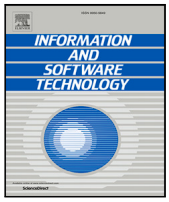
Information and Software Technology 159 (2023) 107202

Contents lists available at ScienceDirect

Information and Software Technology

journal homepage: www.elsevier.com/locate/infsof



Zero-shot learning for requirements classification: An exploratory study

Waad Alhoshan [a,1], Alessio Ferrari [b,*,1], Liping Zhao [c,*,1]

[a] College of Computer and Information Sciences, Imam Mohammad Ibn Saud Islamic University (IMSIU), Riyadh, 11564, Saudi Arabia
[b] Istituto di Scienza e Tecnologie dell'Informazione "A. Faedo", Consiglio Nazionale delle Ricerche (ISTI-CNR), Via G. Moruzzi 1, Pisa, 56126, Italy
[c] Department of Computer Science, University of Manchester, Manchester, M13 9PL, UK

ARTICLE INFO

MSC:
0000
1111

Keywords:
Zero-shot learning
Language models
Contextual word-embeddings
Requirements classification
Zero-shot text classification
Unsupervised learning
Multi-label classification
Transfer learning
Deep learning
Requirements engineering
AI for requirements engineering
AI for software engineering

ABSTRACT

Context: Requirements engineering (RE) researchers have been experimenting with machine learning (ML) and deep learning (DL) approaches for a range of RE tasks, such as requirements classification, requirements tracing, ambiguity detection, and modelling. However, most of today's ML/DL approaches are based on *supervised* learning techniques, meaning that they need to be trained using a large amount of task-specific labelled training data. This constraint poses an enormous challenge to RE researchers, as the lack of labelled data makes it difficult for them to fully exploit the benefit of advanced ML/DL technologies.
Objective: This paper addresses this problem by showing how a *zero-shot learning* (ZSL) approach can be used for requirements classification without using any labelled training data. We focus on the classification task because many RE tasks can be framed as classification problems.
Methods: The ZSL approach used in our study employs contextual word-embeddings and transformer-based language models (LMs). We demonstrate this approach through a series of experiments to perform three classification tasks: (1) FR/NFR — classification functional requirements vs non-functional requirements; (2) NFR — identification of NFR classes; (3) Security — classification of security vs non-security requirements.
Results: The study shows that the ZSL approach achieves an F1 score of 0.66 for the FR/NFR task. For the NFR task, the approach yields F1$\sim 0.72-0.80$, considering the most frequent classes. For the Security task, F1 ~ 0.66. All of the aforementioned F1 scores are achieved with zero-training efforts.
Conclusion: This study demonstrates the potential of ZSL for requirements classification. An important implication is that it is possible to have very little or no training data to perform classification tasks. The proposed approach thus contributes to the solution of the long-standing problem of data shortage in RE.

1. Introduction

In requirements engineering (RE), system and software requirements specifications are typically written in natural language (NL) [1,2]. In the last few years, natural language processing (NLP) techniques based on supervised machine learning (ML), and, more recently, deep learning (DL), have been applied to address several RE tasks, driven by the success of these techniques in a range of domains, including medical diagnosis, credit card fraud detection, and sentiment analysis [3–5].

To date, research on ML-based RE has been primarily focused on *supervised* classification approaches [6], as most RE tasks can be framed as *text classification* problems solved by supervised learning techniques. Relevant examples are: classifying requirements into different categories [7–9]; identifying requirements from software contracts [10]; discerning requirements and non-requirements [11]; identifying miscategorised requirements [12]; and discovering requirements-relevant content from app reviews [13,14].

However, supervised ML has some major limitations. The most notable one is that supervised learners need to be trained on a large amount of task-specific, labelled data before they can be ready for predicting the outcomes on new data [4,15]. This problem is exacerbated in domains like RE where collecting and labelling sufficient training data is often expensive, time-consuming and error-prone [10,16]. Labelling data also requires substantial domain- and even project-specific knowledge [12]. Furthermore, labelled data used in previous studies are frequently unavailable. This happens even for the lively area of app review analysis, in which most studies have not released their labelled dataset, according to a recent survey (cf. [13], p. 34). This limitation hinders RE researchers from exploring different learning techniques.

Another limitation of supervised learning methods is that models can only classify the data belonging to *seen classes* (i.e., classes labelled in the training data), but they cannot classify the data into previously

* Corresponding authors.
E-mail addresses: alessio.ferrari@isti.cnr.it (A. Ferrari), liping.zhao@manchester.ac.uk (L. Zhao).
[1] Equal contribution to the research and writing of the paper.

https://doi.org/10.1016/j.infsof.2023.107202
Received 11 August 2022; Received in revised form 13 March 2023; Accepted 14 March 2023
Available online 20 March 2023

unseen classes (i.e., classes not labelled in the training data) [15]. Although this limitation is inherent in supervised learning, the ability to deal with previously unseen classes can bring huge benefits to many real-world applications where classes are artificially defined, with no common consensus, or where classes may evolve over time, with new classes emerging and old ones becoming obsolete. One such example is requirements classification where several classification schemes exist for non-functional requirements (NFRs) [17,18].

As software applications, their requirements, and the theory of NFRs itself evolves over time, so do the classification schemes. Consequently, a dataset labelled using one set of classes (e.g., the PROMISE NFR Dataset [19]) cannot be reused to train a method that intends to predict a different set of classes (e.g., based on the latest ISO/IEC/IEEE 29148 standard [20]). Each time a new classification scheme is used, the datasets must be relabelled accordingly, incurring expensive data-labelling costs.

To address these problems, different learning paradigms have been proposed in recent years [4]. One such paradigm is *transfer learning* [21], which aims to alleviate the problems of data shortages and expensive data labelling efforts by adapting existing well-trained ML models to different, but related domains or tasks [22]. However, model adaptation still requires thousands or tens of thousands of labelled task-specific instances [23].

More recently, *zero-shot learning* (ZSL) has emerged as a promising paradigm [15]. ZSL directly applies previously trained models to predicting both seen and unseen classes without using any labelled training instances [24,25].

Expanding on our preliminary study [26], this paper aims to conduct an in-depth study of using ZSL for requirements classification and to gain insight into this new paradigm in the context of RE. Whereas our preliminary study only assessed ZSL on the classification of security and usability requirements selected from a portion of the PROMISE NFR dataset [19], this paper substantially extends the previous contribution by evaluating ZSL on different classification tasks, namely differentiation between functional requirements (FRs) and non-functional requirements (NFRs), identification of different NFR classes, and classification of security vs. non-security requirements. These tasks are carried out on two datasets: the full PROMISE NFR dataset and the SecReq dataset [27]. In addition, we have selected four different language models (LMs) to evaluate ZSL, to allow us to compare the performance of ZSL under different models.

The remaining paper is structured as follows: Section 2 briefly reviews the current ML approaches for requirements classification. Section 3 introduces the zero-shot learning approach used in this paper and its related concepts. Section 4 defines the research questions for our study and details our study design. Section 5 analyzes the experimental results, while Section 6 answers our research questions based on these results. Section 7 examines the validity threats to our experiments and our mitigation strategies. Finally, Section 8 concludes the paper.

2. Related work

Most studies in ML-based requirements classification focus on the categorization between functional (FR) and non-functional (NFR, or "quality" [28]) requirements, and on the further categorization of different NFR classes, such as security, performance, usability, *etc.* However, the distinction between FR and NFR has been a matter of debate in the RE community [16,29], and the empirical study by Eckhardt et al. [18] shows that NFRs can include functional aspects. Furthermore, there is a more fine-grained representation of FRs and NFRs given by the ISO/IEC/IEEE 29148:2018(E) Standard [20], which distinguishes between functional/performance, quality, usability, interface, and other classes, thus refining the conceptualization already elaborated by the NFR classification from Glinz [17]. Yet, despite the lack of consensus what NFRs are, and how we should classify and represent them, the differentiation between FRs and NFRs is a common categorization in RE, and in the following we will use this distinction, keeping in mind that it is an artificial construct [18].

ML-based approaches for requirements classification were examined in a systematic literature review by Binkhonain and Zhao [6]. Here we briefly review some closely related representative works.

2.1. Classification of FRs and NFRs

One of the earliest adoptions of ML to RE was due to Cleland-Huang et al. [7], who proposed to use a set of indicator terms to identify different classes of NFR. The approach was supervised, in that it first identified a set of indicator terms on a set of manually annotated requirements, and then used this set to classify unseen cases. The approach achieved a recall up to 0.80, but suffered from low precision, up to 0.21. This study also introduced the PROMISE NFR dataset [19], which has been widely used by the research community, and it is also one of the benchmarks of our work.

To mitigate the problem of dataset annotation, Casamayor et al. [30] proposed a semi-supervised method, based on an iterative process similar to active learning, in which the user provided feedback to the classifier. Their approach used Naive Bayes (NB) as a classification algorithm and the PROMISE NFR dataset as the training set. After multiple iterations in which an increasing number of training examples were used, they obtained a maximum precision of above 0.80 and a maximum recall of above 0.70 on most classes, except underrepresented ones.

Another well-known ML approach is provided by Kurtanović and Maalej [8], who applied Support Vector Machine (SVM) for requirements classification. They selected relevant features with an ensemble of different supervised classifiers and achieved precision and recall up to 0.92 for identifying FRs and NFRs on the PROMISE NFR dataset. For the identification of specific NFRs classes, they achieved the highest precision and recall for security and performance classes with 0.92 precision and 0.90 recall. Dalpiaz et al. [16] reconstructed the study by Kurtanović and Maalej and used the results obtained as a baseline to evaluate their proposed approach using interpretable linguistic features.

To overcome the problem of labour intensive feature engineering, Navarro et al. [31] proposed one of the first approaches using a deep learning (DL) model. They used a CNN (Convolutional Neural Network) model on the PROMISE dataset, and obtained precision and recall of 0.80 and 0.79 respectively, thus addressing the problem of limited precision observed by Cleland-Huang et al.. Similar approaches were proposed by Dekhtyar and Fong [32], and more recently, by Aldhafer et al. [33].

A more closely related work to ours is Hey et al. [9], who proposed NoRBERT, a transfer learning approach for requirements classification. Their approach is based on fine-tuning the BERT model (Bidirectional Encoder Representations from Transformers) [23]. They achieved similar or better results with respect to previous works, achieving 0.92 precision and 0.95 recall for FR vs. NFR classification on the PROMISE dataset. NoRBERT also outperformed recent approaches at classifying NFRs classes. The most frequent classes were classified with precision up to 0.94 and recall up to 0.90. The proposed solution was also applied for the classification of different types of functional requirements concerns in PROMISE, achieving precision up to 0.88 and recall up to 0.95.

2.2. Classification of security requirements

One of the early works on security requirements classification was by Knauss et al. [34], who used a Bayesian classifier to identify security-relevant requirements on three industrial datasets. These datasets are also used in our paper (aggregated into the *SeqReq* dataset). They achieved precision > 0.8 and recall > 0.9. In another work, Riaz

et al. [35] proposed an approach to extract security-relevant sentences from requirements documents. They used a dataset of 10,963 sentences belonging to six different documents from the healthcare domain. The proposed approach was semi-automatic and based on KNN (K-nearest Neighbours) classification. The authors achieved a precision of 0.82, and a recall of 0.79.

Addressing the lack of domain-specific data sets, Munaiah et al. [36] proposed a domain-independent classification model for identifying domain-specific security requirements. The proposed approach, a one-class SVM classifier, was used to identify general descriptions related to software security weaknesses, but not the actual security requirements *per se*, as the classifier was trained using the Common Weakness Enumeration database [37]. The authors showed that the one-class classifier achieved an average precision and recall of 0.67 and 0.70 respectively. Varenov et al. [38] compared the performance of different LMs, namely BERT, XLNET, and DistilBERT, for security requirements classification. They identified 1086 security requirements of seven different classes collected from multiple existing datasets, such as PURE [39], SecReq [34] and Riaz's dataset [35]. Unlike previous studies, the work by Varenov et al. [38] aimed to classify security requirements into more fine-grained classes, i.e., Confidentiality, Integrity, Availability, Accountability, Operational, Access Control, and Other. DistilBERT achieved the best results, with precision of 0.80 and recall of 0.82.

