Requirement or not, that is the question: A case from the railway industry

Sarmad Bashir[1,2], Muhammad Abbas[1,2]✉, Mehrdad Saadatmand[1], Eduard Paul Enoiu[2], Markus Bohlin[2], and Pernilla Lindberg[3]

[1] RISE Research Institutes of Sweden, Västerås, Sweden, {first.last}@ri.se
[2] Mälardalen University, Västerås, Sweden, {first.middle.last}@mdu.se
[3] Alstom, Västerås, Sweden, {first.last}@alstomgroup.com

Abstract. [**Context and Motivation**] Requirements in tender documents are often mixed with other supporting information. Identifying requirements in large tender documents could aid the bidding process and help estimate the risk associated with the project. [**Question/problem**] Manual identification of requirements in large documents is a resource-intensive activity that is prone to human error and limits scalability. This study compares various state-of-the-art approaches for requirements identification in an industrial context. For generalizability, we also present an evaluation on a real-world public dataset. [**Principal ideas/results**] We formulate the requirement identification problem as a binary text classification problem. Various state-of-the-art classifiers based on traditional machine learning, deep learning, and few-shot learning are evaluated for requirements identification based on accuracy, precision, recall, and F1 score. Results from the evaluation show that the transformer-based BERT classifier performs the best, with an average F1 score of 0.82 and 0.87 on industrial and public datasets, respectively. Our results also confirm that few-shot classifiers can achieve comparable results with an average F1 score of 0.76 on significantly lower samples, i.e., only 20% of the data. [**Contribution**] There is little empirical evidence on the use of large language models and few-shots classifiers for requirements identification. This paper fills this gap by presenting an industrial empirical evaluation of the state-of-the-art approaches for requirements identification in large tender documents. We also provide a running tool and a replication package for further experimentation to support future research in this area.

Keywords: Requirements identification · Requirements classification · tender documents · NLP

1 Introduction

Like many other industries, the project acquisition in the railway industry also starts with a call for tender. A tender document is a formal request calling for competing offers from different potential suppliers or contractors. A tender document typically consists of chunks of English text in the form of high-level technical specifications, supporting information, and contractual obligations. The

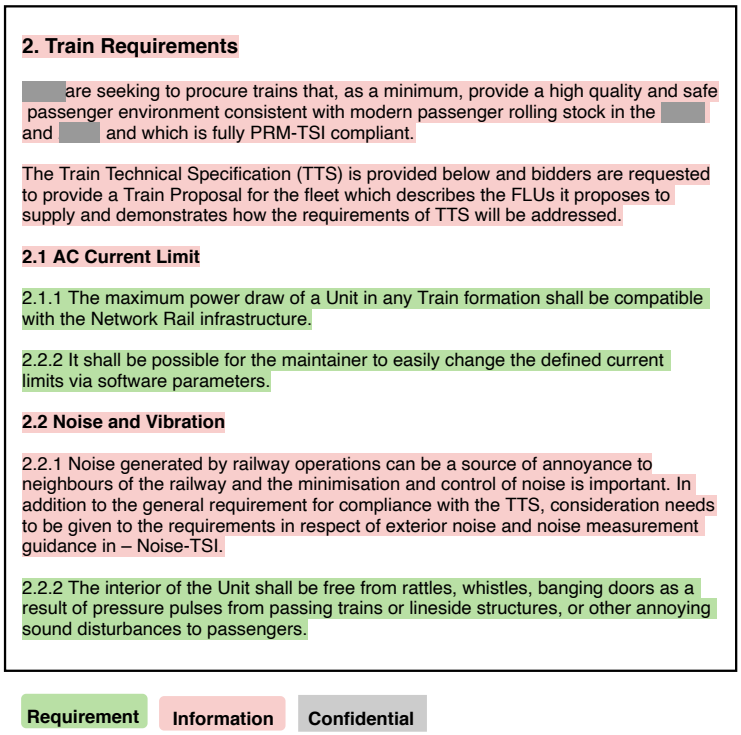

Fig. 1: Motivating example of requirements mixed with supporting text

very competitive market of railway vehicle manufacturing necessitates a quick response to such a call for tender. This requires estimating the risk associated with the call and the time required to deliver the end product. Extracting high-level technical specifications (requirements) from the tender document becomes crucial to estimate the risk and time required for a call. The high-level requirements are used to derive low-level requirements to be agreed upon. Furthermore, the requirements in the tender document are compared with already delivered products to estimate risk based on the novelty of the requirements. Therefore, identifying technical specifications from tender documents becomes a pre-requisite to enable project acquisition and later Requirement Engineering (RE) tasks. Moreover, errors and inaccuracies in this phase can have cascading effects on the rest of the development process.

Figure 1 shows a motivating example from a real tender document showing requirements mixed with technical specifications. Manually identifying the requirements in large tender documents could be time-consuming and prone to human error. Requirement identification in a large document can be automated by formulating it as a binary text classification problem. According to Berry [7], the automated solutions for RE tasks should ideally have a 100% recall rate; however, this is not often achieved in Machine learning (ML)-based solutions. Despite

this, utilizing automated solutions could still accelerate the process since human input would only be required to sanitize the final output. The ultimate goal must be to optimize the performance of these automated classification solutions to alleviate the workload in practical settings. In RE, requirement classification is one of the most prominent activities, as reported in literature [39]. Related work on distinguishing requirements from other information often experimented with traditional ML-based approaches for classification [35, 4, 13]. Furthermore, the work of Abualhaija *et al.* [4, 3] considers single sentences as a unit of classification. However, requirements and information could range over multiple sentences in our case. Nevertheless, the same approaches could be modified to take multi-sentence input. However, the performance of large transformer-based language models and few-shot classifiers in the task is still unclear. On the other hand, work on distinguishing functional and non-functional requirements [20, 29, 5, 15] is a different use case, and studies in the domain often use public datasets with some exceptions.

