TRs.

Implementation

In this section, first, we describe the structure of the dataset (TRs) and the input and output of the model and then present the training and the inference process.

Data

To train and test BERTicsson we use the Ericsson troubleshooting dataset consisting of finished 4G and 5G radio networks TRs over the past year. The language of these TRs is telecom/company-specific, which is very different from a general-domain text. Generally, each TR in the dataset has the following fields:

- *Heading/Subject*: A short sentence that gives a summary overview description of the problem.
- *Observation*: A longer text describing the observed behavior of a problem, including any useful information for its solution (e.g., logs and configuration).
- *Answer*: A longer text that contains the resolution given to the fault as well as the reason for the fault.
- *Faulty Product*: It is a specific code of the product on which the fault is reported. In our implementation, we create an extra field, called *Faulty Area*, from the product name that is mapped to the Faulty Products. By studying historical TRs, we can create some Faulty Areas and map the products to these Faulty Areas. This way, we can reduce the hundreds of products to a few areas that provide extra valuable information to the query.

```
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
```

Figure 7: An example of the Observation field of a TR.

We extract these fields from TRs and apply further changes (as explained in the pre-processing section) to create the model input. Figure 7 shows an example of the Observation field of a TR. TRs can have different lengths. For example, Figure 6 shows the length of the Observation and Answer fields in the dataset after being tokenized. As explained in the Background section, BERT models have a limitation in the length of the input of 512 tokens, and as we see in Figure 6, most of the Observations and the Answers in TRs are less than 512 tokens. However, if they are longer than 512, we truncate them to give them to our model.

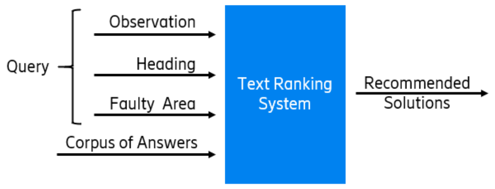


Figure 8: Diagram of the input to the text ranking system.

BERTicsson also receives TRs of reported new problems that we need to create queries based on them. As Figure 8 shows, we form a query by concatenating different fields of the TRs: the Observation, the Heading/Subject, and the Faulty Area, which is a synthetic field that we build from the Faulty Product.

Training

The IR and RR stages are supervised-learning models. We train these stages separately as each requires different training types. Both stages contain a BERT model in their structure, i.e., the IR stage has Sentence-BERT, and the RR stage uses monoBERT. In the IR stage, we take a model pre-trained using the MSMARCO dataset (Nguyen et al. 2016) that includes search queries and passages from a search engine. The model is available in the Hugging Face open-source transformers library (Wolf et al. 2020). Then, we fine-tune it to work with telecom/company-specific data. The training set is composed of the query and document pairs (q, d), which are input into the model in batches of m samples, $\{(q_i, d_i)\}_{i=1}^{m}$. After making the embedding of each query q and document d, denoted by Q and D, respectively, the model minimizes the negative log-likelihood for softmax normalized scores:

$$\mathcal{L}(Q, D) = -\frac{1}{m}\sum_{i=1}^{m}\left[S(Q_i, D_i) - \log\sum_{j=1, j\neq i}^{m} e^{S(Q_i, D_j)}\right] \tag{2}$$

where $S(Q_i, D_j)$ is the cosine similarity between the embedding of the query q_i and the answer d_j (Equation 1). We use the Multiple Negative Ranking loss function (Henderson et al. 2017), and train the model for eight epochs using a learning rate of $6 \cdot 10^{-5}$ with a linear warm-up.

In the RR stage, similar to the IR stage, we use a pre-trained model available in the Hugging Face library (Wolf et al. 2020), and fine-tune it for the ranking task. To make the RR training set, first, we need to define the *positive* and *negative* samples. For each query q_i in the training data, the document d_i is positive if it contains the correct solution for the problem in q_i. All the other documents d_j $(j \neq i)$ in the dataset are negative samples, i.e., they are not correct answers for q_i.

To create the RR training set, in addition to the positive document for each query q, we select three random negative samples. Thus, the training set for the RR is composed of a query q and its correct document pair, and q and three non-relevant documents pairs, which are input into the model in batches. Then, we compute a similarity score for each pair in the batch and use a binary classification to indicate if a pair of a query and a document is relevant or non-relevant. Finally, the labels (i.e., relevant or non-relevant) and the batch scores are given to a Binary Cross-Entropy loss function that is minimized during the training process. We train the model for four epochs using a learning rate of $2 \cdot 10^{-5}$ and a linear warm-up.