2.3. Our contribution

In comparison with the related work, our study aims to present a comparative analysis of different ZSL configurations for the classification of requirements. Similarly to the proposal of Hey et al. [9], we explore the potential of a DL solution on the widely used PROMISE dataset. Differently from Hey et al. [9], this is the first work in RE that proposes to use ZSL for the classification task. While Hey et al. focus on addressing generalizability of the classifier by means of transfer learning, our proposal: (1) avoids the need of a tagged dataset, therefore addressing the well known problem of the scarcity of annotated datasets in RE [1,12,39,40]; (2) is inherently generalizable to different projects, thus addressing the problem of decreasing performance with unseen projects, which typically affects requirements classifiers [9,16]. Concerning security requirements classification, our proposal overcomes the problem of dataset annotation as Munaiah et al. [36]. However, their approach is specific to security requirements, while our proposal is more generalizable and adaptable to different classification tasks.

3. Zero-shot learning

Zero-short learning is an emerging learning paradigm that aims to perform learning tasks without using training data. ZSL was originally used in image processing to predict unseen images [41], but has recently been adapted to many NLP tasks, including entity recognition [42], relation extraction [43], document classification [44], and text classification [45]. The fundamental idea of ZSL is that some previously trained language models are so accurate that they can be directly applied to predicting new data without any training [24,25]. In this section, we first introduce the concepts of language models and then focus on a specific ZSL approach – embedding-based ZSL – used in our study.

3.1. Language models and transfer learning

Language models (LMs) are deep neural networks for representing words and sentences in natural language. These models are developed to support NLP tasks such as language understanding and inference. Traditional LMs, such as Skip-gram [46], Word2Vec [47], and GloVe [48], use *static word embeddings*, i.e., fixed vector representations, to represent the words and sentences in a text [46]. More recently, *transformer-based LMs*, such as OpenAI GPT [49] and BERT [23], have made significant improvements over the traditional LMs in language representation, as they can capture the deep meaning of words and sentences through *dynamic or contextual word embeddings* [50]. In other words, traditional LMs are *static* as they simply expand the words in a sentence with related ones. For example, the sentence "*This is not about usability*" is mapped onto a vector similar to "*This is about usability*", as the two sentences only differ in one word. By contrast, with transformer-based LMs, the vector representations of these two sentences are different, as they have opposite meanings. This characteristic naturally plays a crucial role in requirements analysis, as requirements sentences often use a very restricted vocabulary [39], but convey different meanings.

Today, pretrained LMs (PLMs) are widely available.[2] These PLMs are typically pretrained for some generic NLP tasks using unlabelled data such as app reviews and Wikipedia, available *en masse* through the Internet [51]. Such LMs can be adapted— i.e., *transferred*—to different downstream tasks [52]. This concept is known as *transfer learning* [53].

There are two approaches for adapting a PLM: *feature-based* and *fine-tuning* [54]. The feature-based approach uses the representations of a pre-trained model as input features to train a downstream task model (e.g., ELMo [55]), while the fine-tuning approach, adopted by OpenAI GPT and BERT, involves modifying the weights and parameters in certain layers of the PLM so as to enable the model to perform a specific NLP downstream task. Transfer learning helps to reduce the cost and effort required for training new models, and allows users to explore different NLP tasks with a relatively small amount of task-specific labelled data.

3.2. Embedding-based zero-shot learning

There are two common approaches to ZSL: *entailment-based* and *embedding-based*. The former treats the classification task as a natural language inference (NLI) task [56], whereas the latter uses language representations to predict if an input text is related to a given class label. More specifically, the entailment-based approach treats an input text sequence as a premise and the labels as a hypothesis, and then infers if the input text is an entailment of any of the labels or not [56]. For example, given the sentence *"the system must be deployed on Azure"* as a premise, and the label string *"this is about software architecture"* as a hypothesis, the entailment-based classifier provides a score which is then translated into one of the following outputs: entailment (yes), contradiction (no), or undecided. This ZSL approach requires a large inference-based PLM[3] that can interpret the entailment relation between an input sequence and a label.

The embedding-based ZSL approach was introduced by Veeranna et al. [57]. Under this approach, both class labels (e.g., "usability", "security") and input text are represented as word sequences using word embeddings. Text classification then involves computing the *semantic similarity* between each label sequence and the text sequence. If the similarity score is greater than a certain threshold, then the text can be classified into a specific category represented by the label; otherwise, the text does not belong to that category. Note that, since a label is treated as a sequence of words, it can contain any number of words or their combinations.

The simplicity of the embedding-based approach to ZSL has led us to applying it to our study. However, unlike the original proposal by Veeranna et al. [57], who used the static word-embedding technique

[2] For example, a large number of PLMs are freely available at Hugging Face website: https://huggingface.co/models.

[3] Hereafter we simply call a "PLM" "LM", as the LMs used in our study are PLMs.

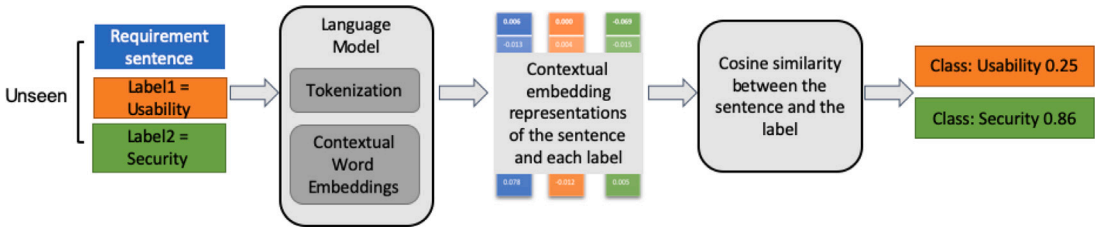


Fig. 1. An illustration of the contextual word embedding-based ZSL approach.

(Skip-gram) for word representation, we take advantage of transformer-based LMs and use them to produce contextual word-embeddings for both labels and input text. In so doing, our embeddings not only capture syntactic and semantic characteristics of the words, but also their context.

Another difference between our approach and the original one is that we do not use similarity thresholds to determine the predicted label; instead, we treat all text classification as a multi-label classification task and rank-order all the labels with their similarity scores. For a binary or multi-class classification task, we select the label with the highest similarity score as the predicted label. For a multi-label task, we check the top-n labels. The usage of similarity thresholds could help identify misclassifications, when the highest similarity has a low value. However, selecting appropriate thresholds is a task *per se*, which we do not consider in this paper.

The *contextual word embedding-based* approach adopted in our study (Fig. 1) can be explained using a simple example: Given a requirement "*CNG shall support mechanisms for secure authentication and communication with the remote management system*" and two class labels as "usability" and "security", we want to find out if the requirement should be classified into the "usability" or the "security" class. ZSL performs this classification task by taking the three word sequences (the requirement statement plus the two labels) as input to a LM to produce three contextual word embeddings. It then compares the requirement with each label using the *Cosine* similarity function.[4] The comparison will return n similarity scores, each score for a label and the requirement pair. The pair with the highest similarity score means that the requirement belongs to the category denoted by its associated label. In our example, ZSL would return two similarity scores: 0.86 for the "security" label and the given requirement, and 0.25 for the "usability" label and the given requirement. Based on these scores, we deduce that the given requirement is a security requirement.

The above example shows that the accuracy of the embedding-based ZSL approach depends highly on the choice of (1) the **labels** and (2) the **PLMs**. Whereas the above example only uses single word labels (i.e., "usability" and "security"), our study will investigate different label configurations. For example, by composing the usability label using a set of synonyms and related words, such as instructive, easy, helpful, useful, learnable, explainable, intuitive, and understandable, the LM can produce a more dynamic embedding that can capture a range of connotations of the usability requirements.

4. Experimental design

To evaluate the effectiveness of embedding-based ZSL for requirements classification, our study aims to answer the following three research questions (RQs):

RQ1: *Which **language model** is more effective for which zero-shot requirements classification task?*

RQ2: *To what extent do different **label configurations** affect the effectiveness of zero-shot requirements classification?*

RQ3: *How **effective** is zero-shot learning for requirements classification compared to related supervised learning approaches?*

We answer these questions through a series of experiments. In this section, we describe our experimental design, consisting of five steps: selection of datasets and tasks; selection of LMs; label configuration; performance measure selection; technical setup of the experiments.

4.1. Dataset and task selection

The following two datasets are selected for our experiments:

- **PROMISE NFR dataset** [19], introduced by Cleland Huang et al. [7]: This dataset contains 625 requirements, partitioned into 255 FRs, and 370 NFRs. The NFRs are further partitioned into 11 different classes, namely: A = Availability (21 requirements), L = Legal (13), LF = Look and feel (38), MN = Maintainability (17), O = Operational (62), PE = Performance (54), SC = Scalability (21), SE = Security (66), US = Usability (67), FT = Fault tolerance (10), and PO = Portability (1). These classes are unevenly distributed, ranging from 67 requirements for Usability to one for Portability. Each one of the most frequent classes—Usability, Security, Operational, and Performance—has more than 50 examples, while the less frequent classes—Fault Tolerance, Legal, Maintainability and Portability—have from one to 17 requirements each. The dataset has been widely used in the literature, e.g., by Kurtanović and Maalej [8], and by Hey et al. [9].
- **SecReq dataset** [27], introduced by Knauss et al. [34]: This dataset contains 510 requirements, made of security-related requirements (187) and non-security related requirements (323). The requirements were collected from three projects: Common Electronic Purse (ePurse), Customer Premises Network (CPN), and Global Platform Spec (GPS). The dataset has been used, e.g., by Varenov et al. [38].

We select the following typical requirements classification tasks for our study:

- **Task FR/NFR** — *Binary Classification of FRs vs. NFRs.* With this task we aim to distinguish FRs from NFRs, assuming that a requirement belongs to either a FR or a NFR class. We use the PROMISE NFR dataset for this task.
- **Task NFR** — *Binary, Multi-class and Multi-label Classification of NFRs.* This task aims to classify different types of NFRs based on the 10 different classes of the PROMISE NFR dataset (we excluded the Portability class as it only has one single sample in the dataset). We perform three sub-tasks to understand how ZSL reacts to different ways of classifying NFRs: (1) binary classification which discerns if a NFR belongs to a particular class or not; (2) multi-class single-label classification (simply, *multi-class classification*) which assigns a NFR to one of the top or all NFR classes; (3) multi-class multi-label classification (simply, *multi-label classification*), which allocates a NFR to one or more NFR

[4] The *Cosine* similarity function is the standard way to compute semantic similarity between a label and a text [47,48,50,58].

classes. The purpose of the third sub-task is to check if the top-*n* NFR classes returned by the ZSL classifier correlate with the assigned NFR label in the dataset.

- **Task Security** — *Binary Classification of security related vs. non-security related requirements.* This task assumes that a requirement belongs only to one of these two classes: security related and non-security related. We use the SeqReq dataset for this task.

These datasets and tasks are selected for our experiments as they are frequently considered in the literature (cf. Section 2) and will enable us to compare our results directly with those obtained by previous work.

4.2. Language model selection

We select the following four BERT-based LMs for our study: two generic and two domain-specific LMs. The two generic LMs are **Sentence-BERT (Sbert)** and **All-MiniLM-L12 (AllMini)**, which are freely available at the HuggingFace website,[5] a well-known NLP community repository that provides open source pretrained LMs and other language resources. The two domain-specific LMs, **Bert4RE** [59] and **BERTOverflow (SObert)** [60], were developed for requirements and software engineering tasks.

We focus on the BERT-based models, due to their popularity and suitability for requirements classification [9]. Other LMs, such as GPT-2 and GPT-3 by OpenAI, and XLNet [61], have not been included in our study, as they are not suitable for requirements classification. For example, GPT-2 and GPT-3 are mainly for language generation tasks, such as language translation and text summarization [62], whereas XLN-Net is for NLP involving processing long texts such as paragraphs [61]. Requirements classification typically deals with texts at the sentence level. Below we introduce the four LMs used in our experiments.