This study is conducted in close collaboration with Alstom, Sweden (Alstom), a world-leading railway vehicle manufacturing company. The main objective of this study is to find a practical solution to the requirements identification problem at Alstom. Therefore, this study reports an empirical evaluation of 20+ different classification pipelines for distinguishing requirements from supporting text in large documents. The selected seminal pipelines include approaches from traditional ML, deep learning, and transformer-based classifiers. In addition, we leverage new approaches based on few-shot learning to address the common challenge of data scarcity in the RE domain. Furthermore, to support further research on the topic, we evaluated the same pipelines on a public dataset and provided a replication package with a running tool[4]. This paper may also refer to requirements identification as distinguishing requirements or classification.

The rest of the paper is structured as follows. Section 2 provides a brief overview of the related work and background. Section 3 presents the study design and the selected classification pipelines for requirements identification. Section 4 presents and discusses the results. Section 5 presents potential validity threats and limitations. Finally, Section 6 concludes the paper with future directions.

2 Related Work & Background

Related Work. This paper focuses on identifying requirements through automated classification. Binkhonain and Zhao [8] performed a systematic literature review on one aspect of the RE process, i.e., automated requirements classification, specifically providing solutions to distinguish between functional (FRs) and non-functional requirements (NFRs). Often FRs are related to the core functionality, and NFRs describe the properties and constraints of the system. The distinction between FRs and NFRs impacts the handling of requirements elicitation, documentation, and validation process [12]. Therefore, the task of automatic extraction and classification of requirements has been the focus of RE

[4] Replication package and Tool: https://github.com/a66as/REFSQ2023-ReqORNot

researchers. Within this group of studies, Jindal *et al.* [20] employs an automated approach to extract and classify security requirements. They use term-frequency inverse document frequency (tfidf) weight vectors to analyze the security requirements with the goal of further classifying into sub-categories of security based on the Decision Tree (DT) algorithm. Moreover, Varenov *et al.* [32] proposes a sentence-level classifier based on fine-tuned DistilBERT [29] to allocate security requirements into predefined groups. Recently, Alhoshan *et al.* [5] leverages a Zero-Shot Learning (ZSL) technique on a subset of the PROMISE dataset to classify NFRs into two categories, i.e., Usability and Security. Furthermore, Herwanto *et al.* [15] propose an automated approach to identify privacy requirements in user stories based on the Named Entity Recognition (NER) model, trained on Bi-directional Long Short Term Memory Networks (BI-LSTM) with conditional random field [18].

The other more related thread of work is distinguishing requirements from other information. Similar to our use case, Winkler *et al.* [35] propose a deep learning (DL) classifier based on Convolution Neural Networks (CNNs) to identify requirements from additional material stored in IBM DOORS. Falkner *et al.* [13] propose a Naive Bayes (NB) classifier—trained on unique words—to identify requirements from Request of Proposal (RFP) documents within the railway safety domain. Furthermore, Abualhaija *et al.* [4] proposes an automated ML-based approach to demarcate requirements in textual specifications by considering one sentence as a unit of classification. They empirically evaluate ML classifiers on the industrial dataset consisting of 12 documents. In addition, Sainani *et al.* [28] defines a two-step methodology to first extract requirements from 20 Software Engineering (SE) contracts and then allocate them to their specific types. For identification and extraction of requirements, Bi-LSTM yields the best results compared to ML algorithms. To allocate identified requirements in sub-classes, BERT (Bi-directional Encoder Representations from Transformers) performed better in terms of F-1 score.

While our work shares the same general objective as the above-mentioned approaches, we address the need for extensive empirical evaluation in automated requirements identification and classification on industrial and public datasets. Furthermore, we evaluated a new approach, namely a few-shot classifier, to identify requirements based on a limited dataset, a well-known problem in the RE domain.

Background. Most of the ML or DL algorithms for classification do not work with raw text but instead require transformed data as feature vectors. The feature vectors can be generated with information retrieval (IR)-based lexical approaches or with semantic approaches. In our work, we utilize tfidf vectors—a lexical approach—to represent and train data on classical supervised ML algorithms. We further apply the dimensionality reduction technique, i.e., principal component analysis (PCA), on tfidf vectors to increase interpretability through the creation of newly uncorrelated features with maximum variance.

Traditionally, Language Models (LMs) capture regularities, morphological and distributional properties of a language. For DL algorithms, we consider state-

of-the-art semantic strategies based on LMs and neural networks. The semantic-based LMs are coupled with a statistical classifier to perform classification. We use FastText (FT) [9] and GloVe (GLV) [24] semantic representations to train LSTM neural network for the identification of requirements in large documents. Furthermore, we fine-tuned multiple token-based BERT LM variations based on transformer architecture [33]. Originally, token-based BERT LM comes in two variants for language representation, BERT base and large, pre-trained on 16 GB data from Toronto BookCorpus and English Wikipedia dataset. With the advent of transfer learning, token-based BERT LMs have been widely used for different downstream tasks—in our case, classification to distinguish requirements.

Additionally, we perform few-shot fine-tuning on different variations of Sentence Transformers [25] (ST)—a modified version of the pre-trained BERT LM based on the siamese network. Specifically, we fine-tuned Sentence-BERT [25] (S-BERT) and MiniLM-L12-v2 [34] (Mini-LM) on our datasets. Originally, ST LMs are pre-trained for tasks like clustering and semantic search. However, we can fine-tune ST LMs through Sentence Transformer Fine-Tuning [31] (SETFIT) framework with a small number of examples for our requirements classification task. Few-shot methods are an attractive solution and can address the long-standing problem of data shortage in the RE domain.

3 Study Design

This work can be regarded as an *exploratory* case study oriented towards improving the project acquisition process at Alstom. Following the guidelines of Runeson and Höst [26], this section outlines the context, objectives, data collection, and analysis procedure.

