Requirements Classification for Smart Allocation: A Case Study in the Railway Industry

Sarmad Bashir[1], Muhammad Abbas[1], Alessio Ferrari[2], Mehrdad Saadatmand[1] and Pernilla Lindberg[3]
[1]RISE Research Institutes of Sweden, Västerås, Sweden, {first.last}@ri.se
[2]CNR-ISTI, Pisa, Italy, {first.last}@isti.cnr.it
[3]Alstom, Västerås, Sweden, {first.last}@alstomgroup.com

***Abstract*—Allocation of requirements to different teams is a typical preliminary task in large-scale system development projects. This critical activity is often performed manually and can benefit from automated requirements classification techniques. To date, limited evidence is available about the effectiveness of existing machine learning (ML) approaches for requirements classification in industrial cases. This paper aims to fill this gap by evaluating state-of-the-art language models and ML algorithms for classification in the railway industry. Since the interpretation of the results of ML systems is particularly relevant in the studied context, we also provide an information augmentation approach to complement the output of the ML-based classification. Our results show that the BERT uncased language model with the softmax classifier can allocate the requirements to different teams with a 76% F1 score when considering requirements allocation to the most frequent teams. Information augmentation provides potentially useful indications in 76% of the cases. The results confirm that currently available techniques can be applied to real-world cases, thus enabling the first step for technology transfer of automated requirements classification. The study can be useful to practitioners operating in requirements-centered contexts such as railways, where accurate requirements classification becomes crucial for better allocation of requirements to various teams.**

***Index Terms*—requirements classification, requirements allocation, natural language processing, language models**

I. Introduction

In the domain of large-scale software-intensive embedded systems, such as those running on trains, cars, and airplanes, the platforms to be developed are typically decomposed into multiple interacting sub-systems. This fosters architectural modularity and, in turn, maintainability and adaptability [1], [2]. Developing these complex systems also requires structuring the development teams in a modular way [3]. Having a dedicated team responsible for a particular sub-system or component allows companies to have better control over the process, as the components can be independently implemented, tested, and incrementally integrated.

Embedded software systems responsible for train management and control are one example of complex safety-critical systems that are composed of multiple sub-systems. In the railway industry, products need to comply with various safety standards—e.g., the CENELEC norms [4]—which typically require requirements-centered development processes based on the traditional V-model [5]–[7]. In these contexts, requirements for the overall system are first defined based on the customer's needs. The customer requirements are then refined (*internalized*), and the derived requirements are associated with various sub-systems for which different teams are assigned as responsible. The allocation of requirements is normally performed by a requirements analyst, which can be time-consuming, subject to fatigue effects, and human errors [8]. Since requirements are written in natural language, as common also for other domains [9], [10], analysts can benefit from automated natural language processing (NLP) tools to support allocation.

The problem of automated requirements allocation can essentially be formulated as a text *classification* problem. The literature in this area reveals the use of text classification for different goals in software engineering [10], such as binary classification for identifying requirements and non-requirements [11]–[13], and classification of requirements into functional and non-functional classes [14]–[16]. Some studies also include comparisons of different machine learning (ML) and deep learning (DL) algorithms [17]–[19]. However, only a limited number of studies is dedicated to requirements allocation (e.g., [20], [21]). Furthermore, most related studies on multi-class classification only experiment with public datasets annotated by students or researchers (e.g., PROMISE [22]), and lack application in industry. In addition, with some exceptions [20], existing studies do not focus on *augmenting* the classification with additional useful information that can help requirements analysts in the allocation of requirements. Augmentation is particularly important, as NLP tools are imperfect, and a human-in-the-loop is normally needed to sanitize the output [8].

This study is conducted in close collaboration with Alstom, Sweden (Alstom), a world's leading railway vehicle manufacturing company with customers across the globe. In this paper, we report the development of the REQuirement Assigner—REQA—approach for requirements allocation at Alstom. We first empirically evaluate different transformer-based and traditional classifiers in the studied context. In addition, we leverage lexical clustering to generate information augmentations in relation to the predictions to support well-informed allocation. Evaluation of the proposal on 1,680 requirements at Alstom shows that

Pre-print accepted to the 31st IEEE International Requirements Engineering 2023 conference

Please cite the original paper: https://www.es.mdu.se/publications/6697-

our approach with the SciBERT uncased language model achieves a 67% F1 score. The results further indicate an average F1 score of 76% when considering the most frequent classes with the BERT uncased language model. Furthermore, our approach is able to generate supplementary augmentations with a 76% average F1 score. Within our evaluation, we have also studied (a) the effect of pre-processing on classification performance; (b) the coverage of the vocabulary of our railway-specific requirements in the language models used for representation. Our results show that pre-processing negatively affects the classification performance of transformer-based approaches. The results further indicate that out-of-vocabulary (OOV) words in the selected language models are 65% on average. This calls for developing railway-specific and possibly company-specific language models to further improve the classification performance.

The rest of the paper is organized as follows. Section II briefly introduces the background of natural language processing and text classification. Section III presents the REQA approach for requirements allocation. Section IV presents the evaluation method. Section V presents and discusses the obtained results. Section VI discusses the related work in relation to this paper. Section VII presents the potential threats to validity. Finally, Section VIII concludes the paper with future directions.

II. Background

This section introduces the NLP approaches and algorithms used for allocation via classification.

1) Pre-Processing: Requirements classification using NLP starts with pre-processing the input requirements. A typical pre-processing pipeline can include but is not limited to tokenization, Part-of-Speech (POS) tagging, stop-words removal, stemming, or lemmatization. In this paper, we use classification pipelines both with pre-processing—using all the filters listed above—and without pre-processing.

2) Requirement Representation and Classification: Most NLP approaches do not work on raw text but transform each text unit—in our case, a single requirement—into a feature vector. The vectors can be generated with information retrieval (IR)-based *lexical* approaches, or with *semantic* approaches. After representation, classification of the feature vectors can be performed with different ML or DL techniques.

a) Lexical Approaches: From lexical approaches, we use the *tf-idf* (term frequency-inverse document frequency) representation. With this strategy, each component of the feature vector is associated with a word in the entire vocabulary of the considered requirements corpus. The value of the vector component for a certain requirement is given by the frequency of the word in the requirement, divided by its frequency in the corpus. In our work, the *tf-idf* vectors are reduced via principal components analysis (PCA). Based on our previous experience, we notice that pipelines with *tf-idf* perform better when the dimensionality is reduced [7], [23], [24]. We use the normalized vectors extracted from the *tf-idf* matrix and use PCA to automatically select a subset of features that explains at least 95% of the variance in the dataset. This way, we drop most of the correlated and duplicate features and, in turn, reduce the dimensions of the vectors.