- **Sbert:** This generic LM,[6] proposed by Reimers and Gurevych [63], is a fine-tuned version of BERT LM which aims to enrich the semantic embedding representation, i.e., to aid in deriving semantically meaningful sentence embeddings. The LM overcomes the drawbacks of the original BERT models, which use word embeddings to generate sentence embeddings and thus result in weak semantic representations of sentences [63].
- **AllMini:** Introduced by Wang et al. [64] at Microsoft Research, this LM[7] aims to overcome the complexity of some LMs such as BERT models which usually consist of millions of parameters and can be challenging for pre-training and fine-tuning. AllMini reduces (or *distils*) the size of the BERT models, while preserving their performance. The main purpose of AllMini is to support sentence embeddings. In this experiment, we use a version of AllMini (**All-MiniLM v2)**, which was fine-tuned using one billion sentence pairs. This LM is used for encoding sentences and short paragraphs and is particularly efficient for semantic search and sentence clustering tasks.
- **Bert4RE:** This is a RE domain-specific LM [59] which was trained on the $\text{BERT}_{\text{base}}$ model using more than seven million words from different RE-related datasets, including the PROMISE NFR dataset, the PURE dataset [39], and app reviews from Google Playstore and App Store. Although Bert4RE aims to support a wide range of RE tasks, it has only been tested on the task of identifying semantic roles from requirements documents. As this is the only publicly available RE-specific LM, we include it in our study. The BERT4RE LM is provided by the authors in a Zenodo repository.[8]
- **SObert:** This is a SE domain-specific LM [60] that was trained on 152 million sentences from Stack Overflow.[9] SObert[10] shares the same vision as Bert4RE, aiming to capture semantics of the SE terminology. Although SObert has been trained to perform SE specific *named entity recognition (NER)* tasks, it is among the few SE specific LMs that can potentially be adopted for requirements classification.[11]

4.3. Label creation and configuration

Different label creation strategies are used to produce the labels for each requirements class, described as follows.

- **Original labels:** These labels were *derived* from the original class names used in the dataset without using any external knowledge. For example for the task FR/NFR, the original label for the class FR is "functional", while for NFR we use two types of label: (1) the expression "not about functional"[12] and (2) a string including all the NFR class names ("usability, security, availability, ...").
- **Expert curated labels:** These labels were *curated* by the three authors of this paper based on their understanding of the requirements classes. The curation process took three steps: First, we independently provided a set of terms to describe each requirements class, resulting in three sets of terms per class. Second, we discussed our selections and produced a set for each class together. Third, we performed preliminary trials on a part of the requirements, to select the best subset of expert-curated labels. For example, for the task FR/NFR, the label for the class FR is composed of the terms "functional, system, behavior, shall, must", which are typically associated with FRs. These expert-curated labels are expected to complement the aforementioned original labels as they can better discriminate requirements classes.
- **Word-embedding generated labels:** These labels consisted of the terms *extracted* and *selected* from word-embeddings learned from the text of Wikipedia pages belonging to the Computer Science (CS) portal. The idea was that the embeddings learned from the CS portal represent the meaning of words in the CS domain, and are therefore more suitable than a generic LM in providing similar terms for our CS context. We adopted the embedding approach and code provided by Ferrari and Esuli [66]. To agree on a label for each requirements class, we followed these steps: (1) The top-*n* most similar terms were selected according to the word-embeddings; (2) each of the three authors independently annotated the terms with *yes* (indicating the term to be representative for the class), *no* (indicating the term not to be representative for the class), or *maybe* (indicating that the term could be representative for the class); (3) each author revised their "maybe" answer as either *yes* or *no*. (4) the final answer for each class was decided through majority voting. After this procedure, the terms that were tagged with *yes* were included in the label. To assess the degree of agreement between the three authors in step (2), we evaluate the overall percentages of agreement on each term (i.e., all annotators tagged the term with *yes* or *no*), partial agreement (i.e., two out of the three annotators tagged the term with *yes* or *no*), and disagreement (i.e., the three annotators selected three different tags: *yes*, *no*, and *maybe*). In

[5] https://huggingface.co.
[6] huggingface.co/deepset/sentence_bert.
[7] huggingface.co/sentence-transformers/all-MiniLM-L12-v2.
[8] zenodo.org/record/6354280.
[9] https://stackoverflow.com/.
[10] huggingface.co/jeniya/BERTOverflow.
[11] Note that another LM for SE is CodeBERT [65], but it was trained on both natural language and programming texts for specific tasks that involve code retrieval based on natural language queries, which are not suitable for requirements classification.
[12] Preliminary experiments have shown that this term is more effective for ZSL with respect to "non functional".

Table 1
Label configurations for *Task FR/NFR*.

Label Abbr.	Label configuration	FR Label	NFR Label
FR_A	Original label	"functional"	"not about functional"
FR_B	Expert curated	"functional, system, behavior, shall, or must"	"not about functional, system, behavior, shall, or must"
FR_C	Word embedding (selected from top 20 words) + Expert curated	"functional, system, behavior, shall, must, procedural, structural, or characterize"	"not about functional, system, behavior, shall, must, procedural, structural, or characterize"
FR_D	Original label	"functional"	"usability, security, availability, legal, look & feel, scalability, fault tolerance, performance, operational, maintainability, or portability"
FR_E	Expert curated + Original label	"functional, system, behavior, shall, or must"	"usability, security, availability, legal, look & feel, scalability, fault tolerance, performance, operational, maintainability, or portability"
FR_F	Word embedding (selected from top 20 words) + Expert curated + Original label	"functional, system, behavior, shall, must, procedural, structural, or characterize"	"usability, security, availability, legal, look & feel, scalability, fault tolerance, performance, operational, maintainability, or portability"

step (3), after resolving the tags *"maybe"*, we consider the inter-rater agreement (IRR) based on Krippendorff's alpha test [67] as well as Fleiss's Kappa [68]. These statistical tests are used to measure the level of agreement among us. The interpretation of the test results follows the guidelines reported in the Koch Kappa benchmark [69].

The above three label creation strategies are combined into different configurations which are then adopted to each specific classification task to produce task specific labels. Different label configurations and their associated classification tasks are reported in Section 5 for each task.

4.4. Performance measures

For each LM and its label configuration, we measure their performance on each class with respect to a specific classification task using both unweighted and weighted precision (P), recall (R), and F1-score (F1). The weighted P, R and F1 (represented respectively as wP, wR and wF1) are calculated using the distribution of the NFR classes in the dataset. Basically, we performed a weighted sum of the different measures, where the weights are the percentages of requirements in a certain class. These weighted results enable us to compare our results with other studies in requirements classification [8,9]. Instead, the unweighted values, reported by each class, allow us to provide more-fine grained analyses, especially for the multi-class and multi-label cases.

4.5. Experimental setup

Based on the combinations of different LMs, different label selection strategies, and different classification tasks, we have designed and conducted more than 360 experiments. We set up each experiment as a combination of one of the selected LMs, one specific label configuration and one specific task for each dataset. We call each LM-Label combination as a **ZSL classifier**. We use the Transformer API Python package[13] to import and prepare the four selected LMs with their transformer-like tokenizers, and we use the *torch.nn* module[14] in PyTorch to compute the cosine similarity score between two tensors (i.e., the PyTorch tensor objects obtained from the contextual representation by the selected LMs). The *sklearn.metrics* module [70] is used to calculate the classification performance results in terms of the P, R and F1 scores.

[13] huggingface.co/docs/transformers.
[14] pytorch.org/docs/stable/generated/torch.nn.CosineSimilarity.html.

5. Experimental results

In this section, we first report our label configurations and the IRR scores achieved from our label selection for the word-embedding strategy; we then report the experimental results obtained from using the best label configurations.

5.1. Task FR/NFR

For the FR/NFR task we perform a binary classification, which aims to classify a requirement as either FR or NFR.

5.1.1. Label configuration

The label configurations for the FR/NFR Task are reported in Table 1. The labels consist of two groups, one group represents the FR class, and the other the NFR class. Six configurations are used, which combine the different strategies discussed in Section 4.3. In the selection of the word-embedding generated terms, the annotation procedure produced the following statistics: 75% perfect agreement, 25% partial agreement, and 0% disagreement. We also computed the IRR, and we obtained 0.41 as Krippendorff's alpha and a Fleiss' kappa score of 0.40, indicating a moderate agreement.

Concerning the other strategies, it is worth remarking the usage of "functional" vs. "not about functional" (strategy FR_A, Original 1). This type of strategy, in which the original label is negated with the prefix "not about" is also applied for the NFR class label of FR_B and FR_C, and will be applied also later on in this paper to represent the negation of a class in a binary classification.

5.1.2. FR vs. NFR binary classification

Table 2, column Total, reports the overall classification results for all LMs and labelling strategy combinations. In **bold**, we highlight the best combination for each LM.

The overall best combination is Sbert + FR_E, achieving a wF1 score of 0.66, with wP = 0.71 and wR = 0.66. This indicates that the domain agnostic Sbert model, designed to provide a semantic-laden representation for generic sentences, substantially outperforms the other models for this task. Furthermore, the best labelling strategy for Sbert is FR_E, i.e., the one that uses the Expert curated labels + Original labels, which identifies the NFRs using the names of the NFR classes (Usability, Security, Availability, *etc.*).

Looking at Table 2, last six columns, we can see how the performance is divided between FR and NFR classification. We see that the model tends to have higher precision on NFR (P = 0.82), and higher recall on FR (R = 0.82). This is an interesting result, as FRs are less frequent in the dataset (255 FR, 370 NFR), and one would expect to have the opposite result. Indeed, the most frequent class is typically

Table 2
Classification results for FR and NFR classes on *Task FR/NFR*, obtained from the best combination for each LM and label configuration.

ZSL classifier	Total			FR (255)			NFR (370)		
	wP	wR	wF1	P	R	F1	P	R	F1
Sbert + FR_E	**0.71**	**0.66**	**0.66**	**0.55**	**0.82**	**0.66**	**0.82**	0.54	0.65
AllMini + FR_D	0.63	0.59	0.59	0.50	0.71	0.58	0.72	0.50	0.59
Bert4RE + FR_C	0.58	0.56	0.57	0.47	0.65	0.54	0.67	0.50	0.57
SObert + FR_C	0.58	0.59	0.58	0.50	0.41	0.45	0.64	**0.72**	**0.68**

returned more frequently in ML approaches, as it happens, e.g., for NoRBERT (cf. Hey et al. [9], Table III of their paper). This phenomenon occurs also for the other best configurations of LMs. This highlights a characterizing element of ZSL: the performance does not depend on the size of the dataset for each class, because no actual *learning* is performed on the tagged data. A further increase in the accuracy on the FR class could be potentially achieved with a more project-specific labelling strategy for functional requirements (i.e., choosing terms that characterize functional requirements in the specific project).

5.2. Task NFR

In this task, we performed three classification sub-tasks: a binary classification to detect a specific NFR category (e.g., "usability" vs. "other"); a multi-class classification to classify a requirement into one class out of a set of NFR classes; a multi-label classification, in which each requirement is associated with a ranked list of NFR classes, and we want to see if the correct label is in the top-k classes. This last approach can be applied in a semi-automatic classification context, in which the two top-k classes are shown to the requirements analysts, and they are asked to select the correct one. For all the sub-tasks, we evaluate the results: (1) considering only requirements in the largest classes, namely security, usability, performance and operational, which include the majority of the requirements; (2) considering all the classes, except the portability class, which includes one requirement only. For the multi-label classification case, we consider $k = 2$ when only the 4 largest classes are considered, and $k = 3$ when all the classes are considered.

5.2.1. Label configuration

Two labelling configurations are used for the three sub-tasks, one for the binary case (Table 3), and the other for both the multi-class and multi-label classification cases (Table 4).

Binary classification. For the binary case, we have 5 configurations for each NFR class considered, i.e., each binary ZSL classifier. In Table 3 we report only the labels for the usability and security classes, while the other labels are reported in the online supplementary materials.[15] The strategies are analogous to those already discussed for the FR/NFR Task. The only main difference is the usage of the top-50 words from the word embeddings, besides the top-20. We performed some preliminary experiments and saw that the list of similar words for the NFR class names included relevant words also beyond the top-20, and therefore we considered it reasonable to extend the list of terms to be included in the labels. This phenomenon was not observed for the previous task.

Concerning the agreement in the word selection (top-50) we have the following statistics: 52% perfect, 44% partial, and 2% disagreement. We obtained a Krippendorff's alpha rate of 0.45 and 0.53 at macro and micro level, respectively, and a Fleiss's kappa of 0.42 and 0.52 at macro and micro level, respectively (moderate agreement).