3.1 Case Context

Rail vehicle manufacturing is a globally competitive market. Like many other industries, customers in the railway industry also publish a call for tender to which companies respond. The tender document often contains contractual obligations, supporting information, and technical specifications of the required product. In response to the call for tender, in addition to understanding the contractual obligations, companies must also identify potential requirements from the documents to achieve the following objectives.
a) The extracted technical specifications must be reflected in deriving the customer requirements to be agreed upon. This can aid the project acquisition process.
b) The risk associated with the new project must be estimated to enable the project resource and time management. This is done by comparing the extracted technical specification to the already delivered projects, currently based on experience.
Alstom is continuously looking for ways to improve the process of project acquisition with tool support. As a first step, automated approaches for distinguishing

requirements from other supporting information are investigated in this study. In this regard, for this paper, the *case* under study is the performance of various classification pipelines in requirements identification. The *units* under analysis are five tender documents from Alstom and the public dronology dataset [10].

3.2 Objective and Research Questions

Our main goal is to improve the project acquisition phase in the studied context. As an initial step (this study), we first need to identify the requirements within the tender documents. Requirements identification problem can be formulated as a binary text classification problem. There have been a number of approaches proposed for binary classification over the years. Therefore, this work is not "*reinventing the wheel*" but instead aims to find an already existing practical—in terms of execution time—solution for the problem in the studied context. As discussed in the following sections, we consider seminal state-of-the-art classifiers for this study. In addition, since the considered approaches for classification might react differently to text pre-processing, we also study the impact of pre-processing on classification performance. To this end, we pose the following research questions (RQs):

- *RQ1: What is the performance of different classification pipelines in requirements identification?*
- *RQ2: What is the impact of pre-processing on classification performance?*
- *RQ3: What is the execution time of each classification pipeline?*

Table 1: Datasets

Dataset	**Reqs.**	**Info.**	**Sent.**	**AW**	**pAW**	**TRD**	**TSD**
Industrial	1680	1293	8332	39	20	2378	595
Public	99	280	533	25	13	303	76

* AW= Avg. words, pAW= Avg. words when pre-processed, TRD= Avg. training dataset rows, TSD= Avg. test dataset rows

3.3 Data collection

Industrial case. We had access to five already annotated tender documents from our industrial partner. The tender documents contain multi-sentence chunks of text explaining the technical specifications, contractual obligations, and supporting information. The requirements among the documents were already tagged, and the projects were already delivered to customers. Therefore, the ground truth on whether a chunk of text is a requirement or not is already available in the

dataset. Note that the selected pipelines (see coming sections) for distinguishing requirements require annotated input for training only. We selected all the requirements and non-requirements among these five documents using the following steps. First, we removed all the duplicates across the five files and considered unique chunks of text. To avoid selecting potential non-requirements as requirements, we selected only the requirements that were also allocated to a team for development. A total set of 1680 requirements and 1293 non-requirements from the industrial documents was reached, as shown in the first row of Table 1.
Public dronology dataset. The dronology public dataset consists of 398 entries of various types such as "components", "requirements", "design definitions", and "sub-task". Among these entries, 99 entries are tagged as requirements. We prepared the dronology dataset for this study as follows. First, we considered all the requirements as requirements and components, design definitions, and subtasks as non-requirements. Then, we dropped (19 entries) entries with no text. A total set of 99 requirements and 280 non-requirements was reached, as shown in the last row of Table 1. Note that this dataset does not directly represent the studied context; however, we argue that evaluating the pipelines on this similar dataset would support replication and reproducibility of our results.

All the considered classification pipelines (see the coming section) are fed the data with and without pre-processing. As shown in the `Sent.` column, the total number of sentences in the considered datasets are 8,332 and 533, respectively. On average, each of the entries consists of 39 and 25 words. After pre-processing and stop word removal, the average words across all entries drops to 20 and 13 for the industrial and public datasets, respectively. Due to the uneven distribution of data over the labels, the *p-fold* cross-validation method is not employed for evaluation [11]. As typical, we used stratified five-fold cross-validation to evaluate the selected pipelines. The average number of entries per fold in the training set and the test are 2378 and 595, respectively.

3.4 Pipelines for distinguishing requirements

For this study, we considered the most seminal text classification approaches for evaluation in distinguishing requirements from ordinary text. As typical in the NLP domain, pre-processing of the input text might impact classification performance. Therefore, we also consider the datasets both with and without pre-processing. In addition, we also consider a baseline random pipeline (W. Rand.) that classifies input as a requirement or not based on their frequency distribution in the dataset.

Pre-processing: Our pre-processing pipeline consists of tokenization, stopwords removal, part-of-speech (POS) tagging, and lemmatization of the input text using spaCy [17]. An output of the pre-processing pipeline for the requirement '6.3.1' from Figure 1 is presented as follows. "maximum power draw unit train formation compatible network rail infrastructure".

Traditional ML-based classifiers: For lexical classifiers, we considered widely used and recommended ML algorithms, e.g., Support Vector Machines (SVM),

Logistic Regression (LR), DT, Random Forest (RF), and NB. For a fair comparison and tuning, we applied random multi-search optimization [6] to select the optimal hyperparameters. SVM and LR achieved better results on evaluation metrics when trained with normalized and reduced tfidf vectors using PCA. However, the rest of the ML pipelines—RF, DT, and NB—performed better with normalized TF-IDF vectors without PCA-based dimensionality reduction.

