PRCBERT: Prompt Learning for Requirement Classification using BERT-based Pretrained Language Models

Xianchang Luo
University of Science and Technology of China
Hefei, China
bigluo@mail.ustc.edu.cn

Yinxing Xue*
University of Science and Technology of China
Suzhou, China
yxxue@ustc.edu.cn

Zhenchang Xing
CSIRO's Data61 & Australian National University
Canberra, Australia
zhenchang.xing@anu.edu.au

Jiamou Sun
CSIRO's Data61
Canberra, Australia
u5871153@anu.edu.au

ABSTRACT

Software requirement classification is a longstanding and important problem in requirement engineering. Previous studies have applied various machine learning techniques for this problem, including Support Vector Machine (SVM) and decision trees. With the recent popularity of NLP technique, the state-of-the-art approach NoRBERT utilizes the pre-trained language model BERT and achieves a satisfactory performance. However, the dataset PROMISE used by the existing approaches for this problem consists of only hundreds of requirements that are outdated according to today's technology and market trends. Besides, the NLP technique applied in these approaches might be obsolete. In this paper, we propose an approach of prompt learning for requirement classification using BERT-based pretrained language models (PRCBERT), which applies flexible prompt templates to achieve accurate requirements classification. Experiments conducted on two existing small-size requirement datasets (PROMISE and NFR-Review) and our collected large-scale requirement dataset NFR-SO prove that PRCBERT exhibits moderately better classification performance than NoRBERT and MLM-BERT (BERT with the standard prompt template). On the de-labeled NFR-Review and NFR-SO datasets, Trans_PRCBERT (the version of PRCBERT which is fine-tuned on PROMISE) is able to have a satisfactory zero-shot performance with 53.27% and 72.96% F1-score when enabling a self-learning strategy.

KEYWORDS

requirement classification, prompt learning, zero-shot learning, requirement auto-labeling, self-learning strategy

*Corresponding Author.

ASE '22, October 10–14, 2022, Rochester, MI, USA
© 2022 Association for Computing Machinery.
ACM ISBN 978-1-4503-9475-8/22/10…$15.00
https://doi.org/10.1145/3551349.3560417

ACM Reference Format:
Xianchang Luo, Yinxing Xue, Zhenchang Xing, and Jiamou Sun. 2022. PRCBERT: Prompt Learning for Requirement Classification using BERT-based Pretrained Language Models. In *37th IEEE/ACM International Conference on Automated Software Engineering (ASE '22), October 10–14, 2022, Rochester, MI, USA.* ACM, New York, NY, USA, 13 pages. https://doi.org/10.1145/3551349.3560417

1 INTRODUCTION

Software requirements play a vital role in software development and product utilities, which usually include user experience, functional requirements, and quality issues. Generally, a software requirement could be divided into two classes: functional requirements (FR) and non-functional requirements (NFR). The former describes the services and functional behaviors of a software system, while the latter involves user experience, such as quality, usability, security, and issues such as privacy or software permissions [16].

In requirement engineering (RE) domain, plenty of studies [1, 4, 10, 23] have been conducted on software requirement classification — the task of identifying the category of a requirement sentence. NFR classification models based on decision tree and support vector machine [1, 4, 10] are the most common, but often require complex feature engineering. Fine-tuning a pre-trained language model [23] has gradually become the mainstream method of requirement classification. Insofar as these studies have suggested, the existing machine learning (ML)- or natural language processing (NLP)-based approaches could achieve a satisfactory performance on this task with accuracy above 90%. Nevertheless, it needs to be pointed out that all the above studies are all conducted on a small-size requirement dataset, named PROMISE, consisting of 625 software requirements only. Besides, the BERT language model, which NoRBERT [23] applies on PROMISE for fine-tuning, is no longer the most sophisticated techniques in today's NLP domain. *To summarize, the dataset for software requirement classification is outdated and the NLP technique applied for this task could be obsolete.*

To address the aforementioned issues, it is desired to have a large-size dataset consisting of real-world software requirements that could be representative of today's new market demands and technical trends. As social networking sites or app reviews has revolutionized the science of data analysis such as stock trading volatility prediction [28], sentiment analysis [37], these sites or apps could be a ideal source for collecting today's software requirements.