Multi-class and multi-label classification. For these tasks, we have a list of labels for each configuration, cf. Table 4. The list is represented with squared brackets, the elements in the list are separated by commas, and each element is a label, expressed between quotes. In the table, each label in the list is associated to one of the 4 largest classes, namely usability, security, performance and operational NFR. The label configurations considering *all* classes are reported in GitHub.[15] We did not use combinations of labelling strategies for these cases, given the extensive number of experiments, and the exploratory nature of the study.

5.2.2. NFR binary classification

Table 5 reports the classification results for the 4 largest NFR classes. The overall results indicate acceptable performance rates, with wF1 > 0.71 for all classes.

The highest wF1 score of 0.84 is achieved for the security class, with AllMini + SE_D, which uses the word embedding selected labels (top-20) for the security class, and the original NFR labels for the "Other" class. Following that is the usability class, with wF1 = 0.80, using Sbert + US_E, which includes the word embedding selected labels (top-50) for the usability class, and the original labels for the "Other" class. This suggests that, for this task and for highly represented classes such as security and usability, generic LMs combined with word-embedding terms as labels appear to be the most effective configuration. It is also worth noting that this binary classification task leads to better results with respect to the FR vs. NFR task (best wF1 = 84 vs. wF1 = 0.66), and the best results are obtained with more complex label configurations. This means that, when using ZSL, it is preferable to select relevant NFR classes and perform binary classification on them, rather than classifying FRs vs. NFRs. Furthermore, while for the task FR/NFR simpler label selection strategies are preferable, more complex labels are appropriate for the NFR binary task.

Table 5, last six columns, considers P, R, and F1 for each class. We see that all the best classifiers tend to achieve higher performance on the "Other" class (best F1 for NFR class 0.70, vs. 0.89 for the "Other" class). This suggests that the ZSL binary classifier encounters some difficulty in associating the requirements to the specific labels, despite the extensive set of terms used. A more accurate selection of terms, or the usage of terms directly coming from the requirements themselves,[16] could overcome this issue.

Finally, Table 6 lists the top performance rates considering all classes, and the entire set of requirements. Comparing these results with Table 5, we see that there is no substantial decrease in terms of performance for the largest classes, e.g., US still achieves wF1 = 0.80, while SE achieves wF1 = 0.85, which is even higher than wF1 = 0.84 in Table 5. However, the results are *all* biased towards the 'Other" class, since, even in the best case, F1 for the NFR class is lower than 0.50. This problem was not so evident in Table 5, at least for the US and SE classes, where F1 is still acceptable also for the NFR class. We can therefore conclude that, in the case of requirements belonging to many different classes, a binary ZSL classification leads to poor classification results, with the selected labelling strategies.

5.2.3. NFR multi-class classification

Table 7 reports the multi-class classification results for the 4 largest NFR classes. For this case, and for the multi-label case, we do not report the weighted measures for the sake of space. However, we remark that the results by class enable a more fine-grained analysis. Weighted F1 is presented in Table 13 to facilitate comparison with other works. We see that, compared to the binary classification, results are substantially lower, although still acceptable for SE (F1 = 0.76), O (F1 = 0.64) and PE (F1 = 0.64). In the majority of the cases, the best results are

[15] https://github.com/waadalhoshan/ZSL4REQ/tree/main/Appendix.

[16] We did not consider this option, as it would have biased the classification. However, it is a viable choice in practical contexts.

Table 3
Label configurations for Usability and Security classes for *Task NFR*, binary classification case.

Label Abbr.	Label configuration	NFR Label	"Other" Label
Usability			
US_A	Original label	"usability"	"not about usability"
US_B	Expert curated	"instructive, easy, helpful, useful, learnable, explainable, affordable, intuitive, or understandable"	"not about instructive, easy, helpful, useful, learnable, explainable, affordable, intuitive, or understandable"
US_C	Word embedding (selected from top 20 words)	"accessibility, aesthetic, contextual, experience, satisfaction, HCI, UX, questionnaire, ease, or ergonomics"	"not about accessibility, aesthetic, contextual, experience, satisfaction, HCI, UX, questionnaire, ease, or ergonomics"
US_D	Word embedding (selected from top 20 words) + Original label	"accessibility, aesthetic, contextual, experience, satisfaction, HCI, UX, questionnaire, ease, or ergonomics"	"security, performance, operational, look feel, legal, fault tolerance, maintainability, scalability, availability, or portability"
US_E	Word embedding (selected from top 50 words) + Original label	"accessibility, aesthetic, contextual, experience, satisfaction, HCI, UX, questionnaire, ease, ergonomics, designer, evaluate, multimodal, practitioner, prototyping, preference, personalization, suitability, focus, clarity, responsiveness, judgement, feel, or helpful"	"security, performance, operational, look feel, legal, fault tolerance, maintainability, scalability, availability, or portability"
Security			
SE_A	Original label	"security"	"not about security"
SE_B	Expert curated	"security, authorization, or protection"	"not about security, authorization, or protection"
SE_C	Word embedding (selected from top 20 words)	"vulnerability, securing, protecting, protection, cybersecurity, assurance, cyber, countermeasure, threat, privacy, authentication, prevention, or confidentiality"	"not about vulnerability, securing, protecting, protection, cybersecurity, assurance, cyber, countermeasure, threat, privacy, authentication, prevention, or confidentiality"
SE_D	Word embedding (selected from top 20 words) + Original label	"vulnerability, securing, protecting, protection, cybersecurity, assurance, cyber, countermeasure, threat, privacy, authentication, prevention, or confidentiality"	"usability, performance, operational, look & feel, legal, fault & tolerance, maintainability, scalability, availability, or portability"
SE_E	Word embedding (selected from top 50 words) + Original label	"vulnerability, security, protection, cybersecurity, assurance, countermeasure, threat, privacy, authentication, prevention, confidentiality, trusted, intrusion, compromise, safety, insecure, defensive, breach, proactive, tampering, penetration, policy, phishing, vulnerable, authorization, dependability, or certification"	"usability, performance, operational, look & feel, legal, fault & tolerance, maintainability, scalability, availability, or portability"

Table 4
Label configurations for the top 4 largest NFR classes (US, SE, O, and PE) for the multi-class and multi-label classification sub-tasks in Task NFR.

Label Abbr.	*Label configuration*	*List of labels*
MultiNFR_A	Original label	["usability", "security", "performance", "operational"]
MultiNFR_B	Expert curated	["instructive, easy, helpful, useful, learnable, explainable, affordable, intuitive, or understandable", "security, authorization, or protection", "periodic execution or efficacy performance", "working, running, connecting, interfacing, or operative environment"]
MultiNFR_C	Word embedding (selected from top 20 words)	["accessibility, aesthetic, contextual, experience, satisfaction, HCI, UX, questionnaire, ease, or ergonomics", "vulnerability, securing. protecting, protection, cybersecurity, assurance, cyber, countermeasure, threat, privacy, authentication, prevention, or confidentiality", "throughput, reliability, scalability, responsiveness, efficiency, workload, benchmark, latency, speed, improvement, or accuracy", "environmental, organizational, coordination, systemic, or logistics"]
MultiNFR_D	Word embedding (selected from top 50 words)	["accessibility, aesthetic, contextual, experience, satisfaction, HCI, UX, questionnaire, ease, ergonomics, designer, evaluate, multimodal, practitioner, prototyping, preference, personalization, suitability, focus, clarity, responsiveness, judgement, feel, or helpful", "vulnerability, security, protection, cybersecurity, assurance, countermeasure, threat, privacy, authentication, prevention, confidentiality, trusted, intrusion, compromise, safety, insecure, defensive, breach, proactive, tampering, penetration, policy, phishing, vulnerable, authorization, dependability, or certification", "throughput, reliability, scalability, responsiveness, efficiency, workload, benchmark, latency, speed, improvement, accuracy, achieve, tuning, bottleneck, better, high, optimize, effectiveness, low, enhances, reducing, increased, quality, faster, or degrades", "environmental, organizational, coordination, systemic, logistics, coordination, or automation"]

obtained again with the domain-generic LMs, and using MultiNFR_A or MultiNFR_B as labels. These are the shortest labels, which do not use the word embedding strategies. This result is the opposite of what was observed for binary classification for NFR. We argue therefore that, in a multi-class classification setting, longer and more informative labels can lead to some possible overlapping between the represented meaning of each class. Instead, in a binary classification setting, more informative labels, i.e., using word embeddings, appear to be more effective.

For the multi-class classification results for all the NFR classes (Table reported in supplementary material), the performance in terms of F1 remains acceptable only for SE class (F1 = 0.69), while for the other classes poor results are obtained.

5.2.4. NFR multi-label classification

Table 8 reports the multi-label classification results for the 4 largest NFR classes, considering the top-2 labels returned by the classifier.

Table 5
Binary classification results of the top 4 NFR classes in *Task NFR*. The best results achieved by a LM on a specific class are in boldface.

NFR class (249)	ZSL classifier	Total			NFR class			"Other" class		
		wP	wR	wF1	P	R	F1	P	R	F1
US (67)	Sbert + US_E	0.81	0.82	0.80	**0.73**	0.49	0.59	0.83	**0.93**	**0.88**
SE (66)	AllMini + SE_D	**0.84**	**0.84**	**0.84**	0.67	**0.73**	**0.70**	**0.90**	0.87	**0.89**
O (62)	Bert4RE + O_C	0.72	0.73	0.72	0.46	0.37	0.41	0.80	0.86	0.83
PE (54)	Sbert + PE_E	0.78	0.78	0.78	0.50	0.46	0.48	0.85	0.87	0.86
PE (54)	AllMini + PE_B	0.80	0.70	0.78	0.47	0.63	0.54	0.89	0.81	0.84

Table 6
Best performance results of binary classification of all 10 NFR classes in *Task NFR*. The results are different with respect to the binary classification using solely four classes because more requirements are involved in the classification, thus leading to lower performance.

NFR class (369)	ZSL classifier	Total			NFR class			"Other" class		
		wP	wR	wF1	P	R	F1	P	R	F1
US (67)	Sbert + US_E	0.80	0.79	0.80	0.44	0.49	0.46	0.88	0.86	0.87
SE (66)	AllMini + SE_D	0.87	0.85	0.85	0.61	0.33	0.43	0.87	0.95	0.91
O (62)	Bert4RE + O_C	0.78	0.77	0.77	0.32	0.37	0.35	0.87	0.85	0.86
PE (54)	Sbert + PE_E	0.82	0.78	0.80	0.33	0.46	0.38	0.90	0.84	0.87
LF (38)	Sbert + LF_D	0.85	0.76	0.80	0.18	0.21	0.20	0.91	0.89	0.90
A (21)	SObert + A_D	0.90	0.70	0.78	0.10	0.43	0.14	0.95	0.72	0.82
SC (21)	AllMini + SC_E	0.92	0.74	0.81	0.13	0.62	0.21	0.97	0.75	0.85
MN (17)	AllMini + MN_E	0.93	0.86	0.89	0.16	0.47	0.24	0.97	0.88	0.92
L (13)	AllMini + L_E	0.95	0.84	0.88	0.11	0.54	0.19	0.98	0.85	0.91
FT (10)	AllMini + FT_E	0.96	0.86	0.91	0.10	0.50	0.17	0.98	0.88	0.93

Table 7
Multi-class classification results for top 4 NFR classes. Results in boldface indicate the best scores achieved by a LM using a specific labelling configuration; the underlined scores indicate the overall best performance achieved by each LM.