Deep semantic representation based classifiers: For the training and evaluation of DL-based LSTM networks, we use FT and GLV LMs—pre-trained and custom (self-trained)—embeddings for semantic representation. To generate the custom embeddings, we train the FT LM on 20 epochs, with word embeddings (WE) dimension size set to 100 and window size set to three. For custom GLV embeddings, we get the best results when the window size is set to 10, the learning rate is set to 0.05, the WE dimension size is 100, and the epochs are set to 30. We defined a two-layer LSTM network to train on custom and pre-trained WEs. To minimize the training loss function, we used Adam [21] optimizer with a learning rate of 0.001. Furthermore, we prevent the over-fitting of the network by appending a dropout layer—randomly dropping units with their connections—with a rate of 0.1 after every LSTM layer. The batch, epochs, and maximum sequence sizes are set as 32, 10, and 128, respectively.
We selected widely used BERT variants, i.e., SciBERT, RoBERTa, BERT base, XLMRoBERTa (XRBERT), DistilBERT (DisBERT), and XLNet. To fine-tune different variations of the token-based BERT family, we employ a BERT WordPiece [37] tokenizer to prepare the datasets. The WordPiece tokenizer splits the words of a text into one word per token or into word pieces—where one word is tokenized into multiple words. We use the AdamW-optimizer, an adoption of Adam with a weight decay of 0.01, to optimize the weights while fine-tuning the token-based BERT network [23]. Furthermore, we select a maximal learning rate of 2e-05 instead of aggressive learning rates with the purpose of avoiding catastrophic forgetting of BERT pre-trained knowledge [30]. We set a practical batch and maximum sequence size as 16 and 128 across all the token-based BERT pipelines. We set the epoch size as 10 to iterate the datasets over the BERT's network. The reason behind selecting a higher number of epochs on relatively smaller datasets is that BERT's common one-size-fits-all is sub-optimal and needs more training time to stabilize the network [38]. However, some studies in the literature have set an even higher number of epochs (e.g. [16]), but we argue it may lead to over-fitting.

Few-shot learning based on sentence transformers: To fine-tune different variations of pre-trained ST, we utilize the SETFIT framework for our downstream requirements classification task. SETFIT consists of a two-step training approach. In the first step, we fine-tuned ST on a limited dataset—few shots—with a contrastive training approach—frequently used for image similarity [22]. In a few-shot scenario, contrastive fine-tuning enlarges the training dataset by creating positive and negative pairs through in-class and out-class selection. In the second step, we train an LR (Logistic Regression) model as a classification head on the embeddings—encoded through fine-tuned ST—with original labeled

training data. For evaluation, fine-tuned ST generates the sentence embeddings of unseen examples, and then the LR model predicts the class label of the input sentence embeddings. To fine-tune the ST model, we use the cosine-similarity loss function with a learning rate of 2e-5 and a batch size of 16. We set the number of iterations for the generation of text pairs for contrastive learning to 20 with one epoch.

3.5 Metrics for Evaluation

We use the standard evaluation metrics for text classification, as follows. *Accuracy (A)* is the ratio of the number of correct predictions and the total predictions. *Precision (Prec.)* is the ratio of correct positive predictions and the total number of positive predictions. *Recall (Rec.)* quantifies the number of correct positive predictions from all possible positive predictions. *F1 score (F1)* is the harmonic mean of precision and recall. We report the macro and weighted average across the fold for all our evaluation metrics. However, to answer our research questions, we use weighted averages of the metrics for simplicity.

3.6 Execution Procedure

Both datasets' tagged requirements and information are moved to two separate files. Using random stratified five-fold sampling [19], we created five folds from each dataset for cross-validation. As mentioned, each fold consists of 80% of the randomly sampled data in the training set and 20% of the data in the holdout set. All the selected pipelines were fed with the five folds for training the models, and the holdout sets were used to compute the evaluation metrics. In the case of the few-shot classification pipelines, we only selected 10% and 20% of the training set as folds to train the model and evaluated it using the entire holdout set. We executed all the experiments on a local server using parallel computing. The server is configured with four Nvidia Tesla M10 graphics processing units, an Intel Xeon Gold 5122 processor @ 3.60GHz, and primary memory of 256 GB.

4 Results & Discussion

Table 2 and Table 3 show the experiment's results. The names of Pipelines starting with a 'p' indicate that our pre-processing pipeline was coupled with the classification pipeline. The `Weighted Average` and `Macro Average` columns show the weighted and macro averages of our evaluation metrics across the five folds. `Avg. A.` shows the average accuracy of the pipelines.

RQ1: Performance. As shown in Table 2, among the traditional machine learning-based approaches, SVM slightly outperformed the others in terms of F1 score. Regarding accuracy, RF and LR slightly outperformed all other requirement identification pipelines based on ML. A similar trend can also be observed in the public dataset. This is in line with the results in the literature. Interestingly, the deep-learning-based LSTM model combined with the word embeddings