For example, as of March 2021, StackOverflow[1] has more than 14 million registered users, with cumulative questions and answers exceeding 21 million and 31 million respectively. StackOverflow is the most popular social-technical information seeking platform for developers [9], on which developers discusses a wide range of issues related to code, including non-functional requirements. The variety of issues is evident in the post tags, for example, performance used more than 90,000 times, availability used more than 1700 times. As such, it provides software development organizations with an unprecedented opportunity to monitor the opinions of large numbers of users experiencing their systems. Meanwhile, it is also urgent to realize automatic and accurate requirement classification for ultra-large-scale software requirements engineering [17, 38]. Towards these goals, it is inevitable to face the challenges from these aspects: 1) on the available dataset PROMISE, how to improve the requirement classification model via the aid of the merging NLP technique; 2). on the large-size dataset from StackOverflow, how to leverage the learned knowledge from the existing requirement datasets (such as PROMISE) for few-shot or zero-shot learning.

In this paper, we propose the approach of **P**rompt learning for **R**equirement **C**lassification using **BERT** (PRCBERT), which applies flexible prompt templates to achieve accurate classification of software requirements, and then adopts it to auto-label unseen requirements' categories of the large-size requirement dataset crawled from StackOverflow. Overall, we follow the usual method in design science [22]. We first analyze the difficulty of NFR classification and the limitations of current methods, and then design and propose a new BERT-based classification model.

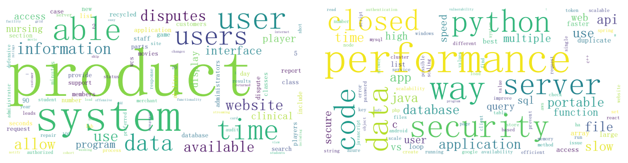
Technically, different from the standard prompt template whose target word is masked by a special token [M] [41, 48, 49], before inputting requirement sequences to transformer-based [51] language model, PRCBERT duplicates each requirement sequence into K samples (in this paper, K means the number of label classes or the size of requirement categories) and then concatenates with K different prompt templates (we call the requirement with a flexible prompt template as assertion in this paper); after acquiring the final hidden state from a transformer-based [51] language model (such as BERT [15] and RoBERTa [35]), PRCBERT adopts the mean pooler strategy to calculate the assertion representation and feed it into the following 2-class sigmoid layer to predict whether these declarative sentences are reasonable or contradictory, respectively (also can be regarded as a two-class natural language inference problem). In addition, we propose an algorithm using the self-learning strategy to enhance the transfer-ability and generality of PRCBERT for auto-labeling unseen requirements.

In experiments, on the two existing small datasets of requirements (i.e., PROMISE and NFR-Review), PRCBERT have exhibited the best classification performance for the task of requirement category prediction in comparison with NoRBERT [23] and BERT-MLM. On the large-size dataset of NFR-SO (we crawl from StackOverflow), under both the scenario of zero- and few-shot learning, PRCBERT is shown to have better transfer-ability and generality than the compared models NoRBERT and BERT-MLM, too. In addition, The proposed algorithm using a self-learning strategy has

[1]StackOverflow: an information sharing and discussion platform for programmers or those who love programming, https://stackoverflow.com/

PROMISE snippets:
A: The product shall be available for use 24 hours per day 365 days per year
PE: The system shall refresh the display every 60 seconds.
FT: The product shall operate in offline mode whenever internet connection is unavailable.
SE: Every user of the system shall be authenticated and authorized.
NFR-SO snippets:
A: Why does NameNode HA use JournalNode instead of Zookeeper istself?
PE: Application Insight doesn't show full End-to-end transaction when latency is too long
FT: How to add fault tolerance support to an existing MPI based system such that the system continues even after a machine goes down?
SE: Unauthorized access when I try to open the generated executable of a Windev project

Figure 1: PROMISE and NFR-SO (non-functional requirements from StackOverflow) requirement snippets (A: availability, PE: performance, FT: fault tolerance, SE: security)



(a) Word Cloud of PROMISE (b) Word Cloud of NFR-SO

Figure 2: Word Clouds of PROMISE and NFR-SO (top-100 most frequent words)

sharply boosted the classification performance of PRCBERT with zero-shot learning on the de-labeled NFR-Review and NFR-SO.