ZSL classifier	US			SE			O			PE		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1
Sbert + MultiNFR_A	0.44	**0.52**	0.48	0.57	0.70	0.63	0.47	0.45	0.46	0.43	0.22	0.29
Sbert + MultiNFR_B	**0.73**	0.43	**0.54**	**0.69**	**0.64**	**0.66**	0.53	**0.71**	**0.61**	**0.48**	0.57	0.52
Sbert + MultiNFR_C	0.62	0.31	0.42	0.54	0.62	0.58	**0.59**	0.21	0.31	0.34	**0.74**	0.47
Sbert + MultiNFR_D	0.69	0.40	0.51	0.65	0.53	0.58	0.38	0.23	0.47	0.46	0.53	**0.58**
AllMini + MultiNFR_A	0.67	0.36	0.47	0.66	0.92	0.77	0.33	0.32	0.33	0.45	0.50	0.47
AllMini + MultiNFR_B	**0.77**	0.36	**0.49**	**0.63**	**0.94**	**0.76**	**0.64**	**0.64**	**0.64**	**0.59**	**0.70**	**0.64**
AllMini + MultiNFR_C	0.54	**0.45**	**0.49**	0.76	0.67	0.71	0.36	0.37	0.37	0.41	0.54	0.47
AllMini + MultiNFR_D	0.34	0.43	0.38	0.88	0.33	0.48	0.25	0.42	0.31	0.47	0.28	0.35
Bert4RE + MultiNFR_A	0.32	0.09	0.14	**0.18**	**0.11**	**0.13**	**0.33**	0.02	0.03	**0.19**	**0.67**	**0.30**
Bert4RE + MultiNFR_B	**0.33**	**0.60**	**0.42**	*0.00*	*0.00*	*0.00*	0.28	0.53	**0.37**	0.13	0.02	0.03
Bert4RE + MultiNFR_C	**0.33**	0.06	0.10	0.18	0.08	0.11	0.24	**0.68**	0.36	0.25	0.17	0.20
Bert4RE + MultiNFR_D	*0.00*	*0.00*	*0.00*	*0.00*	*0.00*	*0.00*	0.24	0.34	0.28	0.25	0.04	0.06
SObert + MultiNFR_A	*0.00*	*0.00*	*0.00*	0.60	0.05	0.08	0.22	0.71	0.33	**0.12**	**0.09**	**0.10**
SObert + MultiNFR_B	0.30	**0.81**	**0.44**	*0.00*	*0.00*	*0.00*	0.29	0.16	0.21	0.08	0.06	0.07
SObert + MultiNFR_C	**0.31**	0.36	0.33	**0.24**	**0.21**	**0.22**	**0.31**	0.56	**0.40**	*0.00*	*0.00*	*0.00*
SObert + MultiNFR_D	0.20	0.04	0.07	*0.00*	*0.00*	*0.00*	0.24	**0.90**	0.38	*0.00*	*0.00*	*0.00*

Table 8
Multi-label classification results for the 4 largest NFR classes from the PROMISE dataset. Results in boldface indicate the best scores achieved by a LM using a specific labelling configuration; the underlined scores indicate the overall best performance achieved by each LM.

ZSL classifier	US			SE			O			PE		
	P	R	F1	P	R	F1	P	R	F1	P	R	F1
Sbert + MultiNFR_A	0.60	0.67	0.60	0.74	0.84	0.79	0.72	0.71	0.72	0.68	0.46	0.55
Sbert + MultiNFR_B	0.93	0.60	0.73	**0.86**	**0.77**	**0.82**	**0.71**	**0.92**	**0.80**	**0.73**	**0.91**	**0.81**
Sbert + MultiNFR_C	0.84	0.73	0.78	0.76	0.79	0.78	0.81	0.48	0.60	0.57	0.91	0.70
Sbert + MultiNFR_D	**0.92**	**0.70**	**0.80**	0.83	0.76	0.79	0.62	0.85	0.72	0.81	0.80	0.80
AllMini + MultiNFR_A	0.82	0.60	0.69	0.79	1.00	0.88	0.62	0.58	0.60	0.69	0.74	0.71
AllMini + MultiNFR_B	0.93	0.58	0.72	**0.81**	**0.98**	**0.89**	**0.80**	**0.84**	**0.82**	**0.82**	**0.94**	**0.88**
AllMini + MultiNFR_C	0.63	0.70	0.66	0.94	0.73	0.82	0.63	0.63	0.63	0.64	0.72	0.68
AllMini + MultiNFR_D	**0.63**	**0.88**	**0.74**	0.96	0.74	0.84	0.61	0.58	0.60	0.74	0.63	0.68
Bert4RE + MultiNFR_A	0.84	0.70	0.76	**0.39**	**0.20**	**0.26**	**0.96**	**0.44**	**0.60**	0.32	0.78	0.45
Bert4RE + MultiNFR_B	0.41	0.67	0.51	1.00	0.08	0.14	**0.45**	**0.87**	**0.60**	0.93	0.24	0.38
Bert4RE + MultiNFR_C	**0.98**	**0.76**	**0.86**	0.28	0.11	0.15	0.30	0.73	0.45	0.53	0.31	0.40
Bert4RE + MultiNFR_D	1.00	0.12	0.21	*0.00*	*0.00*	*0.00*	0.31	0.95	0.47	**0.90**	**0.87**	**0.89**
SObert + MultiNFR_A	*0.00*	*0.00*	*0.00*	**1.00**	**0.44**	**0.61**	0.36	0.98	0.53	**0.57**	**0.54**	**0.55**
SObert + MultiNFR_B	0.39	0.85	0.54	1.00	0.05	0.09	**0.75**	**0.92**	**0.83**	0.29	0.13	0.18
SObert + MultiNFR_C	0.71	0.10	0.83	0.46	0.38	0.42	0.43	0.66	0.52	1.00	0.09	0.17
SObert + MultiNFR_E	**0.89**	**1.00**	**0.94**	*0.00*	*0.00*	*0.00*	0.34	0.92	0.49	1.00	0.06	0.11

Table 9
Label configurations for *Task Security*.

Label Abbr.	Label configuration	Security	Non-Security
Sec_A	Original label	"Security"	"not about security"
Sec_B	Expert curated	"Security, authorization, or protection"	"not about security, authorization, or protection"
Sec_C	Word embedding (selected from top 20 words)	"vulnerability, securing. protecting, protection, cybersecurity, assurance, cyber, countermeasure, threat, privacy, authentication, prevention, or confidentiality"	"not about vulnerability, securing. protecting, protection, cybersecurity, assurance, cyber, countermeasure, threat, privacy, authentication, prevention, or confidentiality"
Sec_D	Word embedding (selected from top 50 words)	"vulnerability, security, protection, cybersecurity, assurance, countermeasure, threat, privacy, authentication, prevention, confidentiality, trusted, intrusion, compromise, safety, insecure, defensive, breach, proactive, tampering, penetration, policy, phishing, vulnerable, authorization, dependability, or certification"	"not about vulnerability, security, protection, cybersecurity, assurance, countermeasure, threat, privacy, authentication, prevention, confidentiality, trusted, intrusion, compromise, safety, insecure, defensive, breach, proactive, tampering, penetration, policy, phishing, vulnerable, authorization, dependability, or certification"

In other terms, when the right label is returned by the classifier in the top-2 labels, we consider it a true positive. We see that in this case the performance substantially increases with respect to the multi-class classification case in Table 7, e.g., reaching $F1 = 0.94$ for US requirements, 0.89 for SE, 0.83 for O, and 0.89 for PE. This suggests that the multi-label classification strategy may be the most effective when dealing with NFR classification.

Looking at the results based on the LMs, we do not have a clear pattern, and each LM appears to be suitable for a certain requirement type. Concerning labels, simple configurations as MultiNFR_A and MultiNFR_B appears to be the most effective for all classes, except US, for which the embedding-based labels are more effective. This could be due to a better, and more clear-cut characterization of US requirements with respect to other types.

Performance is similar in case of the multi-label classification results for all the NFR classes (Table reported in supplementary material), considering the top-3 results — i.e., if the right label is returned among the top-3 labels, we consider it a true positive. The performance remain rather high, frequently with F1 above 0.90 for the best configurations. Overall, we can say that ZSL in the multi-label classification context appears to be effective also in case of NFRs belonging to many classes.

5.3. Task security

5.3.1. Security label configuration

The labelling of the security class (Table 9) is similar to the labels groups related to security as an NFR class in the binary classification task (cf. 5.2.1). The agreement results obtained for the word-embedding of the term "security" are the following: 50% perfect, 48% partial, and 2% disagreement. For IRR we obtained 0.46 as Krippendorff's alpha and a Fleiss' kappa score of 0.45, indicating a moderate agreement.

5.3.2. Security binary classification

Table 10, column Total, reports the results for the Security task, considering all the requirements in the three datasets. The best performance is achieved by AllMini + Sec_B, with a wF1 score of 0.66, with wP = 0.68 and wR = 0.65. The generic LM, AllMini, thus achieves best results. On the other hand, the other generic model, Sbert, achieves the worst results (wF1 = 0.31), thus suggesting that generic models are not necessarily better for this specific task. The best set of labels, Sec_B, is the expert's curated one, which includes a limited set of three security-related words. This suggests that a limited number of well-selected terms is sufficient to identify security requirements in this dataset.

Table 10, last six columns, shows the performance for the two classes. We see that better performance in terms of F1 is achieved for Non-Security requirements (F1 = 0.70 vs. 0.58), using AllMini + Sec_B, i.e., the best configuration. Looking in more detail, the best recall (R = 0.92) is obtained by Sbert + Sec_D. Therefore, if one seeks for a better ability to identify security requirements, i.e., high recall on this set, this configuration – though having the worst overall performance – should be preferred. This is an important observation, since for many requirement tasks, including this one, high recall is more important than high precision, as remarked by Berry [71] — if one searches for security requirements, then one wants as less false negatives as possible.

Table 10
Binary classification results for *Task Security*. All the requirements are considered together in one single SeqReq dataset.

ZSL classifier	Total			Security (187)			Non-Security (323)		
	wP	wR	wF1	P	R	F1	P	R	F1
Sbert + Sec_C	0.58	0.41	0.31	0.37	**0.92**	0.53	0.70	0.11	0.19
AllMini + Sec_B	**0.68**	**0.65**	**0.68**	**0.52**	0.67	**0.58**	**0.77**	0.63	**0.70**
Bert4RE + Sec_A	0.52	0.53	0.52	0.34	0.30	0.32	0.62	**0.66**	0.64
SObert + Sec_C	0.56	0.54	0.55	0.39	0.51	0.45	0.66	0.55	0.60

Table 11
Binary classification results for *Task Security*. The results are obtained from each of the three projects in SecReq — CPN, GPS and ePurse.

ZSL classifier	wP	wR	wF1
CPN			
Sbert + Sec_C	0.85	0.30	0.25
AllMini + Sec_B	**0.79**	**0.77**	**0.78**
Bert4RE + Sec_B	0.65	0.80	0.72
SObert + Sec_A	0.65	0.80	0.72
SObert + Sec_B	0.65	0.80	0.72
GPS			
Sbert + Sec_D	0.67	0.40	0.30
AllMini + Sec_B	0.62	0.61	0.61
Bert4RE + Sec_C	0.59	0.53	0.54
SObert + Sec_C	**0.63**	**0.63**	**0.63**
ePurse			
Sbert + Sec_D	0.59	0.64	0.60
AllMini + Sec_A	**0.69**	**0.70**	**0.69**
Bert4RE + Sec_A	0.67	0.44	0.40
SObert + Sec_D	0.66	0.69	0.62

Table 11 reports the results for the Security task, divided by each dataset included in SecReq. Best results are achieved for CPN (wF1 = 0.78), while worst results are for GPS (wF1 = 0.63). This could be due to the specific characteristics of the datasets. In some cases, security and non-security requirements in GPS are expressed with very similar sentences and are likely to be classified similarly though they belong to different classes (e.g., class Security: *The Load File Data Block Hash is used in the computation of the Load File Data Block Signature* vs. Non-Security: *The Load File Data Block Hash is used in the computation of The Load Token*).

6. Research findings

Based on the above detailed analysis of the experimental results, in this section we answer our three RQs one by one. Additionally, we also offer some general observations based on our experiments.

6.1. Best language model (RQ1)

- For *Task FR/NFR*, the best overall performance is obtained with Sbert, which has achieved a wF1 score of 0.66 (with wP = 0.71 and wR = 0.66). This indicates that the generic Sbert model, although designed to provide a semantic representation for generic sentences, substantially outperforms the other LMs (including two domain-specific LMs) for this task.
- For *Task NFR*, the performance of LMs in each sub-task are as follows:
  - For binary classification of NFR, the generic AllMini model outperforms the other three LMs, particularly on the SE class.
  - For multi-class classification of NFR, in the majority of the cases, the best results are obtained by the generic LMs (Sbert and AllMini).
  - For multi-label classification of NFR, there is no clear winner as each LM appears to be suitable for a certain requirement class.
- For *Task Security*, the best overall performer is the generic AllMini model; on the other hand, the generic Sbert model achieves the worst results (wF1 = 0.31). This suggests that generic models are not necessarily better for this specific task and a careful selection of the best LM is key to the success of ZSL.

Based on the above findings, we can state that:

> In the majority of the cases, generic LMs perform better than domain-specific LMs on requirements classification tasks. When applying ZSL in practice one does not need to define domain- or project-specific LMs and can rely on larger ones that are freely available.