Table 2: Performance and execution time of the pipelines on industrial case

Pipeline	**Setup**	**Weighted Average**			**Macro Average**			**Avg. A.**	**Time (mins)**	
		Prec.	**Rec.**	**F1**	**Prec.**	**Rec.**	**F1**	**A**	**Tr**	**Ts**
W. Rand.	Freq. based	.49	.49	.49	.49	.49	.48	.49	-	-
SVM	Norm., PCA	.79	.79	.79	.80	.78	.78	.78	.70	.02
pSVM	Norm., PCA	.78	.78	.78	.79	.77	.77	.78	.74	.09
NB	Norm.	.74	.69	.69	.73	.71	.69	.69	<.01	<.01
pNB	Norm.	.74	.68	.67	.72	.70	.68	.68	.29	.07
DT	Norm.	.72	.72	.72	.71	.71	.71	.71	<.01	<.01
pDT	Norm.	.71	.71	.71	.71	.71	.71	.71	.29	.07
LR	Norm., PCA	.79	.79	.78	.79	.77	.78	.79	.30	<.01
pLR	Norm., PCA	.78	.78	.78	.79	.76	.77	.78	.41	.07
RF	Norm.	.79	.79	.79	.79	.78	.78	.79	<.01	<.01
pRF	Norm.	.79	.78	.78	.79	.77	.77	.78	.38	.07
LSTM	FT custom	.77	.77	.77	.76	.76	.76	.77	1	.02
pLSTM	FT custom	.75	.75	.75	.75	.75	.74	.75	1	.08
LSTM	FT pre-train	.75	.75	.75	.74	.74	.74	.75	2	.02
pLSTM	FT pre-train	.72	.72	.72	.72	.72	.72	.72	1.2	.08
LSTM	GLV custom	.77	.77	.77	.77	.76	.76	.77	2	.02
pLSTM	GLV custom	.76	.76	.76	.76	.75	.76	.76	1.2	.09
LSTM	GLV pre-train	.78	.78	.78	.78	.77	.78	.78	2	.02
pLSTM	GLV pre-train	.78	.78	.78	.78	.77	.78	.78	1.3	.08
SciBERT	uncased	**.82**	.81	.81	**.82**	.80	.80	.81	34	.25
pSciBERT	uncased	.80	.78	.76	.81	.75	.75	.78	32	.30
RoBERTa	base	.81	.81	.81	**.82**	.80	.80	.81	39	.27
pRoBERTa	base	.80	.79	.79	.81	.78	.78	.79	37	.32
BERT	base, cased	**.82**	**.82**	.81	**.82**	**.81**	**.81**	**.82**	35	.29
pBERT	base, cased	.79	.79	.79	.79	.79	.79	.80	32	.32
BERT	base, uncased	**.82**	**.82**	**.82**	**.82**	**.81**	**.81**	**.82**	34	.29
pBERT	base, uncased	.80	.80	.80	.80	.79	.79	.80	32	.33
XRBERT	base	**.82**	.81	.81	**.82**	.80	**.81**	.81	57	.29
pXRBERT	base	.78	.77	.77	.78	.76	.76	.77	41	.25
DisBERT	base, cased	.81	.81	.81	.81	.80	.80	.81	31	.13
pDisBERT	base, cased	.80	.80	.80	.80	.79	.79	.80	25	.18
DisBERT	base, uncased	.81	.81	.81	.81	**.81**	.80	.81	31	.15
pDisBERT	base, uncased	.80	.80	.70	.81	.78	.79	.80	29	.21
XLNet	base	.81	.81	.80	.81	.80	.80	.81	47	.36
pXLNet	base	.81	.80	.80	.81	.79	.79	.80	47	.42
S-BERT	10% train	.75	.75	.75	.75	.74	.75	.75	24	.14
pS-BERT	10% train	.73	.73	.73	.72	.72	.72	.73	18	.20
Mini-LM	10% train	.74	.74	.74	.74	.74	.74	.74	7	.04
pMini-LM	10% train	.72	.72	.72	.72	.72	.71	.72	6	.10
S-BERT	20% train	.77	.77	.76	.76	.76	.76	.77	45	.17
pS-BERT	20% train	.74	.74	.74	.74	.74	.74	.74	37	.20
Mini-LM	20% train	.75	.75	.75	.75	.74	.74	.75	14	.03
pMini-LM	20% train	.72	.72	.72	.72	.72	.72	.72	11	.10

Table 3: Performance and execution time of the pipelines on the Dronology public dataset

Pipeline	**Setup**	**Weighted Average**			**Macro Average**			**Avg. A.**	**Time (mins)**	
		Prec.	**Rec.**	**F1**	**Prec.**	**Rec.**	**F1**	**A**	**Tr**	**Ts**
W. Rand.	Freq. based	.60	.58	.59	.48	.48	.48	.58	-	-
SVM	Norm., PCA	.78	.79	.75	.76	.63	.64	.78	.18	< .01
pSVM	Norm., PCA	.78	.77	.70	.80	.57	.55	.77	.03	<.01
NB	Norm.	.70	.55	.58	.58	.61	.54	.55	<.01	<.01
pNB	Norm.	.71	.56	.58	.60	.62	.55	.56	.03	<.01
DT	Norm.	.74	.74	.74	.67	.66	.66	.74	<.01	<.01
pDT	Norm.	.72	.72	.72	.64	.63	.63	.72	.03	<.01
LR	Norm., PCA	.74	.75	.67	.73	.54	.51	.75	.14	.02
pLR	Norm., PCA	.70	.75	.64	.67	.52	.46	.74	.03	< .01
RF	Norm.	.76	.78	.75	.72	.64	.65	.78	.01	<.01
pRF	Norm.	.77	.78	.75	.74	.63	.65	.78	.04	<.01
LSTM	FT custom	.75	.78	.73	.72	.61	.61	.78	.24	.01
pLSTM	FT custom	.76	.77	.75	.71	.66	.67	.77	.18	.02
LSTM	FT pre-train	.74	.76	.75	.67	.66	.66	.76	.23	.01
pLSTM	FT pre-train	.67	.68	.68	.58	.57	.58	.68	.17	.02
LSTM	GLV custom	.73	.75	.73	.65	.66	.63	.75	.23	.01
pLSTM	GLV custom	.77	.78	.77	.72	.69	.69	.78	.17	.02
LSTM	GLV pre-train	.80	.80	.79	.74	.73	.73	.80	.26	.01
pLSTM	GLV pre-train	.74	.75	.74	.68	.65	.66	.75	.18	.02
SciBERT	uncased	.84	.84	.83	.79	.78	.78	.83	5	.03
pSciBERT	uncased	.87	.87	.86	.84	.80	.82	.87	5	.03
RoBERTa	base	.82	.86	.84	.76	.78	.77	.86	5	.03
pRoBERTa	base	.80	.81	.79	.77	.69	.70	.81	5	.04
BERT	base, cased	**.88**	**.88**	**.87**	**.85**	.83	**.83**	**.88**	3	.01
pBERT	base, cased	.83	.84	.82	.81	.74	.76	.84	3	.03
BERT	base, uncased	**.88**	**.88**	**.87**	.84	**.84**	**.83**	**.88**	3.5	.02
pBERT	base, uncased	.83	.84	.83	.80	.75	.77	.84	3	.03
XRBERT	base	.86	.86	.86	.82	.83	.82	.86	7	.03
pXRBERT	base	.86	.86	.86	.82	.83	.82	.81	7	.04
DisBERT	base, cased	.85	.85	.85	.80	.81	.80	.85	3	.01
pDisBERT	base, cased	.83	.83	.82	.79	.75	.76	.83	3	.02
DisBERT	base, uncased	.85	.86	.85	.81	.81	.80	.86	3	.01
pDisBERT	base, uncased	.82	.83	.82	.79	.74	.75	.83	3	.02
XLNet	base	**.88**	.87	**.87**	**.85**	.83	**.83**	.87	6	.04
pXLNet	base	.82	.83	.83	.78	.76	.77	.83	6.5	.05
S-BERT	10% train	.75	.65	.66	.64	.67	.62	.65	3	.04
pS-BERT	10% train	.68	.59	.61	.57	.58	.55	.59	2	.04
Mini-LM	10% train	.76	.67	.69	.66	.70	.64	.67	1	<.01
pMini-LM	10% train	.70	.56	.58	.58	.60	.54	.56	.40	.01
S-BERT	20% train	.75	.65	.66	.64	.67	.62	.67	4	.03
pS-BERT	20% train	.74	.65	.67	.63	.66	.62	.65	4	.04
Mini-LM	20% train	.79	.68	.70	.68	.73	.66	.68	2	<.01
pMini-LM	20% train	.77	.71	.72	.67	.72	.67	.71	1	.01