After *tf-idf*-based representation, classification is performed through five classical supervised ML algorithms: Support Vector Machine (SVM), Multi-nomial Naive Bayes (MNB), Logistic Regression (LR), Decision Trees (DT), and Random Forest (RF). The objective of the *SVM* classifier is to find the hyperplane in an n-dimensional space for separating the data points (i.e., the feature vectors) into classes. SVM utilizes a kernel function to convert the input data points into a higher dimensional space, in which linear separation of the data points is possible [25]. The *MNB* classifier is based on the Bayes theorem and classifies a feature vector based on the Maximum A Posterior (MAP) decision rule. Based on the training instances, this supervised classifier establishes the most probable class so as to minimize the probability of miss-classification [26]. *LR* utilizes cross-entropy loss to minimize training error and softmax function to compute the probability distribution to be used to predict the class given the feature vector [27]. The *DT* classifier learns the decision rules inferred from feature vectors used for training to predict the target class. The leaf nodes in the tree are associated with the target classes, and classification is based on comparing the values of the components of the feature vector against learned thresholds at each node [28]. Finally, the *RF* algorithm is based on multiple DTs that operate as an ensemble, where each DT casts its vote, and the target class with the most votes is the prediction from the model.

b) Semantic Approaches: From the *semantic* approaches, we consider multiple state-of-the-art strategies, which are based on neural networks. These are fastText [29] (FT), and transformer-based approaches, based on the well-known BERT architecture, (Bidirectional Encoder Representations from Transformers), proposed by Devlin *et al.* [30]. A fundamental building block of all the semantic approaches is the concept of the Language Model (LM). LMs capture the regularities, morphological, and distributional properties of a language.

With FT, the LM is based on the FT algorithm, which considers the internal structural information of the words by representing each word as a bag of character n-grams. FT learns a distributed representation of n-grams based on their context (i.e., co-occurring n-grams), to maximize the average log probability of n-grams.

The BERT LM is based on a type of neural network called *transformer*. This differently weights the significance of each part of the input text by means of the so-called attention mechanism. During training, BERT randomly masks words in a sentence of the training data (typically large sets of generic documents), and then it tries to predict them. BERT looks in both directions, and it uses the full

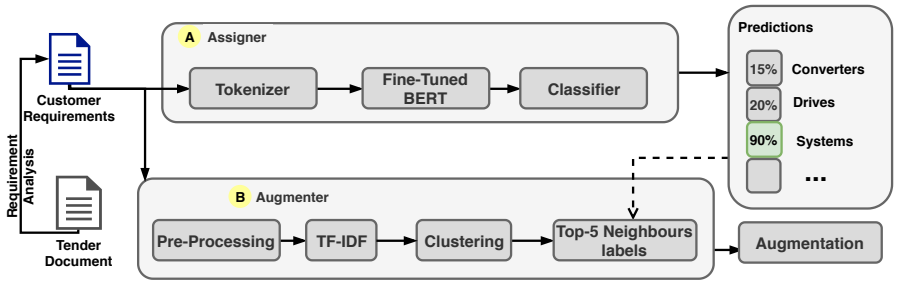

Fig. 1: REQA approach for requirements allocation with augmentation

context of the sentence, both left and right words, in order to predict the masked word. Unlike other LMs, it takes both the previous and next words into account simultaneously. This task is used to learn the LM for requirements representation. The BERT LM is then coupled with a hidden layer of 512 nodes followed by a softmax classifier with n nodes as output (with n equal to the number of possible classes). The classifier is designed to output a probability for each class and is trained on the examples from the requirements dataset. This step is called *fine-tuning*. In other terms, BERT leverages generic documents to first learn the principles of text syntax and semantics and the requirements data to learn the classification task. The BERT variants considered in our paper are BERT base [30], RoBERTa [31], and SciBERT [32]. We do not consider software engineering-specific approaches, namely, such as BERTOverflow [33], which is oriented to named entity recognition, which are substantially different tasks from ours. BERT base is pre-trained on Toronto BookCorpus and English Wikipedia datasets for a total of 16 GB of data. RoBERTa (Robustly Optimized BERT pre-training Approach) is an optimized version of BERT, which uses a slightly different training process. In addition to the BERT datasets, it is also trained on other datasets like CC-News (Common Crawl-News), Open WebText, and others. The total size of these datasets is around 160 GB. Finally, SciBERT is trained on scientific papers from Semantic Scholar. The corpus size is 1.14M papers, about 643GB.

To perform classification, these semantic approaches for text representation are coupled with a statistical classifier. For FT, the classifier is based on the Long Short-Term Memory (LSTM) neural network. FT is coupled with LSTM for classification. This is a recurrent neural network (RNN), which can distinguish relevant and irrelevant information within a sequence of input, to support classification [34]. For BERT-based approaches, the classifier is a softmax layer at the end of the vector representation pipeline.

III. REQA — Approach

The REQA approach aims to support requirements engineers in the allocation of requirements to different teams. Since we address the problem of requirements allocation through classification, we will use the two terms interchangeably. The same holds for the words "team" and "class". The REQA approach is composed of two main modules, `Assigner` and `Augmenter`, as shown in Figure 1.

The `Assigner` module is responsible for generating a representation for the input requirement and for suggesting a possible allocation based on the results of a classification algorithm. We select the best configuration for the `Assigner` in the evaluation section IV, and these are used in the final design. Given a requirement, the `Assigner` outputs a list of potential allocation classes, ranked by likelihood. Only the most probable class is shown to the user, while the ranked list will be used by the `Augmenter`.

Input: Additional brake resistor capacity shall be installed in the event of DC conversion to create dual voltage Units. The DC Brake Resistor shall be fitted on the underframe of the DMS, MS1 and MS2 vehicles. The DC Brake Resistor shall be fully rated.
Ground truth for Allocation: 'Systems'
Assigner prediction: 'Systems'
Generated Augmentation: 3/5, existing similar requirements were allocated to 'Systems'.
Example of Similar Requirement: The Over Voltage / blending resistor modules shall be mounted on the underframe of the DMS, MS1, MS2 and MS3 vehicles.

Input: In the event that the 25kV AC supply is lost, either unexpectedly, or due to traversing a neutral section, then the ACM 400V 3 phase output shall not be interupted while the train is moving. Any minimum operating speed that applies to this requirement shall be stated.
Ground truth for allocation: 'Systems'
Assigner prediction: 'Systems'
Generated Augmentation:

3/5 existing similar requirements were allocated to 'Converters'.

Example of Similar Requirement: The system shall be compatible with a static converter generated supply with the following characteristics: Nominal 3 phase supply, phase to phase: 400Vrms ±10% Nominal single phase supply, phase to neutral: 230Vrms ±10% Nominal frequency: 50Hz ± 2Hz Harmonic distortion: $<$ 10% Maximum peak voltage: $<$ 1000V

Listing 1: Example outputs.

The Augmenter module produces additional information to complement the predictions generated by the Assigner. This additional information helps in providing the most likely classes derived from *lexical* similarity-based measurements. The Augmenter searches for the most similar requirements in the training set used to train the Assigner. Then, it checks whether the classes produced by the Assigner match with the classes of the most similar requirements identified based on lexical features. This can be regarded as a complementary channel to better inform the requirements analyst in deciding the allocation. If the classes proposed by the Augmenter include the class predicted by the Assigner, then one can have higher confidence in the results of the Assigner itself. Otherwise, the requirements analyst needs to consider the prediction more carefully. The examples (shown in Listing 1) above report the output of the two modules in case of a consistent and inconsistent prediction, respectively.