To sum up, our work mainly makes the following contributions:

- To the best knowledge of ours, we make the first attempt on applying prompt learning for software requirement classification. Even in software engineering domain, our study should be, if not the first, one of the earliest studies on exploring prompt learning for software engineering tasks.
- We propose flexible prompting templates by converting one multi-class classification problem into K binary classification problems, which make PRCBERT perform moderately better than NoRBERT (BERT followed by a softmax layer for fitting K-class classification) and BERT-MLM (BERT with the standard prompt template).
- We construct a large-scale requirement dataset NFR-SO, which is 28 times larger than PROMISE and 14 times large than NFR-Review. By directly transferring PRCBERT on NFR-Review and NFR-SO for zero-shot learning, it achieves the F1 score of 37.79% and 66.20%. After combining a self-learning strategy, PRCBERT could achieve the F1 score of 53.27% and 72.96%, respectively.

2 BACKGROUND AND MOTIVATION

In this section, we first introduce the new trend and problem in the software requirement classification task. Then, we introduce several existing approaches towards this task, including RNN-, CNN-, Transformer- or Prompt-based classification.

2.1 A Motivating Example

The PROMISE [47] dataset is a software requirement dataset maintained by the School of Information and Engineering, University of Ottawa, Canada in 2005, which aims to collect various software engineering knowledge for the model to learn to understand and master the "category prediction of software requirement". PROMISE is crowdsourced by 15 teams according to ISO/IEC-25010 standard [24] and only consists of 625 software requirements.

Surprisingly, despite the disadvantage of being a small size (total 625) and from a few crowdsourcing sources (15 teams), PROMISE has been one of most frequently-used requirements dataset [2, 39] in the past two decades. Even in 2020, it is still used [23]. Besides, PROMISE has two other limitations. First, the requirements in PROMISE might not be representative of today's software requirements, considering the rapid development of software techniques. Apart from being outdated, second, the expressions of requirements in PROMISE are very rigid and the sentence structural are almost identical. As shown in Figure 1, the four requirement statements in PROMISE have simple and similar syntactic structures — being subject–verb–predicate (SVP) or subject–verb–object (SVO).

In contrast, today's software requirements from Internet websites such as StackOverflow (see Table 1) contain very varied sentence structures (e.g., interrogative and negative sentences in NFR-SO) and more syntactic characteristics (e.g., acronyms and technical terms). The previous study [21] has already confirmed that data drift still exists between historical and newly published vulnerability descriptions despite being on the same platform. This data drift phenomenon also exists in non-functional requirements. Figure 2 shows the word clouds (the top-100 most frequent words) of PROMISE and NFR-SO, respectively. Though both of them mainly concentrate on the non-functional requirements, such as *available* and *functionality* in PROMISE, and *scalable*, *security*, *performance*, and *availability* in NFR-SO, NFR-SO pays more attentions to the non-functional requirements considering the frequencies and number of these terms. In addition, NFR-SO mentions and lists non-functional requirement issues about these qualitative terms: using *slow*, *slower*, *high*, *different*, *best*, *vs*, *large*, and *faster*, etc. Hence, in RE, absorbing the advanced zero-shot or few-shot learning [11, 42] with prompt is desired to help construct or auto-label the large-scale dataset on the basis of the available small-size labeled dataset.

In this paper, we aim to *construct a promising architecture with the cutting-edge DL (or NLP) techniques to perform requirement classification on large-scale labeled dataset of software requirements from real-world technical forum such as StackOverflow and the existing available small size datasets (such as PROMISE)* .