model does not exhibit a significant improvement compared to traditional ML algorithms for classification in both public and industrial cases. Our results indicated that SVM is closely followed by LSTM coupled with the pre-trained GLV WE model. We argue that this could be because of the impact of feature engineering in SVM. Generally, SVM's performance is similar or better compared to artificial neural networks (ANN) when there is less training dataset. On the other hand, on average, the BERT family slightly outperformed all other pipelines with traditional fine-tuning. The BERT base uncased-based pipeline for requirements identification slightly outperformed all other pipelines across all evaluation metrics, with an average F1 score of 0.82 for the industrial dataset and 0.87 on the public dataset. This could be explained by BERTs' ability to capture the long-range dependencies in sequential data through its so-called self-attention mechanism. Additionally, the capability of fine-tuning BERT pre-trained parameters on task-specific datasets allows the model to better incorporate domain-specific knowledge than other traditional approaches. All other sub-families of BERT—SciBERT, RoBERTA, XRBERT, and DisBERT—closely followed the BERT base uncased-based pipeline. In addition, the performance of XLNet is also close to the performance of the BERT family. The architecture of XLNet and BERT family models are different, but they share a similar pre-training objective to capture the contextual relationships in natural language data. Therefore, when fine-tuned on a similar dataset for a classification task, there is not a significant difference in terms of evaluation metrics.

The lack of larger datasets in the RE domain is a commonly highlighted problem in the literature [5, 14, 39]. Therefore, evaluating few-shot learning approaches for requirements identification is equally important. Our results show that our selected sentence transformer-based few-shot classification pipelines for requirements identification achieved comparable results with as little data as 20% of the training set used for training the models. The few-shots classification pipelines also performed very well when only 10% of training data was used for training. In our industrial case, the best-performing few-shot classification pipeline is the pipeline based on the S-BERT model. For comparison, fine-tuned S-BERT achieves an F1 performance score of 0.76, which is only 0.06 less than the best-performing fined-tuned BERT uncased pipeline on a complete dataset. This is a significant step in the RE domain because S-BERT only requires a few samples to fine-tune the model, and it can address the standing challenge of insufficient annotated RE datasets. Furthermore, this can help RE researchers to completely exploit DL classifiers' usage in various phases of the RE.

Based on the presented results, we summarize an answer to RQ1.

Answer to RQ1. The BERT base uncased-based pipeline for distinguishing requirements from general text slightly outperformed all other pipelines across the two datasets with an average F1 score of 0.85. However, no significant difference in the performance of the BERT family is observed. Results further indicate that few-shot classification pipelines for distinguishing re-

quirements perform well (with an F1 score of 0.76) on significantly fewer samples.

Note that the current performance evaluation of the pipelines is based solely on the results obtained from the already annotated datasets. However, to further assess and validate the effectiveness of the pipelines in practice, it is necessary to conduct a controlled experiment in an industrial setting. This is because the real-world efficiency of the automated pipelines for such tasks is also dependent on the environment in which it will be used [7]. Additionally, the evaluation of automated pipelines must consider the context for its performance evaluation relative to the task manually performed by humans. Therefore, following the study of Winkler *et al.* [36], in the future, we plan to perform an empirical study to further validate our solution in practice. The findings of such an experiment would lead to further improvements in the pipelines and would highlight the avenues for future research in the studied context.

RQ2: Impact of pre-processing. Table 2 and 3 also contains the evaluation results of the pipelines with pre-processing. Based on our experience and literature in NLP, traditional ML-based approaches typically improve performance when pre-processing is applied [2, 1]. However, we found a general trend in the task of distinguishing requirements; on average, pre-processing has a negative impact on classification performance. Particularly among the traditional approaches, all pipelines (except LR and NB) show a decrement of up to .02 in the F1 score when pre-processing is applied. The LSTM family also shows a negative relationship between pre-processing and F1 score. However, the LSTM pipeline based on the GLV word embedding model for distinguishing requirements shows no impact on performance when pre-processing is applied. Finally, as expected for all the transformer-based models, pre-processing has a negative impact on the model performance. Based on these results, we summarized the answer to RQ2 as follows.

Answer to RQ2. Generally, we observed a negative impact of pre-processing (with stop words removal and lemmatization) on model performance in the task of distinguishing requirements from general text.