Our findings thus contrast the claims that generic LMs do not perform particularly well on domain-specific tasks, as they cannot recognize highly domain-specific vocabulary [10,59,72–74].

Based on our experimental results we can conclude that *generic LMs, being trained on generic data, are more generalizable and adaptable* – the actual sense of being generic; by contrast, *domain-specific LMs, being trained on domain-specific data, are less generalizable and adaptable*, – the actual sense of being specific. Future developments of LMs, we posit, should not differentiate between generic vs. specific, but rather, should focus on continual learning on new tasks and new data [53]. As LMs retain and accumulate knowledge across many tasks, they will become more adaptable to new tasks, domain-specific or otherwise.

6.2. Best label configuration (RQ2)

- For the *Task FR/NFR*, the best label configuration is *FR_E* for Sbert. This configuration is composed of the Expert Curated and the Original labels, which identify the NFRs using the names of the NFR classes (Usability, Security, Availability, etc.). The result shows that expert knowledge of NFR characteristics plays an important role on label configuration for this task.
- For *Task NFR*, we show the performance of label configurations for each sub-task as follows:
  - For binary classification of NFR, the best label configuration is *SE_D* for AllMini, which uses the word embedding with top-20 words for the SE class, and the original NFR labels for the "Other" class.
  - For multi-class classification of NFR, in the majority of the cases, the best label configurations for individual NFR classes are *MultiNFR_A* (Original label) and *MultiNFR_B* (Expert curated label) for Sbert and AllMini.
  - For multi-label classification of NFR, simple label configurations based on either original label (*MultiNFR_A*) and expert curated label (*MultiNFR_B*) appear to be most effective for all classes, except US, for which the embedding-based labels (*MultiNFR_D* and *MultiNFR_E*) are more effective.
- For *Task Security*, the best label configuration is *Sec_B*, curated by expert. Although this label only contains three security-related words, it has shown to be effective in identifying security requirements.

Based on the above findings, we can conclude that:

> Label selection has a relevant impact on the performance of ZSL classification. In general, simple label configurations with the original class names or with a combination of original and expert-curated labels appear to be more effective than more complex word-embedding generated labels.

> The above conclusion implies that, when applying ZSL in practice, for a given requirements classification scheme, domain experts can manually select their own labels for the classes, without using word embeddings. Furthermore, they can also consider project-specific terms that can better distinguish between classes. Preliminary trials to select the best configuration for the problem at hand are also recommended.

> An exception is the binary classification of NFRs, where word-embeddings enable better performance. In these cases, more complex label configurations based on word embeddings should be preferred.

Selecting the most effective label configuration is a difficult task and requires testing many different labels by trial and error. Our study shows how we have handcrafted each label using one of the aforementioned three strategies. However, more work is needed in search for a more systematic approach to label configuration. We argue that expert knowledge of RE, both domain-specific and possibly project-specific, plays an important part in choosing the correct terms for the labels.

6.3. Effectiveness of ZSL for RE (RQ3)

Here we address the effectiveness of ZSL by first comparing our best ZSL results to the state-of-the-art results achieved by Kurtanovìc and Maalej (K&M) [8], Hey et al. (NoRBERT) [9] and Knauss et al. (Knauss) [34], with respect to the same classification tasks (i.e., binary and multi-class classification). Second, we discuss our best ZSL results obtained from multi-label classification with state-of-the-art results and provide our insight into ZSL classification.

- *Binary Classification of FR vs. NFR:* Table 12 shows that both K&M and NoRBERT outperform all our ZSL classifiers. In particular, on FR, K&M produces the best results with a SVM model that applies all the word features in the PROMISE dataset (i.e., without feature selection), achieving F1 = 0.93. On NFR, NoRBERT produces the best results with the fine-tuned $BERT_{large}$ model, with F1 = 0.93. By contrast, the best ZSL classifier (with Sbert LM) has only managed to achieve F1 = 0.66 on FR and F1 = 0.65 on NFR. On average, the performance of the best ZSL classifier is 0.27 lower than that of K&M and NoRBERT. Clearly, these results show that the ZSL approach is (much) less effective than K&M and NoRBERT with respect to this particular task.

Table 12
Binary classification results obtained from *Task FR/NFR* compared to the results obtained by K&M [8] and NoRBERT [9].

Approach (model, train/test)	FR (255)			NFR (370)			wF1
	P	R	F1	P	R	F1	
K&M (word features, 10-fold)	**0.92**	0.93	**0.93**	0.93	0.92	0.92	0.92
K&M (best 100 features, 10-fold)	0.86	0.51	0.63	0.65	0.92	0.76	0.71
K&M (best 500 features, 10-fold)	**0.92**	0.79	0.85	0.82	0.93	0.87	0.86
NoRBERT (base+ep.16[a], 10-fold)	0.89	0.88	0.89	0.92	0.93	0.92	0.91
NoRBERT (large+ep.10+OS, 10-fold)	**0.92**	0.88	0.90	0.92	**0.95**	0.93	0.92
ZSL(Sbert + FR_E, all)	**0.55**	**0.82**	**0.66**	**0.82**	0.54	0.65	**0.65**
ZSL(AllMini + FR_D, all)	0.50	0.71	0.58	0.72	0.50	0.59	0.59
ZSL(Bert4RE + FR_D, all)	0.47	0.65	0.54	0.67	0.50	0.57	0.56
ZSL(SObert + FR_C, all)	0.50	0.41	0.45	0.64	**0.72**	**0.68**	0.59

[a]ep. refers to the number of passes of the training dataset during LM learning process.

- *Binary Classification of NFRs*: Table 13 shows that overall, both K&M and NoRBERT outperform our best ZSL classifier, with wF1 = 0.83 achieved by their best model; on the other hand, the performance of our best classifier (ZSL with Sbert) is 0.10 points worse, with wF1 = 0.73. By examining the results obtained for each class, on US, our ZSL classifier (with Sbert) performs slightly worse than K&M, but outperforms NoRBERT. On SE, although both K&M and NoRBERT outperform our best ZSL classifier (with AllMini), the difference is not large. A similar observation can be made to classes Operational (O) and Performance (PE).
- *Multi-class Classification of NFRs*: For this task, we notice in Table 13 large gaps exist between the results of K&M and NoRBERT and our results on every class. As the purpose of this task is basically the same as the binary classification of NFR task, the inconsistent results achieved by ZSL in these two tasks indicate that when a requirement belongs to many classes, ZSL does not appear to be sufficiently effective.
- *Binary Classification of Security vs. Non-Security Requirements*: Table 14 reveals interesting results. When treating all the security requirements as a whole (i.e., without separating them into different projects), Knauss outperforms the best ZSL classifier by 0.18 points on wF1. However, when the requirements are divided into three projects (i.e., CPN, GPS and ePurse), ZSL outperforms Knauss on all individual projects. In particular, ZSL (AllMini + Sec_B) achieved a high wF1 = 0.78, compared to Knauss's wF1 = 0.40 on CPN. This again seems to suggest that ZSL performs well with binary classification of security requirements when opposite labels are clearly defined.

Based on the above findings, we can conclude that:

> Unsupervised learning with ZSL achieves acceptable performance for binary and multi-class classification tasks. However, it does not outperform supervised classification models, as RE tasks are narrowly defined, and often require well-trained, specifically fine-tuned models on specifically labelled dataset. Nevertheless, without training or fine-tuning, ZSL is more flexible, open to less data-rich tasks, and easily adaptable to the evolution of classification schemes.

> When using ZSL in practice, a company can choose its requirements classification scheme. This will also entail selection of new labels. Given its lower performance in comparison with supervised methods, ZSL is recommended for contexts with large sets of non mission-critical requirements, where misclassification can be tolerated.

In relation to multi-label classification, the following results are obtained: From Table 8, concerning the 4 largest NFR classes, best performance for each class are F1 $\sim 0.83 - 0.94$, which are comparable with the average results of NoRBERT (large + ep.32) for multi-class classification (average F1 = 0.84, cf. Table 13), and are higher than those of K&M.

> To achieve state-of-the-art performance of ZSL for multi-class classification, a multi-label strategy is recommended. In practice, this implies that a semi-automated classification approach should be followed, in which a human operator is asked to select the most suitable class among the top ones returned by the ZSL classifier.

7. Threats to validity

Construct validity. The first threat in our study is the adopted concept of FR and NFRs. This is an artificial distinction [18], as NFRs are often referred to as *qualities* [20], and their classification is often non-binary, i.e., a multi-label classification. However, FR/NFR is a traditional distinction, still common in industrial practice and research. Furthermore, using ZSL for a multi-label binary case would introduce the need for threshold values in the classification (i.e., when both classes have a correlation score above a certain threshold, then classify the requirement as *both* FR and NFR). For this reason, we excluded the binary, multi-label variant of the PROMISE dataset annotated by Dalpiaz et al. [16] from our evaluation. Therefore, the presented approach does not apply to cases in which a requirement can be considered to be both FR and NFR. For security vs non-security requirements, the same observations as for FR/NFR hold. Finally, the adopted metrics for evaluation (precision, recall, weighted F1, accuracy) are those typically used for ML systems, so we do not foresee any major construct validity issue in this aspect.

Internal validity. As our experiments deal with software subjects, which require only limited human intervention, this ensures minimal bias. In the evaluation, we have used established and widely used annotated datasets from the literature. The only internal validity threats are somewhat inherited from the labelling performed by previous work. While the accuracy of the labelling of the PROMISE dataset has been questioned by previous work [9,16], the dataset represents a classical benchmark, which can be used to compare our results with previous proposals. Concerning internal threats due to implementation issues, we have adopted widely used LMs. These models have been tested in other environments, thus increasing confidence in their reliability. Concerning the implementation of the ZSL approach, we have used the Transformers package in Python to retrieve the LMs from HuggingFace hub and to apply encoding for the labels and requirements representations. This package is also widely used, and we have made our code available for inspection in a Google Colab Notebook, so that the results can be replicated. Another possible threat is related to expert-curated labels. To mitigate bias in label selection, we followed a procedure of independent selection, followed by majority voting, and we reported the obtained agreements, which was moderated in all the cases.

External validity. Our results apply to requirements classification cases that are similar to the task considered in the paper. Different results may be observed, e.g., for the classification of requirements vs. non-requirements, and the extraction of relevant content from app reviews. A main threat to external validity is due to the PROMISE dataset, as the requirements in the dataset were largely written and labelled by students, and this may not be representative of industrial requirements [8,9]. We agree that the quality of the dataset can affect the performance of the LMs used in our evaluation. However, the PROMISE dataset has been widely used in the RE community, as a *de facto* benchmark for requirements classification, and using this dataset for research evaluation will allow RE researchers to compare their results with ours. We also make our code and data publicly available so that further replication or reproduction of our approach can be carried out. We also recognize that the lack of labelled requirements datasets has been an open challenge to using ML approaches for RE tasks [6].

Table 13
Binary and multi-class classification results obtained from *Task NFR* compared to the results obtained by K&M and NoRBERT. Only top 4 NFR classes are considered.