Inference

The inference phase corresponds to when the model receives a query and outputs a ranked list of documents. When a fault is detected, a TR is submitted, and we extract the Observation, the Heading, and the Faulty Product from it. The Faulty Product is then mapped to the corresponding Faulty Area. Then, we concatenate these fields to form the query (Figure 8). Note that we store the representation of all the documents in the corpus after the training phase. So, during the inference time, the IR stage only needs to compute the representation of the new query, which is a quick process.

Once the representation of the new query is computed, the IR stage computes its similarity with the different documents and generates a candidate list of top-K documents for the RR stage. When the RR stage receives the top-K candidate list and the query, it processes them and creates the final ranked list; the N recommended documents to a new TR. At the inference time, we must limit the computations of the RR as much as possible, as it is a time-consuming stage. The IR stage that outputs a candidate list allows this to happen, as the RR stage processes only K answers instead of the whole corpus. If M is the total size of the corpus of answers, we need a candidate list with a length of $K \ll M$. That way, by having the two stages and pre-saved representations of all the M documents, we only need to do one forward pass through a BERT model at IR for the query, and K times at RR for each candidate document, a total of $K + 1$. However, if we do not use the IR stage, the number of forwarding passes through BERT would be M for each query, which is much greater than $K + 1$.

Evaluation

In this section, we study the performance of BERTicsson. First, we introduce the metrics we use in the experiments and then study the performance of the IR and RR stages separately, and finally, we evaluate the performance of the whole model. We use the dataset described in the previous section to train the model and test the model using 15% of all data points. Since we test the model using the TR dataset, we know the correct document (answer) for each problem (query) in the dataset, and we can also check if this document is placed in a high position in the resulting recommended list. We assume a binary relevance between a query and a document, i.e., whether the document is relevant or not.

Table 1: Different BERT models in Sentence-BERT.

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%

Evaluation Metrics

We evaluate BERTicsson using three metrics: *Recall@K*, *Mean Reciprocal Rank (MRR)*, and *Normalised Discounted Cumulative Gain (nDCG)*. The Recall is the fraction of relevant documents for a query in the entire corpus retrieved in the ranked list. Recall@K is the Recall at a cutoff K. However, the Recall does not consider graded relevance or the positions in the ranking. The Reciprocal Rank (RR) captures the appearance of the first relevant document, which is the multiplicative inverse of the rank of the first correct document. For example, if the first relevant document appears at position 3, then RR is $1/3$. MRR is the average of RR of results for a set of queries. nDCG shows the usefulness of a document based on its position in the ranked list, meaning that the earlier a document appears in the ranked list, the more useful it is. These last two metrics will help us evaluate how well correct answers are placed in the ranking.

Initial Retrieval Stage Results

To evaluate the performance of the IR stage, we conduct four experiments: (i) comparing three BERT models in Sentence-BERT, (ii) evaluating the impact of including the Faulty Area to the query, (iii) comparing the performance of Sentence-BERT with BM25 (Robertson et al. 2009), and (iv) studying the relationship between the IR cosine similarity score and the ranked lists. In all these experiments, we use Recall@K as the evaluation metric.

Different BERT models in Sentence-BERT. Sentence-BERT is composed of a BERT model and a pooling layer. Here, we compare the performance of Sentence-BERT using different BERT models. In particular we consider DistilBERT (Sanh et al. 2019), RoBERTa (Liu et al. 2019), and DistilRoBERTa (Sanh et al. 2019). Table 1 shows the Recall@K of the IR stage using different BERT models in Sentence-BERT. As we see, the difference in performance is small; however, DistilRoBERTa performs slightly better than the others. Therefore, in the rest of the experiments, we use DistilRoBERTa in Sentence-BERT.

Having Faulty Area in the query. Here, we evaluate the performance of the IR stage in two cases: (i) making the query by concatenating the Heading and the Observation, and (ii) making the query by concatenating the Faulty Area, the Heading, and the Observation. These cases are applied both in the training and inference phase. Table 2 shows the results of this experiment, and as we see, including the Faulty Area in the query significantly improves the Re-

Table 2: Using different fields in the query.

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 3: BM25 vs. Sentence-BERT in the IR stage.