RQ3: Execution time. The `Time (mins)` column in Table 2 and Table 3 also shows the average execution time of the pipelines per fold both in training (`Tr`) and in inference mode (`Ts`). Pipelines with pre-processing—both in training and inference mode—also report an average pre-processing time added to the overall time. In other words, the `Time (mins)` column shows the pipeline's average end-to-end execution time per fold.

As expected, the traditional ML-based and LSTM-based approaches converge faster, with an average end-to-end execution time of under two minutes in training. Likewise, the average end-to-end execution time in inference mode across the folds—with 595 and 76 entries per fold in the industrial and public dataset—is also under a minute. In addition, for fine-tuning the BERT family, the end-to-end execution time is under an hour in the worst case. Note that the

pre-processing for LSTM and BERT family does add an overhead. However, in some cases, the same pipeline with pre-processing takes even less time than the one without pre-processing. This could be explained by the fact that the same model has to train on less vocabulary than when the data was not pre-processed. Nevertheless, the reported execution time in training still shows a trend of increase as the size of the model increases. BERT family averaged an inference time of under a minute per fold. As fine-tuning is done only once per task and can be done at night, the engineers do not have to wait for more than a minute in inference mode—which is how end-users use these models. Based on the results, we summarised an answer to RQ3 as follows.

Answer to RQ3. Pipeline for distinguishing requirements produces results for input (500+ entries as input) in under two minutes in the worst case. Fine-tuning large language models for classification tasks could take hours on high-compute units when training on a dataset with 2300 entries. However, fine-tuning is often done once per task. Therefore, the approaches could still be practical in a real-world context and can aid the project acquisition process.

5 Threats to Validity

This section presents validity threats according to Runeson et al. [26].

Construct validity. As typical, we cast the requirement identification as a binary text classification problem. Our unit of classification ranges over multiple sentences. However, in some cases, the input might contain some sentences that are requirements and others that are not. We do not tackle such cases. We argue that considering already delivered projects' tender documents where experts tagged requirements and allocated them to teams for implementation resolves such issues.

Internal validity. Internal validity threats affect the validity and credibility of our results. We based our implementation on open-source libraries and publicly available language models to address potential internal validity threats. Furthermore, we shared the replication package and a running tool to support future research.

External validity. Our results are obtained from five representative documents from one company that might not represent the whole railway domain. Therefore, for the generalizability of our results, we also include a public dataset for evaluation. However, as typical for case studies, we do not claim the generalizability of our results beyond the studied context.

6 Conclusion and Future Work

Requirements identification in larger documents enables a quick response to the call for tenders and could help later RE tasks such as retrieval for reuse and deriving low-level requirements. This study is oriented toward finding a

practical solution to the requirements identification problem in a large railway company using classification. Therefore, the study evaluates a variety of classification approaches in the requirements identification contexts. Our results show that the transformer-based approaches slightly outperform all other approaches in the requirements identification task. Particularly, the BERT base uncased-based pipeline performs the best in terms of F1 score and produces results in practical time. Finally, results also indicate that few-shot classifiers can achieve comparable performance with as little as 20% of the training data. We argue that the use of few-shot learning in RE tasks should be investigated further.

In the future, we plan to conduct a controlled experiment in the studied settings to evaluate the effectiveness of the developed solution for comparison to manual requirements identification. Additionally, we aim to pre-train large language models on railway industry-specific documents and compare the results in two classification tasks, i.e., requirements identification and allocation of requirements to different teams. Extending the current tool to estimate the risk associated with a tender call is also planned for future work.

Acknowledgement. This work is partially funded by the AIDOaRt (KDT) and SmartDelta [27] (ITEA) projects.

References

1. Abbas, M., Ferrari, A., Shatnawi, A., Enoiu, E., Saadatmand, M., Sundmark, D.: On the relationship between similar requirements and similar software. Requirements Engineering pp. 1–25 (2022)
2. Abbas, M., Saadatmand, M., Enoiu, E., Sundamark, D., Lindskog, C.: Automated reuse recommendation of product line assets based on natural language requirements. In: International Conference on Software and Software Reuse. pp. 173–189. Springer (2020)
3. Abualhaija, S., Arora, C., Sabetzadeh, M., Briand, L.C., Traynor, M.: Automated demarcation of requirements in textual specifications: a machine learning-based approach. Empirical Software Engineering **25**(6), 5454–5497 (2020)
4. Abualhaija, S., Arora, C., Sabetzadeh, M., Briand, L.C., Vaz, E.: A machine learning-based approach for demarcating requirements in textual specifications. In: 2019 IEEE 27th International Requirements Engineering Conference (RE). pp. 51–62. IEEE (2019)
5. Alhoshan, W., Zhao, L., Ferrari, A., Letsholo, K.J.: A zero-shot learning approach to classifying requirements: A preliminary study. In: International Working Conference on Requirements Engineering: Foundation for Software Quality. pp. 52–59. Springer (2022)
6. Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. Journal of machine learning research **13**(2) (2012)
7. Berry, D.M.: Empirical evaluation of tools for hairy requirements engineering tasks. Empirical Software Engineering **26**(6), 111 (2021)
8. Binkhonain, M., Zhao, L.: A review of machine learning algorithms for identification and classification of non-functional requirements. Expert Systems with Applications: X **1**, 100001 (2019)