2.2 Existing Text Classification Techniques

In general, many existing classification techniques in NLP have already been or could potentially be applied to solve the software requirement classification. We categorize them into three types of classification and briefly introduce each type as follows.

RNN- and CNN-based Classification. RNN-based classification models use LSTM or GRU cell to extract sequence semantics in time steps [32], while CNN-based models process and integrate local information through convolution operations [25, 43]. Although the following various enhancements (e.g., combining RNN and CNN in TextCNN [30]; integrating lexical, syntactic, and semantic characteristics into contextualized word representation in ELMo [3]; or applying attention mechanism to autofocus sequence semantics in HAN [56, 59]) have improved the above vanilla language models. However, limited by the causal LSTM architecture and the local convolution operations, RNN- and CNN-based models concentrate on local information extraction (attention mechanism and feature engineering heuristics can alleviate it to a certain extent, but cannot make a breakthrough). Hence, *RNN- and CNN-based classification models have been popular for quite a while, until BERT proposed in 2018, surpassed by the transformer-based classification.* Numerous experiments on transformer-based language models [13, 15, 18, 46, 57] show that CNN- or RNN-based models have no strong competitiveness in classification performance nowadays — where 66M parameters DistilBERT [46] outperformed 180M parameters ELMo [3] on GLUE task STS-B [52], in addition, DistilBERT takes less inference time.

Table 1: Samples of NFR-SO for 7 NFR Classes[*]

NFR Class (Frequency)	Samples
A (1301)	• How to ensure that the data in the consumption Kafka is not lost?
PE (6794)	• Why is checking for a variables existence taking more time than copying an array which should be a O(1) vs O(n) operation?
MN (295)	• How can I maintain an iOS Framework initially installed via CocoaPods but modified manually later on?
PO (1393)	• How can I put the animate transition of my javascript into my css?
SC (2206)	• How to store and index several billion logs every day
SE (5211)	• Fortify: How to automate getting issues (vulnerability) list under a project using Fortify API/CLI, to break my pipeline if there are vulnerabilities
FT (234)	• How to add fault tolerance support to an existing MPI based system such that the system continues even after a machine goes down?

[*] A: availability; PE: performance; MN: maintainability; PO: portability; SC: scalability; SE: security; FT: fault-tolerance.

Transformer-based Classification. Transformer-based language models achieve nearly perfect performance jump from the previous best reported (i.e., RNN- and CNN-Based language models). Transformer-based language models typically consist of two steps: (1) unsupervised pre-training from the unlabeled corpus and (2) transferring to downstream tasks (generally fine-tune for sentence-level and subword-level tasks with just an additional neural network following the pre-trained model). One of the most influential transformer-based language models **BERT** (Bidirectional Encoder Eepresentations for Transformers) [15] utilizes the two auxiliary tasks, NSP (next sentence prediction) and MLM (masked language model), to implement the efficient understanding of the input sequence. **RoBERTa** (Robustly optimized BERT approach) [35] removes NSP objective and pre-trains the model longer with bigger batches and larger corpus, the performance of the model is further improved. **NoRBERT** (Non-functional and functional Requirements classification using BERT) [23] utilizes a pre-trained BERT model to perform classification task on the PROMISE [47] dataset (see Figure 5 NoRBERT Input) and outperforms the other compared methods. Although transformer-based models can easily acquire excellent results on handling NLP tasks [33, 55], *simply adding an additional neural network after a transformer-based model may result in abruptness and a low correlation between the input sequence and the target task* — for example, given the input sequence "The system shall refresh the display every 60 seconds", BERT will use a special token [CLS] which means the semantics of input sequence. However, it fails to connect the vector representation of [CLS] to the task of requirement category prediction.