Let us now consider the behavior of the Augmenter in more detail. To produce the output, the Augmenter takes as input: (1) the top-3 classes that are produced by the Assigner, c_1, c_2, c_3; (2) the requirement r, which has been just classified. The input requirement is first preprocessed and represented as a feature vector with *tf-idf* and PCA. The Augmenter then uses the *k-Nearest Neighbor* (kNN) algorithm to find the 5 most similar requirements $r_1, \ldots, r_5$ in the training set that was used to train the Assigner. To perform this step more efficiently, the Augmenter first partitions the search space. This is done by preprocessing and vectorising all the requirements in the training set with *tf-idf* and PCA, and then by applying *k-means* clustering to partition this space. Once clustering has been performed, given the requirement r, the *k-means* algorithm—in the so-called *query* or *inference* mode—returns the cluster that is closer to r. The search for the most similar requirements $r_1, \ldots, r_5$ is performed only within this cluster. Based on the ranked prediction of the Assigner, the classes c_1, c_2, c_3 are scanned, and the Augmenter produces an augmentation together with an example similar requirement. If no class in c_1, c_2, c_3 is found within the classes of $r_1, \ldots, r_5$, no augmentation is generated. The generated augmentation contains c_1, if c_1 is found also within the classes of $r_1, \ldots, r_5$. Otherwise, the augmentation contains c_3 if it appears more than once within the classes of $r_1, \ldots, r_5$, or else c_2. This order was decided based on preliminary trials using the training set. The augmenter also reports the fraction of requirements among $r_1 \ldots r_5$ that belong to the class of the augmentation.

It is important to remark that Assigner and Augmenter are not two independent modules, as the latter considers solely the top-3 classes produced by the former to provide its recommendation. In principle, we could have used as output the ranked list of classes produced by the Assigner, and let the requirements analyst select the right class. However, this was not considered sufficiently useful by the company, who required more reliable suggestions. The Augmenter module was thus introduced to increase the confidence in the output of the Assigner, or to suggest a better reflection on its output when disagreements occur with the Augmenter. In addition, the Augmenter's output is also more interpretable, as it depends on lexical features [16].

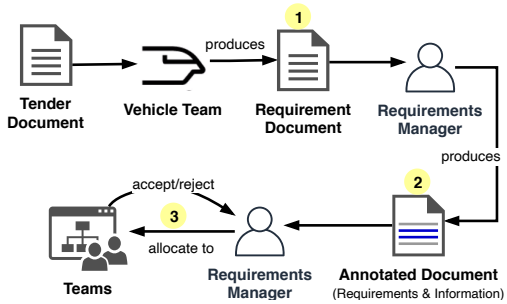

Fig. 2: Requirements flow in the studied context

For *k-means* clustering, the optimal k is determined with the elbow method. For kNN, we select the five nearest neighbors to have a manageable size of similar requirements. Only the top-3 classes of the Assigner are considered to avoid information overload. These values have been tuned through preliminary trials.

IV. Evaluation

Our evaluation is conducted by means of a case study, following the guidelines of Runeson *et al.* [35] for reporting.

A. Study Context

This study is conducted in close collaboration with Alstom, a railway vehicle manufacturing company with sites around the globe. Developing a railway vehicle requires the development of several sub-systems—such as alarm control, door control, traction, and propulsion control—that must comply with different regulatory, regional, and customer requirements. When agreed-upon customer specifications for a vehicle are derived, they must be assigned to their respective teams—often geographically distributed—for implementation and rigorous testing. The assignment of the requirements is done using a requirements management tool, and the respective team might accept or reject the allocation. If teams are assigned requirements unrelated to their teams, the allocation is rejected. This creates longer feedback cycles, and therefore, Alstom is looking for smart approaches to assist engineers in allocation. This is expected to reduce human errors as well. Below, we discuss a typical flow of requirements in Alstom.

Many of the projects at Alstom start with an answer to a call for tender. As shown in Figure 2, after the successful project acquisition, the vehicle team internalizes and extracts potential requirements from the tender documents [36]. The documents typically contain chunks of text, normally in English, which includes requirements and supplementary

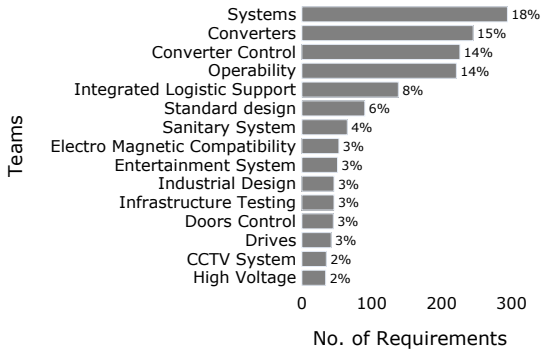

Fig. 3: Requirements distribution over teams in the data

TABLE I: Datasets

Dataset	Reqs.	AW	APP	Train	Test
Complete data	1680	32	17	1344	336
Top-5 labels data	1124	31	17	900	224

* AW= Avg. words, APP= Avg. words in pre-processed reqs.
Train= Training requirements, Test= Test requirements

information to explain the requirements (1). A typical document consists of 600 to 1800 individual chunks of text. Manual labeling is performed by the requirements analyst to distinguish between requirements and information. This produces an annotated requirements document (2). After requirements are identified, they are assigned to 15+ teams for implementation and testing. This study is focused on improving step (3) to assist the requirements analyst in allocation and possibly reduce the rejection of allocations from the teams.

B. Research Questions

Our goal is to first identify the best ML pipelines to support accurate allocation. The best pipelines will be used as part of the `Assigner`. As different ML algorithms may perform differently depending on the use of preprocessing, we also want to analyze the impact of preprocessing on the performance. In addition, since the observed performance for BERT-based approaches—which are the ones achieving the best results—could depend on the *coverage* of the requirements-specific vocabulary by the LMs, we also investigate whether a relationship exists between vocabulary coverage and performance. Finally, based on the best pipelines identified, we want to measure the performance of the `Augmenter` in terms of consistent recommendations and misleading ones. According to the rationales above, we formulate the following research questions (RQs).

- **RQ1.** *Which requirements classification pipeline yields the most accurate results on our data?*
- **RQ2.** *What is the impact of pre-processing on the results?*
- **RQ3.** *What is the coverage of our domain's vocabulary in BERT-based language models?*
- **RQ4.** *What is the performance of the augmentations generated by the approach?*

C. Data collection & Preparation

We had access to six recent and representative requirements documents at the company. The documents already had ground truth on team allocations of the requirements. In addition, all the non-requirement text in the document was also tagged. We prepared all the requirements (excluding non-requirements) from the six documents using the following steps. First, we removed duplicates across all documents. Then, we found some requirements being assigned to more than one team. As these cases were around 15%—low in number for training multi-label classifiers—we considered only the top team as the label after agreement with the industrial partner. As shown in Table I in the Reqs. column, a final set of 1,680 requirements was reached. All the 1,680 requirements were allocated to 15 teams, as shown in Figure 3. For cross-validation of the potential results, we created five random stratified folds, selecting 20% (336 requirements) of the data as a test set. As shown in Figure 3, our dataset is imbalanced since around 70% of the requirements in our data are allocated to the top-5 teams. Classification pipelines typically perform worse on imbalanced data. Therefore, as shown in the last row of Table I, we also considered the requirements allocated to top-5 teams only. The folds for top-5 teams also resulted in around 20% of the data (224 requirements) in the test set.