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%

call@K of the IR stage. It confirms that adding more information to the query (in form of the Faulty Area) helps the model to recognize which type of answers will work best for a query.

Sentence-BERT vs. BM25. In the third experiment, we compare the performance of the IR stage in making the candidate list in two cases of using Sentence-BERT (Reimers et al. 2019) and using BM25 (Robertson et al. 2009), which is a popular EM method. Here, we use the Okapi BM25 implementation. As we see in Table 3, Sentence-BERT outperforms BM25 by improving the Recall@1 by around 65% (from 18.2% by BM25 to 30.2% by BERTicsson).

The IR similarity scores and the ranked list. This experiment evaluates how the cosine similarity scores change along with the length of the top-K ranked list. We plot the mean and the standard deviation of all the queries in our test set in Figure 9. Here, the X-axis shows the length of the candidate list (i.e., K in top-K), and the Y-axis shows the similarity value. As we see, the similarity score decreases rapidly in the first positions in the ranked list, and it decreases more slowly from position 20. Given this result, we consider $K = 15$ is a reasonable length for the candidate list, so the IR stage sends the top-15 documents to the RR stage.

Re-Ranker Stage Results

We conduct three experiments to study the performance of the RR stage: (i) comparing different BERT models in monoBERT, (ii) studying the impact of the RR stage on improving the performance of the IR stage, and (iii) studying how the RR similarity scores change along with the ranked lists. To evaluate these, we use the Recall@K, the MRR, and the nDCG.

Different BERT models in monoBERT. In the RR stage, we use monoBERT (Nogueira et al. 2019a), and here, we

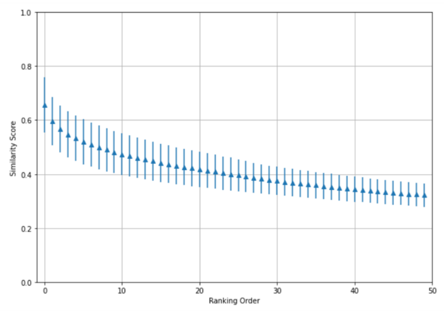


Figure 9: Mean and standard deviation of the cosine similarity scores in the IR along with the ranked list length.

Table 4: Different BERT models in monoBERT.

BERT models	ELECTRA	DistilRoBERTa	Ensemble Model
Recall@1	36.6%	34.1%	37.1%
Recall@3	48.5%	47.3%	48.6%
Recall@5	53.5%	52.3%	54.0%
Recall@10	59.8%	59.6%	60.3%
Recall@15	64.0%	64.0%	64.0%
MRR	0.44	0.42	0.45

study the performance of monoBERT, in case of using three different BERT models: ELECTRA (Clark et al. 2020), DistilRoBERTa (Sanh et al. 2019), and an ensemble of ELECTRA and DistilRoBERTa, where we combine the scores of these two models by computing their average. Table 4 shows the results, and as we see, ELECTRA has a better performance than DistilRoBERTa. Although DistilRoBERTa is a faster model compared to ELECTRA, we need a more accurate model in this stage. Therefore, in the rest of the experiments, we use ELECTRA in monoBERT. We also see that the ensemble model outperforms the other models; however, the improvement is not significant enough while requiring more computational resources.

The impact of the RR stage. The RR stage receives a top-K candidate list from the IR stage, and then it re-ranks them, such that the correct document climbs to the top of the list. Here, we study how the RR stage improves this ranking. To do so, we compare the position of the correct document in the candidate list in two cases: (i) after the IR stage and (ii) after the RR stage. In this experiment, we set $K = 15$. As we see in Table 5, the RR stage improves the MRR by 12% and nDCG by 9%, and increases the Recal@K (for small K); meaning that although we get the correct document from the IR stage, the RR stage improves the ranking by pushing the correct document to the top positions.

The RR similarity scores and the ranked list. Here, we study how the RR similarity scores change along with the ranked list. In Figure 10, we plot the mean and the standard deviation of all the queries in our test set. There are only 15 ranking orders in the Figure 10 as the candidate list for-

Table 5: The impact of RR stage.