Approach (parameters)		US (67)			SE (66)			O (62)			PE (54)			wF1
		P	R	F1	P	R	F1	P	R	F1	P	R	F1	
NFR binary classification														
10-fold cross val. on Top-4 NFR (249)	K&M(w/o features selection)	0.81	**0.85**	0.82	0.91	**0.90**	0.88	0.72	**0.75**	0.73	**0.93**	**0.90**	**0.90**	**0.83**
	K&M (best 50 features)	0.70	0.57	0.61	0.81	0.77	0.74	0.78	0.50	0.57	0.87	0.57	0.67	0.65
	K&M (best 500 features)	0.80	0.71	0.74	0.74	0.81	0.74	0.72	0.73	0.71	0.87	0.81	0.82	0.75
	NoRBERT(base + ep.10)	0.81	0.69	0.74	**0.93**	0.82	0.87	0.80	0.53	0.64	0.88	0.80	0.83	0.77
	NoRBERT(base + ep.10 - OS[a])	0.78	0.70	0.74	0.90	0.86	0.88	0.88	0.71	**0.79**	0.88	0.80	0.83	0.81
	NoRBERT(large+OS+ES[b])	**0.89**	0.70	**0.78**	0.89	0.89	**0.89**	**0.90**	0.71	**0.79**	0.88	0.81	0.85	**0.83**
Test Top-4 NFR (249) w/o training	ZSL(Sbert + (NFR)_D)	0.77	0.78	0.78	0.72	0.71	0.72	0.68	0.72	0.70	0.80	0.67	0.70	0.72
	ZSL(Sbert + (NFR)_E)	**0.81**	**0.82**	**0.80**	0.76	0.78	0.75	0.68	0.63	0.65	0.78	**0.78**	**0.78**	**0.74**
	ZSL(AllMini + (NFR)_B)	0.54	0.35	0.35	0.73	0.73	0.73	0.71	0.65	0.67	0.80	0.70	**0.78**	0.62
	ZSL(AllMini + (NFR)_D)	0.75	0.75	0.75	**0.84**	**0.84**	**0.84**	0.65	0.55	0.58	**0.81**	0.69	0.71	0.72
	ZSL(BERT4RE + (NFR)_C)	0.52	0.43	0.46	0.62	0.65	0.63	**0.72**	**0.73**	**0.72**	0.68	0.41	0.43	0.56
	ZSL(BERT4RE + (NFR)_D)	0.54	0.42	0.45	0.56	0.70	0.61	0.63	0.41	0.42	0.60	0.46	0.51	0.50
	ZSL(SObert + (NFR)_B)	0.55	0.51	0.53	0.81	0.74	0.63	0.65	**0.73**	0.67	0.61	**0.78**	0.68	0.62
	ZSL(SObert + (NFR)_C)	0.51	0.32	0.30	0.66	0.53	0.56	0.71	0.74	0.71	0.61	0.55	0.58	0.53
NFR Multi-class classification														
10-fold cross val. on Top-4 NFR (249)	K&M(word features)	0.65	0.82	0.70	0.81	0.77	0.75	0.81	**0.86**	**0.82**	0.86	**0.81**	0.80	0.76
	K&M(best 50 features)	0.49	0.68	0.55	0.60	0.50	0.39	0.42	0.47	0.33	0.85	0.53	0.63	0.47
	K&M(best 500 features)	0.70	0.66	0.64	0.64	0.53	0.56	0.47	0.62	0.51	0.81	0.74	0.76	0.61
	NoRBERT(base+ep.32)	0.78	**0.84**	0.81	0.89	0.85	0.87	0.79	0.73	0.76	0.88	0.78	0.82	0.82
	NoRBERT(large+ep.32)	**0.86**	0.82	**0.84**	**0.91**	**0.91**	**0.91**	**0.83**	0.71	0.77	**0.90**	**0.81**	**0.85**	0.84
Test Top-4 NFR (249) w/o training	**ZSL(Sbert + MultiNFR_B)**	**0.73**	**0.43**	0.54	0.69	**0.64**	**0.66**	**0.53**	0.71	**0.61**	**0.48**	**0.57**	**0.52**	**0.58**
	ZSL(Sbert + MultiNFR_D)	0.69	0.40	0.51	0.65	0.53	0.58	0.38	0.23	0.47	0.46	0.53	0.58	0.53
	ZSL(AllMini + MultiNFR_A)	0.67	0.36	0.47	0.66	0.92	0.77	0.33	0.32	0.33	0.45	0.50	0.47	0.51
	ZSL(AllMini + MultiNFR_B)	**0.77**	0.36	0.49	0.63	**0.94**	**0.76**	**0.64**	0.64	**0.64**	**0.59**	**0.70**	**0.64**	**0.63**
	ZSL(AllMini + MultiNFR_C)	0.54	0.45	0.49	0.76	0.67	0.71	0.36	0.37	0.37	0.41	0.54	0.47	0.51
	ZSL(BERT4RE + MultiNFR_A)	0.32	0.09	0.14	0.18	0.11	0.13	0.33	0.02	0.03	0.19	0.67	0.30	0.14
	ZSL(BERT4RE + MultiNFR_B)	0.33	0.60	0.42	*0.00*	*0.00*	*0.00*	0.28	0.53	0.37	0.13	0.02	0.03	0.21
	ZSL(BERT4RE + MultiNFR_D)	*0.00*	*0.00*	*0.00*	*0.00*	*0.00*	*0.00*	0.24	0.34	0.38	0.25	0.04	0.06	0.11
	ZSL(SObert + MultiNFR_A)	*0.00*	*0.00*	*0.00*	0.60	0.05	0.08	0.22	**0.71**	0.33	0.12	0.09	0.10	0.13
	ZSL(SObert + MultiNFR_B)	0.30	**0.81**	0.44	*0.00*	*0.00*	*0.00*	0.29	0.16	0.21	0.08	0.06	0.07	0.19
	ZSL(SObert + MultiNFR_C)	0.31	0.36	0.33	0.24	0.21	0.22	0.31	0.56	0.40	*0.00*	*0.00*	*0.00*	0.25

[a]OS refers to Oversampling technique for randomly selecting data from the minority class by adding them to the training dataset Sampling.
[b]ES refers Early Stopping, a feature that enables the model training to be automatically stopped when a selected metric (e.g., F1) has stopped improving.

Table 14
Comparison between the classification results obtained by Knauss et al. [34] and *Task Security* divided by the subset of SeqReq dataset: CPN, GPS, and ePurse. Bold values indicate the best performance results. Underlined values refer to the best performance rates with the ZSL classifier.

Approach (parameters)		wP	wR	wF1
SeqReq (510)				
10-fold cross val. on SeqReq (510)	Knauss et al. (Bayesian classifier)	**0.79**	**0.91**	**0.84**
Test SeqReq (510) w/o training	ZSL(Sbert + Labels_Sec_C)	0.58	0.41	0.31
	ZSL(AllMini + Labels_Sec_B)	**0.68**	**0.65**	**0.66**
	ZSL(BERT4RE + Labels_Sec_A)	0.52	0.53	0.52
	ZSL(SObert + Labels_Sec_C)	0.56	0.54	0.55
CPN (210)				
Train on ePurse (124)	Knauss et al. (Bayesian classifier)	0.23	0.54	0.33
Train on GPS (176)	Knauss et al. (Bayesian classifier)	**0.29**	0.65	**0.40**
Train on ePurse + GPS (300)	Knauss et al. (Bayesian classifier)	0.26	**0.85**	**0.40**
Test CPN (210) w/o training	**ZSL(AllMini + Labels_Sec_B)**	**0.79**	**0.77**	**0.78**
GPS (176)				
Train on ePurse (124)	Knauss et al. (Bayesian classifier)	0.43	**0.85**	**0.57**
Train on CPN (210)	Knauss et al. (Bayesian classifier)	0.29	0.19	0.23
Train on ePurse + CPN (334)	Knauss et al. (Bayesian classifier)	**0.51**	0.56	0.53
Test GPS (176) w/o training	**ZSL(SObert + Labels_Sec_C)**	**0.63**	**0.63**	**0.63**
ePurse (124)				
Train on CPN (210)	Knauss et al. (Bayesian classifier)	**0.99**	0.33	0.47
Train on GPS (176)	Knauss et al. (Bayesian classifier)	0.72	**0.48**	**0.58**
Train on ePurse + CPN (386)	Knauss et al. (Bayesian classifier)	0.84	0.31	0.46
Test ePurse (124) w/o training	**ZSL(AllMini + Labels_Sec_A)**	**0.64**	**0.70**	**0.69**

Conclusion validity. To reduce the threat to our conclusion, we used statistical significance tests to compare the variance of the means within the ZSL classifiers to assess if the systems have the same effect or not. We used one-way Analysis of Variance (ANOVA) with repeated measures, and verified the results with another non-parametric significance test, the Friedman Test. From all the variance testing results, all the ZSL classifiers in all tasks are *statistically significant* for $\alpha = 0.05$, confirming that the ZSL performance results are not due to chance. All

the statistical analysis tests are reported in the online supplementary materials.[17]

8. Conclusion

This paper reports on an extensive study of using the contextual word embedding-based zero-shot learning approach for requirements classification. The study tested this approach using **4** LMs (2 generic and 2 domain-specific), **3** groups of requirements classification tasks (Task FR/NFR, Task NFR, Task Security, and their subtasks), **19** label configurations, and **2** datasets with a total of **1020** requirements. More than **360** experiments were conducted, each based on a combination of a specific LM, a specific task, a specific label configuration, and a specific dataset. The study found:

- Generic LMs perform better than domain-specific LMs under the ZSL approach.
- Simple label selection strategies, i.e., using original labels and expert curated labels, outperform complex strategies such as word-embedding generated labels.

The study also found that in comparison with three previously reported supervised learning approaches for requirements classification, the performance results achieved by the best ZSL classifiers (i.e., the best combinations of the LMs and label configurations) are still lower. However, the ZSL approach is fully unsupervised that does not require any labelled dataset or training. This approach therefore has the great potential to address the problem of labelled data shortages in RE and SE.

Another advantage of the ZSL approach is that it is inherently flexible. Unlike supervised approaches that require a set of *fixed classes* preassigned to the dataset, the ZSL approach can classify requirements into *any unseen new classes* directed by the given labels. Consequently, the ZSL approach is suitable for requirements classification tasks facing changing classification schemes. As classification schemes change, all is required is for the ZSL approach to adopt a new set of labels, which can be defined easily, as our study shows.

Future work will consider the following directions: (1) assess ZSL for the classification of app reviews, using existing datasets made available by previous studies (cf., Dabrowski for a complete list [13]); (2) explore other RE tasks to frame them as classification problems suitable for ZSL; (3) replicate current experiments with the entailment-based ZSL approach, to explore whether better performance can be achieved; (4) consider the few-shot learning approach i.e., by only using a handful of labelled examples to train the classifier, and assess to what extent the shortcomings of ZSL can be addressed by including a limited set of labelled examples; (5) evaluate the effects of different label sizes (i.e., the number of words) on the performance of ZSL.

Replication

We shared our experimentation settings including Colab notebook and the results we obtained from all the ZSL classifiers at https://github.com/waadalhoshan/ZSL4REQ.

Declaration of competing interest

No author associated with this paper has disclosed any potential or pertinent conflicts which may be perceived to have impending conflict with this work. For full disclosure statements refer to https://doi.org/10.1016/j.infsof.2023.107202.

[17] https://github.com/waadalhoshan/ZSL4REQ/blob/main/StatAnalysis_ZSL4RE_results.ipynb.

Data availability

Data will be made available on request.

Acknowledgements

We wish to thank the reviewers for their valuable comments and suggestions, for their support and interest in this paper. We thank the Guest Editors of this special issue, Vincenzo Gervasi and Andreas Vogelsang, for their great support and interest in our paper. Liping Zhao and Waad Alhoshan extend their appreciation to the Deanship of Scientific Research at IMSIU for funding and supporting this work through Research Partnership Program **no. RP-21-07-03**.