D. Experimental Setup

To answer the RQs, we considered all the approaches listed in Section II. We also included a random pipeline for comparison. Note that for each pipeline, we use the recommended best configuration for a fair comparison [37], [38]. Below, we detail all the pipelines selected and implemented for the evaluation.

a) Weighted Random (W. Rand.): this pipeline is configured to generate a random number between 1 and 100 with equal probability. We extract ranges for each team in the training set based on their frequency of occurrence in descending order. The generated random number is mapped to a team based on the extracted ranges. For example, team 'Systems' with the frequency of occurrence in allocation is 18%, and if the generated random number is between 1 and 18, then the input requirement from the test set will be allocated to team 'Systems'. For the next team, 'Converters', the allocation frequency of 'Converters' is 15%, and if the generated random number is between 19 and 33, an allocation is predicted for team 'Converters', and so on.

b) Lexical Approaches: include traditional ML-based algorithms for text classification described in Section II. For the selection of the optimal hyperparameters, we apply random multi-search optimization [38]. Each algorithm is implemented in a separate pipeline with optimal hyperparameters. Typically, ML algorithms like SVM and LR are binary classifiers. However, in our case, we applied meta-strategy *One-vs-Rest* to perform multi-class classification. We applied multiple settings to get the best results from ML algorithms. SVM and LR perform better on

evaluation metrics when trained with normalized *tf-idf* vectors, reduced with PCA. The rest of the ML pipelines—DT, RF, and MNB—achieved better results with normalized *tf-idf* vectors without PCA applied.

c) Semantic Approaches: include the ones listed in Section II. For the evaluation of LSTM's network performance on our data, we used custom-trained FT's skip-gram model for requirement representation. To train the FT model, we get the best results when the word embedding dimension is set to 100, window size—maximum distance between a current and predicted word within a sequence—is set to 3, and with 20 epochs. We employ a two-layered LSTM network for training on the custom FT embeddings. We found the optimal hyperparameters using Adam optimizer [39] with a learning rate set to 0.001 with an objective set to reduce the training loss function. We employ a dropout layer after every LSTM layer with a dropout rate of 0.1, which randomly drop units with their connections to prevent over-fitting. Additionally, maximum sequence, batch, and epochs are set as 128, 32, and 10, respectively.

To prepare the dataset for fine-tuning of BERT-based classifiers, we utilize a BERT *WordPiece* tokenizer to split the words in a requirement either into one word per token or into word pieces—where one word is split into multiple tokens. For input representation, the tokenizer encodes *input-ids*—token indices for each requirement—, *pad* and *truncate* requirements with respect to the defined maximum requirement length. Furthermore, the BERT tokenizer prepends a [CLS] token at the start of each requirement and appends a [SEP] token at the end to indicate the end of the requirement. The additional tokens represent a requirement as a single vector. We fine-tuned the BERT models with multiple hyperparameter configurations, as follows. To optimize the weights during fine-tuning of the BERT network, we used AdamW-optimizer—an adaption of Adam—with weight decay set to 0.01. To avoid catastrophic forgetting of BERT pre-trained knowledge, instead of aggressive learning rates, we used a maximal learning rate of 2e-05. We tried multiple batch sizes (16, 32, 64). Experiments show that a batch size of 16 performed best across all runs. A practical maximum sequence size of 128 is used across all the BERT pipelines due to computational constraints. We set the epochs to 10. The intuition behind selecting a relatively high number of epochs on a small dataset is because of the fact that BERT's common one-size-fits-all (three-epochs) practice for fine-tuning is sub-optimal and needs more training time for stabilization of the training network [37]. Some studies in the literature opted for an even bigger number of epochs (e.g., [15]), but we argue it might cause over-fitting.

For the `Augmenter`, we generated *tf-idf* vectors and applied PCA. For computing the optimal k for *k-means*, we use the elbow method. We plot the derivatives of the distortions across k, with k ranging from 1 to 10. An optimal k based on the elbow point was detected at $k = 3$ for all runs.

We performed *stratified* 5-fold cross-validation for the 15 classes, and for the top-5 classes. We implemented and executed the pipelines both with and without pre-processing.

E. Evaluation Metrics

We use the standard metrics, Precision (P), Recall (R), F1 score, and Accuracy (A) to evaluate the performance of different requirements classification approaches with and without pre-processing (**RQ1** and **RQ2**). Since our dataset appears to be highly imbalanced, we report both macro and weighted average scores. To answer **RQ3**, we report the proportion of the out-of-vocabulary words in each BERT-based LM in relation to the weighted F1 observed for classification. Finally, to answer **RQ4**, we also use P, R, F1, and A, considering a prediction to be correct when the generated augmentation belongs to the true class of the requirement r to be allocated. Basically, we evaluate `Assigner` and `Augmenter` independently, and we arguably consider the output of the `Augmenter` as correct also when this correctly predicts the true class, but it disagrees with the `Assigner`. Indeed, in these cases, the requirements analyst can still find the true class as part of the `Augmenter`'s output and take a decision.

F. Implementation

For the implementation of the experimental setup, we used Python. We use transformers library to fine-tune BERT LMs for the downstream classification task, Scikit-learn [40] for traditional ML approaches, NLTK [1] and SpaCy [2] for text pre-processing.

V. Results & Discussion

Table II and Table III show the results of the stratified 5-fold cross-validation for the complete data and the top-5 most frequent classes, respectively. Pipeline names starting with 'p' indicates that pre-processing was applied. The setup column shows the setup of the pipeline execution. In both tables, we show the best-performing pipeline in bold and with an asterisk. In the following, we discuss the best results based on *weighted* measures, as done by other studies [14], [15]. For completeness, we also report macro averages.