	After the IR stage	After the RR stage
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

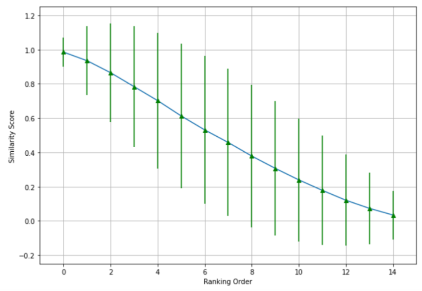


Figure 10: Mean and standard deviation of the similarity scores in the RR stage along the ranked list.

warded to the RR stage is the top-15 list. We need to decide how many documents we want to show as the top-N recommendations in output. Since the scores decrease uniformly and the variance is very high, we decided that a reasonable number of top-N recommended answers is $N = 5$, as it includes the top-5 highest similarity scores, and it is a reasonable number of recommendations to show to an engineer.

Latency Results

In this part, we evaluate the latency of BERTicsson by conducting two tests: (i) studying the latency of the IR and the RR stages and (ii) studying the impact of the candidate list length on the accuracy and the latency of the model.

Latency of the two stages. In this experiment, we study the latency of the IR and the RR stages in the inference time. During the inference, we have the pre-computed representations of the corpus of documents, so the IR stage only needs to compute the representation of the query. We compute the latency of each stage by averaging the time each takes to find the top-K ($K = 15$) relevant documents to a query.

As we see in Table 6, it takes $28ms$ on average for the IR stage to make a candidate list and $550ms$ on average for the RR stage to get the final list of ranked documents. If we only use the RR stage without the IR stage, then the RR stage needs to process all documents instead of just the top K; thus, the latency of the model would increase to minutes just for one query. The latency of the RR increases proportionally to the length of the candidate list. By keeping the candidate list small, we can maintain this low latency. Moreover, if we compare the two IR approaches (i.e., Sentence-BERT and BM25), we see that BM25 does not reduce latency as good as Sentence-BERT.

Table 6: Latency of the two stages. The size of the candidate list is of 15 documents.

Stage	**The IR stage**		**The RR stage**
Model	**Sentence-BERT**	**BM25**	**monoBERT**
milliseconds/query	28	125	550

Table 7: Latency and accuracy of the method by using different length of candidate lists.

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

Latency vs. accuracy. In this experiment, we study the latency of BERTicsson for different candidate list sizes (i.e., different K in top-K). We compare MRR, as a measure of accuracy, with the latency of the whole model (the IR stage latency + the RR stage latency). As we see in Table 7, for larger K, the increase in accuracy is minor compared to the increase in latency, which goes from $0.578ms$ with $K = 15$ to $3.82ms$ with $K = 100$. Therefore, the top-15 is a reasonable size that provides a good accuracy while keeping the latency low.

Consistency of Similar TRs

In the last experiment, we evaluate if BERTicsson can produce similar ranking lists for similar TRs expressed differently. There is a high probability that different customers raise different TRs for the same underlying problem individually. The problems are written in different words and lengths but point to a similar or equal problem. We call these TRs, *duplicate* TRs. We want to check if BERTicsson can recommend the same documents to duplicate TRs. To this end, we compute the ranked lists for all the duplicate TRs, and consider two cases: (i) the model performs well by having the correct document is in the top-15 list, and (ii) the model fails. By analyzing the ranked list of the model for the cases that it performs well, we see that the correct document is ranked at a similar position in 70% of them.

We also try to identify duplicates by checking which document they rank at the top of the list regardless of whether it is correct. If we take all the duplicate TRs, we can check if the document they rank on top is the same as their duplicates. Of all the duplicate TRs in the test set, we can 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 of similar TRs. So, if the model can find duplicate TRs without explicit training, it means that it has learned the domain-specific company language and can make inferences about it.

Related Work

Many BERT-based text-ranking models have been developed over the past years, such as EPIC (MacAvaney et al. 2020), ColBERT (Khattab et al. 2020), and ANCE (Xiong et al. 2020). They all use BERT in diverse ways to rank documents concerning a query and present their results with general-domain data. One of the popular applications of text-raking is bug analysis and resolution, which is studied well in literature (Jiang et al. 2020); however, there is not much work on using BERT for it.

Companies like Ericsson have developed some solutions for automating bug resolution using an ensemble of different pre-BERT techniques (such as LSTM and LDA) (Ferland et al. 2020). However, these solutions do not consider the latency of the process as part of their performance metric. Another example of an ensemble of pre-BERT techniques is (Wang et al. 2015), where the authors formulate the problem as a non-convex optimization problem and solve it using a heuristic solution with a focus on accuracy. Consequently, their approach leads to a sub-optimal solution.