9. Bojanowski, P., Grave, E., Joulin, A., Mikolov, T.: Enriching word vectors with subword information. Transactions of the association for computational linguistics **5**, 135–146 (2017)
10. Cleland-Huang, J., Vierhauser, M., Bayley, S.: Dronology: An incubator for cyber-physical systems research. In: 2018 IEEE/ACM 40th International Conference on Software Engineering: New Ideas and Emerging Technologies Results (ICSE-NIER). pp. 109–112 (2018)
11. Dell'Anna, D., Aydemir, F.B., Dalpiaz, F.: Evaluating classifiers in se research: the ecser pipeline and two replication studies. Empirical Software Engineering **28**(1), 1–40 (2023)
12. Eckhardt, J., Vogelsang, A., Fernández, D.M.: Are" non-functional" requirements really non-functional? an investigation of non-functional requirements in practice. In: 38th International Conference on Software Engineering. pp. 832–842 (2016)
13. Falkner, A., Palomares, C., Franch, X., Schenner, G., Aznar, P., Schoerghuber, A.: Identifying requirements in requests for proposal: A research preview. In: International Working Conference on Requirements Engineering: Foundation for Software Quality. pp. 176–182. Springer (2019)
14. Ferrari, A., Dell'Orletta, F., Esuli, A., Gervasi, V., Gnesi, S.: Natural language requirements processing: A 4d vision. IEEE Softw. **34**(6), 28–35 (2017)
15. Herwanto, G.B., Quirchmayr, G., Tjoa, A.M.: A named entity recognition based approach for privacy requirements engineering. In: 2021 IEEE 29th International Requirements Engineering Conference Workshops (REW). IEEE (2021)
16. Hey, T., Keim, J., Koziolek, A., Tichy, W.F.: Norbert: Transfer learning for requirements classification. In: 2020 IEEE 28th International Requirements Engineering Conference (RE). pp. 169–179. IEEE (2020)
17. Honnibal, M., Montani, I.: spacy 2: Natural language understanding with bloom embeddings, convolutional neural networks and incremental parsing. To appear **7**(1), 411–420 (2017)
18. Huang, Z., Xu, W., Yu, K.: Bidirectional lstm-crf models for sequence tagging. arXiv:1508.01991 (2015)
19. Hubert, M., Rousseeuw, P.: International encyclopedia of statistical science (2010)
20. Jindal, R., Malhotra, R., Jain, A.: Automated classification of security requirements. In: 2016 International Conference on Advances in Computing, Communications and Informatics (ICACCI). pp. 2027–2033. IEEE (2016)
21. Kingma, D.P., Ba, J.: Adam: A method for stochastic optimization. arXiv preprint arXiv:1412.6980 (2014)
22. Koch, G., Zemel, R., Salakhutdinov, R., et al.: Siamese neural networks for one-shot image recognition. In: ICML deep learning workshop. vol. 2, p. 0. Lille (2015)
23. Loshchilov, I., Hutter, F.: Fixing weight decay regularization in adam (2018)
24. Pennington, J., Socher, R., Manning, C.D.: Glove: Global vectors for word representation. In: Proceedings of the 2014 conference on empirical methods in natural language processing (EMNLP). pp. 1532–1543 (2014)
25. Reimers, N., Gurevych, I.: Sentence-bert: Sentence embeddings using siamese bert-networks. arXiv preprint arXiv:1908.10084 (2019)
26. Runeson, P., Höst, M.: Guidelines for conducting and reporting case study research in software engineering. Empirical Software Engineering **14**(2), 131–164 (2009)
27. Saadatmand, M., Enoiu, E.P., Schlingloff, H., Felderer, M., Afzal, W.: Smartdelta: Automated quality assurance and optimization in incremental industrial software systems development. In: 25th Euromicro Conference on Digital System Design (DSD) (September 2022)

28. Sainani, A., Anish, P.R., Joshi, V., Ghaisas, S.: Extracting and classifying requirements from software engineering contracts. In: 2020 IEEE 28th International Requirements Engineering Conference (RE). pp. 147–157. IEEE (2020)
29. Sanh, V., Debut, L., Chaumond, J., Wolf, T.: Distilbert, a distilled version of bert: smaller, faster, cheaper and lighter. arXiv:1910.01108 (2019)
30. Sun, C., Qiu, X., Xu, Y., Huang, X.: How to fine-tune bert for text classification? In: China national conference on Chinese computational linguistics. pp. 194–206. Springer (2019)
31. Tunstall, L., Reimers, N., Jo, U.E.S., Bates, L., Korat, D., Wasserblat, M., Pereg, O.: Efficient few-shot learning without prompts. arXiv:2209.11055 (2022)
32. Varenov, V., Gabdrahmanov, A.: Security requirements classification into groups using nlp transformers. In: 2021 IEEE 29th International Requirements Engineering Conference Workshops (REW). pp. 444–450. IEEE (2021)
33. Vaswani, A., Shazeer, N., Parmar, N., Uszkoreit, J., Jones, L., Gomez, A.N., Kaiser, Ł., Polosukhin, I.: Attention is all you need. Advances in neural information processing systems **30** (2017)
34. Wang, W., Wei, F., Dong, L., Bao, H., Yang, N., Zhou, M.: Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers. Advances in Neural Information Processing Systems **33**, 5776–5788 (2020)
35. Winkler, J., Vogelsang, A.: Automatic classification of requirements based on convolutional neural networks. In: 2016 IEEE 24th International Requirements Engineering Conference Workshops (REW). pp. 39–45. IEEE (2016)
36. Winkler, J.P., Grönberg, J., Vogelsang, A.: Optimizing for recall in automatic requirements classification: An empirical study. In: 2019 IEEE 27th International Requirements Engineering Conference (RE). pp. 40–50. IEEE (2019)
37. Wu, Y., Schuster, M., Chen, Z., Le, Q.V., Norouzi, M., Macherey, W., Krikun, M., Cao, Y., Gao, Q., Macherey, K., et al.: Google's neural machine translation system: Bridging the gap between human and machine translation. arXiv preprint arXiv:1609.08144 (2016)
38. Zhang, T., Wu, F., Katiyar, A., Weinberger, K.Q., Artzi, Y.: Revisiting few-sample bert fine-tuning. arXiv preprint arXiv:2006.05987 (2020)
39. Zhao, L., Alhoshan, W., Ferrari, A., Letsholo, K.J., Ajagbe, M.A., Chioasca, E.V., Batista-Navarro, R.T.: Natural language processing for requirements engineering: A systematic mapping study. ACM Computing Surveys (CSUR) **54**(3), 1–41 (2021)