RQ1. As shown in Table II, SciBERT outperforms all other pipelines in terms of weighted F1=.67, followed by BERT cased and uncased (F1=.66). Best weighted P and R are also similar for the two pipelines. The effectiveness of SciBERT could be explained by its large vocabulary compared to other BERT-based pipelines. In general, it is rather clear that semantic approaches, excluding LSTM, have an average performance that is higher with respect to lexical ones when it comes to unbalanced datasets, as the one that we obtain when considering all the 15 classes. Results do not change when considering the macro-average. One exception is SVM, which performs well in terms of macro P=.73, though it suffers from poor R rates. This indicates

[1]Available online, https://www.nltk.org/

[2]Available online, https://spacy.io/usage/v2

TABLE II: Pipelines performance on all the requirements

Pipeline	Setup	Weighted Average			Macro Average			Avg. A.
		P	R	F1	P	R	F1	A
W. Rand.		.11	.11	.11	.07	.07	.07	.11
SVM	Norm., PCA	.65	.62	.60	**.73**	.53	.58	.64
pSVM	Norm., PCA	.66	.64	.62	.72	.56	.60	.65
MNB	Norm.	.56	.53	.47	.55	.31	.32	.52
pMNB	Norm.	.54	.54	.48	.50	.31	.32	.54
DT	Norm.	.48	.46	.46	.50	.46	.47	.46
pDT	Norm.	.49	.48	.48	.50	.46	.46	.48
LR	Norm., PCA	.64	.59	.56	.71	.43	.48	.59
pLR	Norm., PCA	.65	.61	.58	.73	.47	.51	.61
RF	Norm.	.61	.58	.57	.69	.52	.57	.58
pRF	Norm.	.61	.59	.58	.69	.54	.58	.59
SciBERT*	uncased	**.68**	.67	**.67**	.71	**.66**	**.67**	.67
pSciBERT	uncased	.64	.64	.63	.69	.61	.63	.64
RoBERTa	base	.66	.66	.65	.70	.64	.65	.66
pRoBERTa	base	.61	.61	.58	.64	.54	.55	.61
BERT	base, cased	.67	.67	.66	.69	.63	.64	.67
pBERT	base, cased	.65	.64	.62	**.73**	.57	.60	.64
BERT	base, uncased	**.68**	**.68**	.66	**.73**	.63	.65	**.68**
pBERT	base, uncased	.67	.66	.64	.71	.62	.64	.66
LSTM	FT custom	.53	.50	.49	.48	.43	.43	.50
pLSTM	FT custom	.58	.57	.56	.57	.54	.53	.57

TABLE III: Pipelines performance on the subset of requirements allocated to top-5 most frequently occurring teams

Pipeline	Setup	Weighted Average			Macro Average			Avg. A.
		P	R	F1	P	R	F1	A
W. Rand.		.21	.21	.21	.20	.20	.20	.21
pSVM	Norm., PCA	.76	.74	.74	.78	.73	.75	.75
pMNB	Norm.	**.77**	.73	.73	**.80**	.70	.72	.73
pDT	Norm.	.61	.60	.60	.61	.60	.61	.60
pLR	Norm., PCA	.76	.74	.74	.79	.72	.74	.74
pRF	Norm.	.70	.68	.68	.72	.67	.68	.68
SciBERT	uncased	.76	.75	.75	.77	.75	.76	.75
RoBERTa	base	.76	.75	.75	.77	.75	.76	.75
BERT	base, cased	**.77**	.76	**.76**	.79	.76	**.77**	.76
BERT*	base, uncased	**.77**	**.77**	**.76**	.78	**.77**	**.77**	**.77**
pLSTM	FT custom	.68	.65	.65	.67	.65	.64	.66

TABLE IV: Vocabulary coverage for the BERT pipelines

Model	Vocab.	UW	OOV	W. F1
SciBERT uncased	31090	6919	4057 (59%)	.67
RoBERTa cased	50265	7618	6041 (79%)	.65
BERT cased	28996	7618	4720 (62%)	.66
BERT uncased	30522	6919	4079 (59%)	.66

UW= Unique Words in Data, OOV= Out of Vocabulary

that false positives are limited for SVM, even for the less represented classes, whose contribution is emphasised with macro averages. Overall, SVM-based pipelines also outperform most lexical pipelines. This is in line with the results presented in the literature (e.g., [14]).

Based on the subset of top-5 teams' data, shown in Table III, the BERT uncased classifier outperforms all other pipelines (F1=.76). That means that with more balanced data, BERT uncased could be a more appropriate choice. Nevertheless, it is also tightly followed by RoBERTa and SciBERT (F1=.75). Among the lexical approaches, pSVM has the best performance (F1=.74). Large language models often require high-end hardware for fine-tuning, and therefore, we argue that SVM-based pipelines would be best suited for smaller companies with constrained computational resources.

> ***Answer to RQ1:*** BERT uncased, and SciBERT pipelines outperform all other pipelines for requirements allocation via classification. RoBERTa also performs well when considering the top-5 classes, i.e., a more balanced dataset. However, SVM-based pipelines closely follow semantic approaches in terms of performance.

RQ2. Table II shows the results of all pipelines, both with and without pre-processing. All the lexical pipelines clearly show a slight increase in F1 when pre-processing is applied. LSTM also shows a more substantial increase in performance across all metrics when preprocessing is applied (F1 from .49 to .56). For the BERT family, pre-processing shows a negative impact on the results. A decrease across all metrics could clearly be observed when pre-processing is applied to any of the pipelines from the BERT family. This can be explained by the BERT's use of stop words for

TABLE V: Performance results of Augmenter

Pipeline	Data	Weighted Average			Macro Average			Avg. A.	Cases	
		P	R	F1	P	R	F1	A	✓	X
SciBERT uncased	All	0.68	0.67	0.67	0.70	0.65	0.66	0.67	0.31	0.27
BERT base uncased	Top-5 Teams	0.77	0.76	0.76	0.78	0.76	0.77	0.76	0.44	0.20

contextual learning. Indeed, BERT learns bidirectional representation from requirements by joint conditioning on both left and right contexts. The stop words provide enough contextual information and receive the same attention as other non-stop words.

> ***Answer to RQ2:*** Pre-processing (with stop-words removal and lemmatization) shows a positive impact on the performance of traditional ML-based and LSTM-based pipelines. BERT-based pipelines for requirements allocation show a decrease in performance after pre-processing.

RQ3. Table IV shows all the pre-trained language models that the transformer-based pipelines utilise for fine-tuning. The Vocab. column shows the size of the vocabulary of the pre-trained models; the UW column shows the unique words in our dataset; the OOV column shows the out-of-vocabulary words that exist in our data but aren't found in the pre-trained language models; and the W. F1 column shows the best weighted F1 from Table II.

Based on the data presented in Table IV, around 65% of the unique words in our dataset could not be found in the pre-trained models. The best weighted F1 score is found with the SciBERT and BERT LMs, where most words from our dataset are not OOV (on average 60% OOV words). No huge difference between the results can be observed. This could be explained by how BERT-based LMs handle OOV words using sub-word information. However, the semantic meaning of the in-vocabulary word deteriorates when the word is found as a sub-string in the OOV word [41]. Therefore, clearly, there is a need for industry-specific language models for software engineering-related downstream tasks. There are some models trained on public software engineering-specific data, such as BERTOverflow and CodeBERT, which are, however, not designed for classification (cf. Sect. II). Based on the results, we believe that these models could improve the performance of the approach if they are adapted to the domain and the task. We will assess this hypothesis in future work, keeping in mind that challenges exist when adapting large LMs to specific domains [41].

> ***Answer to RQ3:*** On average, around 65% of the unique words from our dataset are out of the vocabulary in the pre-trained language models. This calls for more studies on adapting large language models to industry-specific data.