References

[1] L. Zhao, W. Alhoshan, A. Ferrari, K.J. Letsholo, M.A. Ajagbe, E.-V. Chioasca, R.T. Batista-Navarro, Natural language processing for requirements engineering: A systematic mapping study, CSUR 54 (3) (2021) 1–41.
[2] M. Kassab, C. Neill, P. Laplante, State of practice in requirements engineering: contemporary data, Innov. Syst. Softw. Eng. 10 (4) (2014) 235–241.
[3] J.A. Sidey-Gibbons, C.J. Sidey-Gibbons, Machine learning in medicine: a practical introduction, BMC Med. Res. Methodol. 19 (1) (2019) 1–18.
[4] I.H. Sarker, Machine learning: Algorithms, real-world applications and research directions, SN Comput. Sci. 2 (3) (2021) 1–21.
[5] S. Minaee, N. Kalchbrenner, E. Cambria, N. Nikzad, M. Chenaghlu, J. Gao, Deep learning–based text classification: a comprehensive review, CSUR 54 (3) (2021) 1–40.
[6] M. Binkhonain, L. Zhao, A review of machine learning algorithms for identification and classification of non-functional requirements, Expert Syst. Appl.: X 1 (2019) 100001.
[7] J. Cleland-Huang, R. Settimi, X. Zou, P. Solc, Automated classification of non-functional requirements, REJ 12 (2) (2007) 103–120.
[8] Z. Kurtanović, W. Maalej, Automatically classifying functional and non-functional requirements using supervised machine learning, in: RE'17, Ieee, 2017, pp. 490–495.
[9] T. Hey, J. Keim, A. Koziolek, W.F. Tichy, NoRBERT: Transfer learning for requirements classification, in: RE'20, IEEE, 2020, pp. 169–179.
[10] A. Sainani, P.R. Anish, V. Joshi, S. Ghaisas, Extracting and classifying requirements from software engineering contracts, in: RE'20, IEEE, 2020, pp. 147–157.
[11] S. Abualhaija, C. Arora, M. Sabetzadeh, L.C. Briand, M. Traynor, Automated demarcation of requirements in textual specifications: a machine learning-based approach, Empir. Softw. Eng. 25 (6) (2020) 5454–5497.
[12] A. Ferrari, F. Dell'Orletta, A. Esuli, V. Gervasi, S. Gnesi, Natural language requirements processing: A 4D vision, IEEE Softw. 34 (6) (2017) 28–35.
[13] J. Dkabrowski, E. Letier, A. Perini, A. Susi, Analysing app reviews for software engineering: a systematic literature review, Empir. Softw. Eng. 27 (2) (2022) 1–63.
[14] W. Maalej, Z. Kurtanović, H. Nabil, C. Stanik, On the automatic classification of app reviews, REJ 21 (3) (2016) 311–331.
[15] W. Wang, V.W. Zheng, H. Yu, C. Miao, A survey of zero-shot learning: Settings, methods, and applications, ACM TIST 10 (2) (2019) 1–37.
[16] F. Dalpiaz, D. Dell'Anna, F.B. Aydemir, S. Çevikol, Requirements classification with interpretable machine learning and dependency parsing, in: RE'19, IEEE, 2019, pp. 142–152.
[17] M. Glinz, On non-functional requirements, in: RE'07, IEEE, 2007, pp. 21–26.
[18] J. Eckhardt, A. Vogelsang, D.M. Fernández, Are "non-functional" requirements really non-functional? an investigation of non-functional requirements in practice, in: ICSE'16, 2016, pp. 832–842.
[19] J. Cleland-Huang, S. Mazrouee, H. Liguo, D. Port, NFR, 2007, http://dx.doi.org/10.5281/zenodo.268542.
[20] I. 29148:2018(E), ISO/IEC/IEEE International Standard - Systems and software engineering – Life cycle processes – Requirements engineering, 2018, pp. 1–104, http://dx.doi.org/10.1109/IEEESTD.2018.8559686.
[21] S.J. Pan, Q. Yang, A survey on transfer learning, IEEE Trans. Knowl. Data Eng. 22 (10) (2009) 1345–1359.
[22] F. Zhuang, Z. Qi, K. Duan, D. Xi, Y. Zhu, H. Zhu, H. Xiong, Q. He, A comprehensive survey on transfer learning, Proc. IEEE 109 (1) (2020) 43–76.
[23] J. Devlin, M. Chang, K. Lee, K. Toutanova, BERT: Pre-training of deep bidirectional transformers for language understanding, in: J. Burstein, C. Doran, T. Solorio (Eds.), NAACL-HLT'19, ACL, 2019, pp. 4171–4186.
[24] H. Larochelle, D. Erhan, Y. Bengio, Zero-data learning of new tasks, in: AAAI'08, 2008, pp. 646–651.
[25] C.H. Lampert, H. Nickisch, S. Harmeling, Learning to detect unseen object classes by between-class attribute transfer, in: CVPR'09, IEEE, 2009, pp. 951–958.

[26] W. Alhoshan, L. Zhao, A. Ferrari, K.J. Letsholo, A zero-shot learning approach to classifying requirements: A preliminary study, in: REFSQ'22, Springer, 2022, pp. 52–59.

[27] E. Knauss, S.H. Houmb, S. Islam, J. Jürjens, K. Schneider, SecReq, 2021, http://dx.doi.org/10.5281/zenodo.4530183.

[28] F.-L. Li, J. Horkoff, J. Mylopoulos, R.S. Guizzardi, G. Guizzardi, A. Borgida, L. Liu, Non-functional requirements as qualities, with a spice of ontology, in: RE'14, IEEE, 2014, pp. 293–302.

[29] M. Broy, Rethinking nonfunctional software requirements, Computer 48 (05) (2015) 96–99.

[30] A. Casamayor, D. Godoy, M. Campo, Identification of non-functional requirements in textual specifications: A semi-supervised learning approach, IST 52 (4) (2010) 436–445.

[31] R. Navarro-Almanza, R. Juarez-Ramirez, G. Licea, Towards supporting software engineering using deep learning: A case of software requirements classification, in: CONISOFT'17), IEEE, 2017, pp. 116–120.

[32] A. Dekhtyar, V. Fong, RE data challenge: Requirements identification with word2vec and tensorflow, in: RE'17, IEEE, 2017, pp. 484–489.

[33] O. AlDhafer, I. Ahmad, S. Mahmood, An end-to-end deep learning system for requirements classification using recurrent neural networks, IST 147 (2022) 106877.

[34] E. Knauss, S. Houmb, K. Schneider, S. Islam, J. Jürjens, Supporting requirements engineers in recognising security issues, in: REFSQ'11, Springer, 2011, pp. 4–18.

[35] M. Riaz, J. King, J. Slankas, L. Williams, Hidden in plain sight: Automatically identifying security requirements from natural language artifacts, in: RE'14, IEEE, 2014, pp. 183–192.

[36] N. Munaiah, A. Meneely, P.K. Murukannaiah, A domain-independent model for identifying security requirements, in: RE'17, IEEE, 2017, pp. 506–511.

[37] S. Christey, J. Kenderdine, J. Mazella, B. Miles, Common Weakness Enumeration, Mitre Corporation, 2013.

[38] V. Varenov, A. Gabdrahmanov, Security requirements classification into groups using NLP transformers, in: REW'21, IEEE, 2021, pp. 444–450.

[39] A. Ferrari, G.O. Spagnolo, S. Gnesi, Pure: A dataset of public requirements documents, in: RE'17, IEEE, 2017, pp. 502–505.

[40] F. Dalpiaz, A. Ferrari, X. Franch, C. Palomares, Natural language processing for requirements engineering: The best is yet to come, IEEE Softw. 35 (5) (2018) 115–119.

[41] B. Romera-Paredes, P. Torr, An embarrassingly simple approach to zero-shot learning, in: ICML'15, 2015, pp. 2152–2161.

[42] Y. Ma, E. Cambria, S. Gao, Label embedding for zero-shot fine-grained named entity typing, in: Proceedings of COLING 2016, the 26th International Conference on Computational Linguistics: Technical Papers, 2016, pp. 171–180.

[43] O. Levy, M. Seo, E. Choi, L. Zettlemoyer, Zero-shot relation extraction via reading comprehension, 2017, arXiv preprint arXiv:1706.04115.

[44] J. Nam, E.L. Mencía, J. Fürnkranz, All-in text: Learning document, label, and word representations jointly, in: AAAI'16, 2016, pp. 1948–1954.

[45] P.K. Pushp, M.M. Srivastava, Train once, test anywhere: Zero-shot learning for text classification, 2017, arXiv preprint arXiv:1712.05972.

[46] T. Mikolov, I. Sutskever, K. Chen, G.S. Corrado, J. Dean, Distributed representations of words and phrases and their compositionality, Adv. Neural Inf. Process. Syst. 26 (2013).

[47] T. Mikolov, K. Chen, G. Corrado, J. Dean, Efficient estimation of word representations in vector space, 2013, arXiv preprint arXiv:1301.3781.

[48] J. Pennington, R. Socher, C.D. Manning, Glove: Global vectors for word representation, in: EMNLP'14, 2014, pp. 1532–1543.

[49] A. Radford, K. Narasimhan, T. Salimans, I. Sutskever, Improving Language Understanding by Generative Pre-Training, Technical Report, OpenAI, 2018.

[50] K. Ethayarajh, How contextual are contextualized word representations? comparing the geometry of BERT, ELMo, and GPT-2 embeddings, 2019, arXiv preprint arXiv:1909.00512.

[51] C. Raffel, N. Shazeer, A. Roberts, K. Lee, S. Narang, M. Matena, Y. Zhou, W. Li, P.J. Liu, et al., Exploring the limits of transfer learning with a unified text-to-text transformer, J. Mach. Learn. Res. 21 (140) (2020) 1–67.

[52] A. Radford, J. Wu, R. Child, D. Luan, D. Amodei, I. Sutskever, et al., Language models are unsupervised multitask learners, OpenAI Blog 1 (8) (2019) 9.

[53] S. Ruder, M.E. Peters, S. Swayamdipta, T. Wolf, Transfer learning in natural language processing, in: NAACL'19, 2019, pp. 15–18.

[54] S. Ruder, Neural Transfer Learning for Natural Language Processing (Ph.D. thesis), National University of Ireland, Galway, 2019.

[55] M.E. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, L. Zettlemoyer, Deep contextualized word representations, in: ACL'18, ACL, 2018, pp. 2227–2237.

[56] J.H. Wenpeng Yin, D. Roth, Benchmarking zero-shot text classification: Datasets, evaluation and entailment approach, in: EMNLP'19, 2019, pp. 3914–3923.

[57] P.V. Sappadla, J. Nam, E.L. Mencía, J. Fürnkranz, Using semantic similarity for multi-label zero-shot classification of text documents, in: ESANN, 2016, pp. 423–428.

[58] P. Bojanowski, E. Grave, A. Joulin, T. Mikolov, Enriching word vectors with subword information, Trans. Assoc. Comput. Linguist. 5 (2017) 135–146.

[59] M. Ajagbe, L. Zhao, Retraining a BERT model for transfer learning in requirements engineering: A preliminary study, in: RE'22, IEEE, 2022, pp. 309–315.

[60] J. Tabassum, M. Maddela, W. Xu, A. Ritter, Code and named entity recognition in StackOverflow, in: ACL'20, ACL, Online, 2020, pp. 4913–4926.

[61] Z. Yang, Z. Dai, Y. Yang, J. Carbonell, R.R. Salakhutdinov, Q.V. Le, Xlnet: Generalized autoregressive pretraining for language understanding, in: NeurIPS, Vol. 32, 2019.

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

[63] N. Reimers, I. Gurevych, Sentence-BERT: Sentence embeddings using siamese BERT-networks, 2019, arXiv preprint arXiv:1908.10084.

[64] W. Wang, F. Wei, L. Dong, H. Bao, N. Yang, M. Zhou, MiniLM: Deep self-attention distillation for task-agnostic compression of pre-trained transformers, in: NeurIPS'20, Vol. 33, 2020, pp. 5776–5788.

[65] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, M. Zhou, CodeBERT: A pre-trained model for programming and natural languages, in: EMNLP'20, ACL, 2020, pp. 1536–1547.

[66] A. Ferrari, A. Esuli, An NLP approach for cross-domain ambiguity detection in requirements engineering, Autom. Softw. Eng. 26 (3) (2019) 559–598.

[67] K. Krippendorff, Content Analysis: An Introduction to its Methodology, Sage publications, 2018.

[68] J.L. Fleiss, J. Cohen, The equivalence of weighted kappa and the intraclass correlation coefficient as measures of reliability, Educ. Psychol. Meas. 33 (3) (1973) 613–619.

[69] J.R. Landis, G.G. Koch, The measurement of observer agreement for categorical data, Biometrics (1977) 159–174.

[70] F. Pedregosa, G. Varoquaux, A. Gramfort, V. Michel, B. Thirion, O. Grisel, M. Blondel, P. Prettenhofer, R. Weiss, V. Dubourg, J. Vanderplas, A. Passos, D. Cournapeau, M. Brucher, M. Perrot, E. Duchesnay, Scikit-learn: Machine learning in Python, J. Mach. Learn. Res. 12 (2011) 2825–2830.

[71] D.M. Berry, Empirical evaluation of tools for hairy requirements engineering tasks, Empir. Softw. Eng. 26 (6) (2021) 1–77.

[72] I. Beltagy, K. Lo, A. Cohan, SciBERT: A pretrained language model for scientific text, 2019, arXiv preprint arXiv:1903.10676.

[73] I. Chalkidis, M. Fergadiotis, P. Malakasiotis, N. Aletras, I. Androutsopoulos, LEGAL-BERT: The muppets straight out of law school, 2020, arXiv preprint arXiv:2010.02559.

[74] J. Lee, W. Yoon, S. Kim, D. Kim, S. Kim, C.H. So, J. Kang, BioBERT: a pre-trained biomedical language representation model for biomedical text mining, Bioinformatics 36 (4) (2020) 1234–1240.