There are also many multi-stage BERT-based approaches for text-ranking. Some examples are duoBERT (Nogueira et al. 2019a) or DeeBERT (Xin et al. 2020). They present the results using general-domain data. However, to the best of our knowledge, none of them consider domain-specific tasks like automating the resolution of telecom TRs.

Conclusions

In this work, we present BERTicsson, a BERT-based model for analyzing Trouble Reports (TR) to retrieve solutions for newly reported problems. BERTicsson receives a dataset of existing TRs and, after pre-processing data, trains the model in two stages: Initial Retrieval (IR) stage and Re-Ranker (RR) stage. The IR stage uses Sentence-BERT to create a candidate list of answers for a problem (query), and the RR stage uses monoBERT to rank the candidate list according to the problem. We compare BERTicsson with BM25, a popular Exact Matching model, and show that BERTicsson can recommend the best possible solutions to a new error report from Ericsson with higher accuracy and lower latency, given the difficulty of the domain-specific data. Moreover, the model is consistent as it recommends the same answers to similar TRs expressed differently.

References

Chicco, D. 2021. Siamese neural networks: An overview. *Artificial Neural Networks*, 73–94.

Clark et al., K. 2020. Electra: Pre-training text encoders as discriminators rather than generators. *arXiv preprint arXiv:2003.10555*.

Devlin et al., J. 2018. Bert: Pre-training of deep bidirectional transformers for language understanding. *arXiv preprint arXiv:1810.04805*.

Ferland et al., N. 2020. Automatically Resolve Trouble Tickets with Hybrid NLP. In *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, 1334–1340. IEEE.

Furnas et al., G. 1987. The vocabulary problem in human-system communication. *Communications of the ACM*, 30(11): 964–971.

Guo et al., J. 2016. A deep relevance matching model for ad-hoc retrieval. In *Proceedings of the 25th ACM international on conference on information and knowledge management*, 55–64.

Harman et al., D. 2019. Information retrieval: the early years. *Foundations and Trends® in Information Retrieval*, 13(5): 425–577.

Henderson et al., M. 2017. Efficient natural language response suggestion for smart reply. *arXiv preprint arXiv:1705.00652*.

Huang et al., P. 2013. Learning deep structured semantic models for web search using clickthrough data. In *Proceedings of the 22nd ACM international conference on Information & Knowledge Management*, 2333–2338.

Jiang et al., J. 2020. 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*, 1410–1420.

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

Khattab et al., O. 2020. 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*, 39–48.

Lan et al., Z. 2019. Albert: A lite bert for self-supervised learning of language representations. *arXiv preprint arXiv:1909.11942*.

Liu et al., Y. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692*.

MacAvaney et al., S. 2020. Expansion via prediction of importance with contextualization. In *Proceedings of the 43rd International ACM SIGIR Conference on Research and Development in Information Retrieval*, 1573–1576.

Mitra et al., B. 2016. A dual embedding space model for document ranking. *arXiv preprint arXiv:1602.01137*.

Nguyen et al., T. 2016. MS MARCO: A human generated machine reading comprehension dataset. In *CoCo@ NIPS*.

Nogueira et al., R. 2019a. Multi-stage document ranking with bert. *arXiv preprint arXiv:1910.14424*.

Nogueira et al., R. 2019b. Passage Re-ranking with BERT. *arXiv preprint arXiv:1901.04085*.

Reimers et al., N. 2019. Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*.

Robertson et al., S. 2009. *The probabilistic relevance framework: BM25 and beyond*. Now Publishers Inc.

Sanh et al., V. 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108*.

Wang et al., Y. 2015. Generalized ensemble model for document ranking in information retrieval. *arXiv preprint arXiv:1507.08586*.

Wolf et al., T. 2020. Transformers: State-of-the-art natural language processing. In *Proceedings of the 2020 Conference on Empirical Methods in Natural Language Processing: System Demonstrations*, 38–45.

Wong et al., W. 2016. A survey on software fault localization. *IEEE Transactions on Software Engineering*, 42(8): 707–740.

Xin et al., J. 2020. Deebert: Dynamic early exiting for accelerating bert inference. *arXiv preprint arXiv:2004.12993*.

Xiong et al., C. 2017. 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*, 55–64.

Xiong et al., L. 2020. Approximate nearest neighbor negative contrastive learning for dense text retrieval. *arXiv preprint arXiv:2007.00808*.