RQ4. As shown in Table V, we coupled the `Augmenter` with the best-performing pipelines for all classes (SciBERT) and top-5 classes (BERT). On average, the `Augmenter` generated augmentations in 98% of the cases, thereby identifying a dominant class within the ones of the five most similar requirements (not shown in the table). When considering all the classes, the `Augmenter` achieves F1=.67, while for the top-5 classes F1=.76. This suggests that, independently of the prediction of the `Assigner`, the `Augmenter` will provide a useful suggestion in a substantial number of cases. It is worth noting that this does not mean that the `Augmenter` has comparable performance with the `Assigner`, since the output of the `Augmenter` *depends* on the top-3 classes produced by the `Assigner`. In other words, the `Augmenter` cannot function independently of the `Assigner`.

When the `Assigner` incorrectly predicts the class, the `Augmenter` disagrees and outputs the right class in 31% of the cases (all classes) and in 44% of the cases (top-5 classes)—cf. ✓column. This further supports the potential usefulness of the `Augmenter`. On average, an agreement, but with incorrect predictions, is observed in 23% of the cases (X column). Reducing these cases is particularly important. Indeed, while in case of disagreement, the analyst must think better about the tool's output, in case of *incorrect* agreement, the analyst could be misled in the allocation. In the studied context, generating additional augmentations is favored by engineers as it is expected to lead to well-informed allocation. However, a rigorous evaluation of the approach in a human-in-the-loop scenario is needed and is planned as part of future work.

> ***Answer to RQ4:*** The augmentations produce correct recommendations with weighted F1=.67 (all classes) and weighted F1=.76 (top-5). On average, the Augmenter achieved an accuracy of around 71%. When wrong predictions are produced by the `Assigner`, correct predictions are produced by the `Augmenter` in 31% of the cases (all classes) and in 44% of the cases (top-5).

VI. Related Work

This paper is concerned with requirements allocation to different teams through automated classification. In the broad field of NLP for requirements engineering (RE), *classification* is the second most common task, right after defect detection, according to the mapping study on NLP for RE by Zhao *et al.* [10]. Furthermore, the systematic literature review of Binkhona and Zhao [42], specifically

focused on automatic requirements classification, identifies about 24 different approaches and 16 different algorithms used. Most of the studies provide solutions to distinguish between functional (FRs) and non-functional requirements (NFRs, or "quality"), and especially considering different NFR classes, such as security, performance, usability, *etc.* [14], [15], [19], [43]–[47]. Within this group, several studies compare multiple language representations, and existing ML algorithms [16]–[18], [48], [49]. A limited set of contributions also aim to identify different classes of FRs, e.g., to facilitate allocation and definition of architectural models [20], [21]. In the following, we will summarise prominent works in the three groups. Though there are also works focusing on the classification of *security* requirements(e.g., [50], [51]), and on distinguishing between requirements and other types of information [11]–[13], these are not discussed here, as their task is substantially different from ours.

a) Classification of FRs vs. NFRs, and NFRs sub-classes: One of the earliest contributions in this group is the study by Cleland-Huang *et al.* [43], [44], which proposes to use a set of key-terms to identify different classes of NFRs. This study introduces the PROMISE NFR dataset [22], which has been widely used as a benchmark by the research community. More recently, Kurtanović and Maalej [14] apply SVM to requirements classification on the same dataset. They select characterising meta-data, lexical, and syntactical features and then apply SVM, achieving performance above 90% in terms of precision and recall. Dalpiaz *et al.* [16] reconstruct this study and manually label 1,500+ requirements from the PROMISE dataset and others. They use SVM to support classification, using *interpretable* linguistic features, as the explainability of automated algorithms is considered particularly important in RE. More recent studies introduce the use of DL methods to bypass the time-consuming task of feature engineering, which affects classical ML methods. Among the works using DL, it is worth mentioning the use of convolutional neural networks (CNN) by Navarro *et al.* [46] together with language representations based on pre-trained word2vec embeddings of the words found in the requirements. Aldhafer *et al.* [47] proposes the usage of another DL approach, namely Bidirectional Gated Recurrent Neural Networks (BiGRU), to classify requirements using their raw text. The advent of *transfer learning* has seen the widespread usage of the BERT model [30] for language representation and classification. BERT is a generic language model trained on a vast amount of domain-generic natural language texts, which is then fine-tuned with additional training examples from a specific downstream task—in our case, requirements classification. Among others, BERT is used by Hey *et al.* [15], achieving state-of-the-art performance, similar to Kurtanović and Maalej [14], on the PROMISE dataset.

b) Comparative Works: Slankas *et al.* [17] present one of the earliest comparative studies in which different *supervised* algorithms, namely SVM, KNN, and Naive Bayes (NB) are compared on the PROMISE dataset, achieving the best performance with SVM. Later, Abad *et al.* [48] compare different ML techniques, namely Latent Dirichlet Allocation (LDA), Biterm Topic Model (BTM), Hierarchical, K-means, Hybrid and Binarized Naïve Bayes (BNB) to distinguish between different NFRs classes using the PROMISE dataset. Best performance is obtained with BNB. In another work, Mahmoud *et al.* [18] compare different semantic techniques to measure the similarity between requirements to enable the identification of different NFR classes. Unsupervised techniques, namely Latent semantic analysis (LSA) and Normalized Google Distance (NGD) are found to be the most effective similarity measures. As these techniques are unsupervised and require no labeled datasets for training. Focusing on requirements *representation* impact, Amasaki *et al.* [49] compare five vector-based representation methods (Term Frequency (tf), tf-idf, Sparse Composite Document Vectors, word2vec and Doc2Vec) and four supervised classification methods, i.e., Logistic Regression (LR), NB, SVM, and RF. When these methods are used to classify FRs *vs.* NFRs, SVM performs well regardless of representation methods. Still, on requirements representation, the recent work of Alhoshan *et al.* [19] compares multiple language models, and use the *unsupervised* zero-shot learning approach on the PROMISE dataset, achieving performance that is comparable with other supervised methods [14], [15].

c) Requirements Allocation: Studies on the classification of requirements into sub-classes to facilitate allocation and architecture decomposition—a similar task to ours—are limited in the literature. Among them, Casamayor *et al.* [21] compares the performance of different clustering algorithms, namely Expectation–maximization (EM), COBWEB, X-Means, and DBSCAN (Density-Based Spatial Clustering of Applications with Noise) on three sample use case-based requirements specifications, achieving best results ($F1 > 80\%$) with EM. Another set of studies is dedicated to *issue* (typically bug) allocation to developers or teams in issue tracking systems [20], [52]. Among them, Aktas *et al.* [20] report the development of an SVM-based classifier and its application to an industrial case in the finance domain, achieving an F1 score of 80%. On the same task, Jonsson *et al.* [52] used an ensemble learner named Stacked Generalization (SG) on data from two different companies, achieving an accuracy of 89%.

Contribution: Compared to works on the classification of NFRs and comparative studies, we also use transfer learning techniques as the most recent works [15], [19]. However, our study differs in terms of *focus*, in that we aim to classify requirements into subsystem-related categories to enable allocation, as in the last group of works discussed in this section. With respect to Casamayor *et al.* [21], we use a substantially larger dataset and more advanced state-of-the-art algorithms. Compared to studies on issue assignment [20], [52], we work with different artifacts, i.e., natural language requirements. This is also one of the few works using an industrial dataset for requirements

classification. Furthermore, our proposal also generates additional augmentations to support informed allocation.

VII. Threats to Validity

Construct Validity. We cast the requirements allocation problem as a multi-class text classification problem. Initially, our dataset contained some requirements that were allocated to multiple teams. We did not account for requirements that were allocated to multiple teams and only limited the allocation to a single team, i.e., a one-to-one assignment. This was done after a discussion with the company. As the number of requirements allocated to multiple teams is rare, we argue that those cases would be insufficient for training multi-label multi-class classifiers. In line with the literature, precision, recall, F1, and accuracy are used to assess the performance of the two modules of the proposed approach.

Internal Validity To mitigate potential internal validity threats, we followed the recommended hyperparameters setting for the pipelines [37], [38]. Additionally, we used standard libraries for the implementation and for computing the metrics. Furthermore, we designed the pipelines in varying configurations and selected the best-performing one as a final candidate for evaluation. Finally, we include authors from diverse backgrounds in validating the study design and experiment setup. Nevertheless, typically machine learning-based pipelines include a higher degree of randomness, and therefore, results may vary in different execution of the same pipelines. To account for the randomness in approaches, we used five-fold cross-validation.

External Validity Our results are based on data from one big railway manufacturing company. Furthermore, we only have limited access to six documents containing 1,680 requirements. While we argue that the six documents were a good representative of requirements documents at the company and that Alstom is a representative of the railway industry, the results might not generalize beyond our studied context. Nevertheless, in the lens of the guidelines for case-based generalization [53], our results might be applicable to similar domains, e.g., railway and automotive industries.

VIII. Conclusion and Future directions

Requirements allocation is a typical task in large distributed companies with multiple teams, which can, in principle, be supported by automated classification approaches. However, there is a lack of empirical evidence on using classification for requirements allocation in industrial contexts. In this paper, we presented the REQA approach and its evaluation at Alstom, a railway company. Our approach makes use of transfer learning-based language models to assign requirements to various teams at the company. Furthermore, it also uses traditional lexical clustering to generate useful supplementary information to enable more informed allocation.

In the future, we aim to qualitatively evaluate our approach and especially the augmentation components with users from Alstom. We also plan to explore the domain adaptability (re-training and its impact) of large transformer-based language models on downstream tasks, including requirements allocation and retrieval.

Acknowledgement. This work is partially funded by the AIDOaRt (KDT) and SmartDelta [54] (ITEA) projects.

References

[1] R. W. Selby, "Enabling reuse-based software development of large-scale systems," *IEEE Transactions on Software Engineering*, vol. 31, no. 6, pp. 495–510, 2005.

[2] D. Šmite, C. Wohlin, T. Gorschek, and R. Feldt, "Empirical evidence in global software engineering: a systematic review," *Empirical software engineering*, vol. 15, no. 1, pp. 91–118, 2010.

[3] J. Bosch, "Software product lines: organizational alternatives," in *Proceedings of the 23rd International Conference on Software Engineering. ICSE 2001.* IEEE, 2001, pp. 91–100.

[4] "EN 50128 – Railway applications – Communication, signalling and processing systems – Software for railway control and protection systems," CENELEC, June 2011.

[5] M. Abbas, R. Jongeling, C. Lindskog, E. P. Enoiu, M. Saadatmand, and D. Sundmark, "Product line adoption in industry: An experience report from the railway domain," in *Proceedings of the 24th ACM Conference on Systems and Software Product Line: Volume A - Volume A*, ser. SPLC '20. New York, NY, USA: ACM, 2020.

[6] A. Ferrari, A. Fantechi, S. Gnesi, and G. Magnani, "Model-based development and formal methods in the railway industry," *IEEE software*, vol. 30, no. 3, pp. 28–34, 2013.

[7] M. Abbas, A. Ferrari, A. Shatnawi, E. Enoiu, M. Saadatmand, and D. Sundmark, "On the relationship between similar requirements and similar software: A case study in the railway domain," *Requirements Engineering*, vol. 28, no. 1, pp. 23–47, 2023.

[8] D. M. Berry, "Empirical evaluation of tools for hairy requirements engineering tasks," *Empirical Software Engineering*, vol. 26, no. 6, pp. 1–77, 2021.

[9] M. Kassab, C. Neill, and P. Laplante, "State of practice in requirements engineering: contemporary data," *Innovations in Systems and Software Engineering*, vol. 10, no. 4, pp. 235–241, 2014.

[10] L. Zhao, W. Alhoshan, A. Ferrari, K. J. Letsholo, M. A. Ajagbe, E.-V. Chioasca, and R. T. Batista-Navarro, "Natural language processing for requirements engineering: A systematic mapping study," *ACM Computing Surveys (CSUR)*, vol. 54, no. 3, pp. 1–41, 2021.

[11] J. Winkler and A. Vogelsang, "Automatic classification of requirements based on convolutional neural networks," in *2016 IEEE 24th International Requirements Engineering Conference Workshops (REW)*. IEEE, 2016, pp. 39–45.

[12] S. Abualhaija, C. Arora, M. Sabetzadeh, L. C. Briand, and M. Traynor, "Automated demarcation of requirements in textual specifications: a machine learning-based approach," *Empirical Software Engineering*, vol. 25, no. 6, pp. 5454–5497, 2020.

[13] A. Sainani, P. R. Anish, V. Joshi, and S. Ghaisas, "Extracting and classifying requirements from software engineering contracts," in *2020 IEEE 28th International Requirements Engineering Conference (RE)*. IEEE, 2020, pp. 147–157.

[14] Z. Kurtanović and W. Maalej, "Automatically classifying functional and non-functional requirements using supervised machine learning," in *2017 IEEE 25th International Requirements Engineering Conference (RE)*. Ieee, 2017, pp. 490–495.

[15] T. Hey, J. Keim, A. Koziolek, and W. F. Tichy, "Norbert: Transfer learning for requirements classification," in *2020 IEEE 28th International Requirements Engineering Conference (RE)*. IEEE, 2020, pp. 169–179.

[16] F. Dalpiaz, D. Dell'Anna, F. B. Aydemir, and S. Çevikol, "Requirements classification with interpretable machine learning and dependency parsing," in *2019 IEEE 27th International Requirements Engineering Conference (RE)*. IEEE, 2019, pp. 142–152.

[17] J. Slankas and L. Williams, "Automated extraction of non-functional requirements in available documentation," in *2013 1st International workshop on natural language analysis in software engineering (NaturaLiSE)*. IEEE, 2013, pp. 9–16.

[18] A. Mahmoud and G. Williams, "Detecting, classifying, and tracing non-functional software requirements," *Requirements Engineering*, vol. 21, no. 3, pp. 357–381, 2016.

[19] W. Alhoshan, A. Ferrari, and L. Zhao, "Zero-shot learning for requirements classification: An exploratory study," *Information and Software Technology*, p. 107202, 2023.

[20] E. U. Aktas and C. Yilmaz, "Automated issue assignment: results and insights from an industrial case," *Empirical Software Engineering*, vol. 25, no. 5, pp. 3544–3589, 2020.

[21] A. Casamayor, D. Godoy, and M. Campo, "Functional grouping of natural language requirements for assistance in architectural software design," *Knowledge-Based Systems*, vol. 30, pp. 78–86, 2012.

[22] J. Cleland-Huang, S. Mazrouee, H. Liguo, and D. Port, "NFR," Mar. 2007. [Online]. Available: https://doi.org/10.5281/zenodo.268542

[23] M. Abbas, A. Ferrari, A. Shatnawi, E. P. Enoiu, and M. Saadatmand, "Is requirements similarity a good proxy for software similarity? an empirical investigation in industry," in *Requirements Engineering: Foundation for Software Quality*, F. Dalpiaz and P. Spoletini, Eds. Cham: Springer International Publishing, 2021, pp. 3–18.

[24] M. Abbas, M. Saadatmand, E. Enoiu, D. Sundamark, and C. Lindskog, "Automated reuse recommendation of product line assets based on natural language requirements," in *Reuse in Emerging Software Engineering Practices*, S. Ben Sassi, S. Ducasse, and H. Mili, Eds. Cham: Springer International Publishing, 2020, pp. 173–189.

[25] K. Crammer and Y. Singer, "On the algorithmic implementation of multiclass kernel-based vector machines," *Journal of machine learning research*, vol. 2, no. Dec, pp. 265–292, 2001.

[26] I. Rish *et al.*, "An empirical study of the naive bayes classifier," in *IJCAI 2001 workshop on empirical methods in artificial intelligence*, vol. 3, no. 22, 2001, pp. 41–46.

[27] D. G. Kleinbaum, K. Dietz, M. Gail, M. Klein, and M. Klein, *Logistic regression*. Springer, 2002.

[28] B. Charbuty and A. Abdulazeez, "Classification based on decision tree algorithm for machine learning," *Journal of Applied Science and Technology Trends*, vol. 2, no. 01, pp. 20–28, 2021.

[29] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *Transactions of the association for computational linguistics*, vol. 5, pp. 135–146, 2017.

[30] 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.

[31] 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.

[32] I. Beltagy, K. Lo, and A. Cohan, "Scibert: A pretrained language model for scientific text," *arXiv preprint arXiv:1903.10676*, 2019.

[33] J. Tabassum, M. Maddela, W. Xu, and A. Ritter, "Code and named entity recognition in StackOverflow," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*. ACL, 2020, pp. 4913–4926.

[34] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[35] P. Runeson and M. Höst, "Guidelines for conducting and reporting case study research in software engineering," *Empirical Software Engineering*, vol. 14, no. 2, 2009.

[36] S. Bashir, M. Abbas, M. Saadatmand, E. P. Enoiu, M. Bohlin, and P. Lindberg, "Requirement or not, that is the question: A case from the railway industry," in *Requirements Engineering: Foundation for Software Quality*, A. Ferrari and B. Penzenstadler, Eds. Cham: Springer Nature Switzerland, 2023, pp. 105–121.

[37] T. Zhang, F. Wu, A. Katiyar, K. Q. Weinberger, and Y. Artzi, "Revisiting few-sample bert fine-tuning," *arXiv preprint arXiv:2006.05987*, 2020.

[38] J. Bergstra and Y. Bengio, "Random search for hyper-parameter optimization," *Journal of Machine Learning Research*, vol. 13, no. 10, pp. 281–305, 2012.

[39] K. Diederik and J. A. Ba, "A method for stochastic optimization. arxiv 2014," *arXiv preprint arXiv:1412.6980*, 2015.

[40] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg *et al.*, "Scikit-learn: Machine learning in python," *Journal of machine learning research*, vol. 12, no. Oct, pp. 2825–2830, 2011.

[41] A. Nayak, H. Timmapathini, K. Ponnalagu, and V. G. Venkoparao, "Domain adaptation challenges of bert in tokenization and sub-word representations of out-of-vocabulary words," in *Proceedings of the First Workshop on Insights from Negative Results in NLP*, 2020, pp. 1–5.

[42] M. Binkhonain and L. Zhao, "A review of machine learning algorithms for identification and classification of non-functional requirements," *Expert Systems with Applications: X*, vol. 1, p. 100001, 2019.

[43] J. Cleland-Huang, R. Settimi, X. Zou, and P. Solc, "Automated classification of non-functional requirements," *Requirements engineering*, vol. 12, no. 2, pp. 103–120, 2007.

[44] J. Cleland Huang, R. Settimi, X. Zou, and P. Solc, "The detection and classification of non-functional requirements with application to early aspects," in *14th IEEE International Requirements Engineering Conference (RE'06)*. IEEE, 2006, pp. 39–48.

[45] A. Casamayor, D. Godoy, and M. Campo, "Identification of non-functional requirements in textual specifications: A semi-supervised learning approach," *Information and Software Technology*, vol. 52, no. 4, pp. 436–445, 2010.

[46] R. Navarro-Almanza, R. Juarez-Ramirez, and G. Licea, "Towards supporting software engineering using deep learning: A case of software requirements classification," in *2017 5th International Conference in Software Engineering Research and Innovation (CONISOFT)*. IEEE, 2017, pp. 116–120.

[47] O. AlDhafer, I. Ahmad, and S. Mahmood, "An end-to-end deep learning system for requirements classification using recurrent neural networks," *Information and Software Technology*, vol. 147, p. 106877, 2022.

[48] Z. S. H. Abad, O. Karras, P. Ghazi, M. Glinz, G. Ruhe, and K. Schneider, "What works better? a study of classifying requirements," in *2017 IEEE 25th International Requirements Engineering Conference (RE)*. IEEE, 2017, pp. 496–501.

[49] S. Amasaki and P. Leelaprute, "The effects of vectorization methods on non-functional requirements classification," in *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 2018, pp. 175–182.

[50] N. Munaiah, A. Meneely, and P. K. Murukannaiah, "A domain-independent model for identifying security requirements," in *2017 IEEE 25th International Requirements Engineering Conference (RE)*. IEEE, 2017, pp. 506–511.

[51] V. Varenov and A. Gabdrahmanov, "Security requirements classification into groups using nlp transformers," in *2021 IEEE 29th International Requirements Engineering Conference Workshops (REW)*. IEEE, 2021, pp. 444–450.

[52] L. Jonsson, M. Borg, D. Broman, K. Sandahl, S. Eldh, and P. Runeson, "Automated bug assignment: Ensemble-based machine learning in large scale industrial contexts," *Empirical Software Engineering*, vol. 21, no. 4, pp. 1533–1578, 2016.

[53] R. Wieringa and M. Daneva, "Six strategies for generalizing software engineering theories," *Science of computer programming*, vol. 101, pp. 136–152, 2015.

[54] M. Saadatmand, E. P. Enoiu, H. Schlingloff, M. Felderer, and W. Afzal, "Smartdelta: Automated quality assurance and optimization in incremental industrial software systems development," in *25th Euromicro Conference on Digital System Design (DSD)*, 2022.