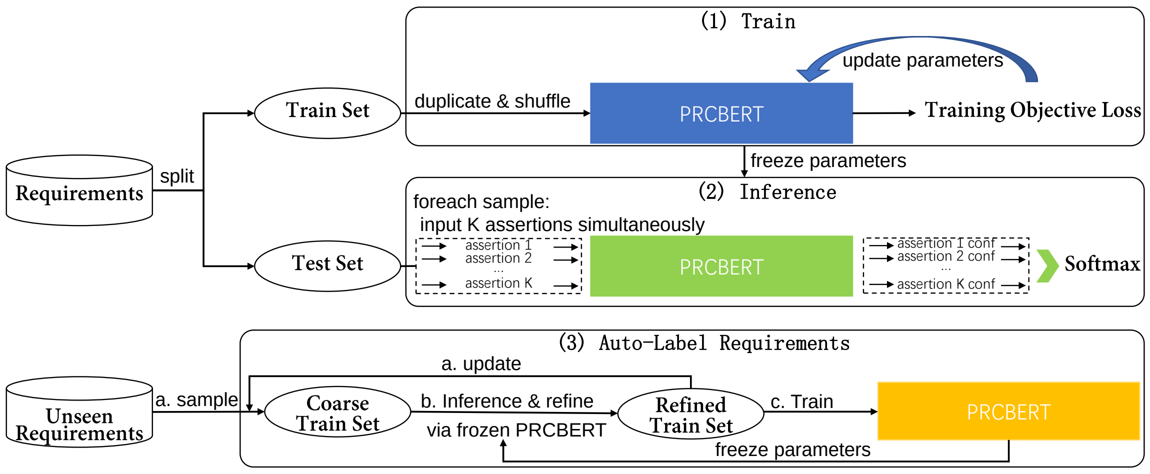


Figure 3: Overview of our approach, where the top part corresponds to the train (fine-tune) step and the inference step respectively; and the bottom part is our proposed auto-labeling method (zero-shot + self-learning) for unseen requirements.

Prompt-based Classification. The key idea of prompt learning is to add a prompt text into the input sequence to enhance the semantic fluency between input and target task [48, 49]. The excellent performance of the prompt-based language models [11, 42] show that the application of prompt is able to bridge the semantics gap between general pre-trained language models and specific classification task. Standard prompt engineering usually masks a target label (we refer to this standard prompt classification as **BERT-MLM**) appended after the input sequence to improve the semantic fluency and correlation between the input sequence and the target task [48, 49]. For example, in Figure 5, given the input sequence "The system shall refresh ...", BERT-MLM will append the text "This is [M] Requirement", where [M] is a special token used to replace the target label. Here, BERT-MLM uses the $h_{[M]}$ (final vector representation of special token [M]) but not $h_{[CLS]}$ (vector representation of input sequence learned by BERT) to predict the target word in the form of cloze test will not create a pretrain-finetune discrepancy (special token [M] occurs in both pretrain and finetune steps) [55]. For the requirement classification problem in RE, traditional ML techniques (e.g., support vector machine, k-nearest neighbors) and transformer-based models [15, 23] have been applied on the dataset PROMISE. However, *we have not witnessed any prompt-based learning approach towards the requirement classification problem and the auto-labeling of a large-scale dataset.*

2.3 Research Challenges

Abundant studies have exhibited that prompt learning can help further improve the transformer-based language models' performance, if a proper prompt template is used. For example, by customizing the prompt template, CLIP [41] has significantly boosted the image-text bimodal classification performance by 5%. GPT-3 [11] and T5 [42] all adopt a prompt to unify NLU (natural language understanding) and NLG (natural language generation) tasks, grasping better semantic understanding ability from joint-learning of multiple tasks, where GPT-3 achieves 81.5% F1 score when zero-shooting on CoQA (conversational question answering) [45] challenge (the previous SOTA is 90.7%). Besides, it is proved that the application of a self-learning strategy is able to help for the classification task, such as relation type prediction [7, 8].

To follow the success of applying prompt-learning in the NLP domain, three research challenges are encountered when adopting the prompt-based classification for solving the task of software requirement classification and auto-labeling.

C1. How to design a more effective prompt template, boosting the classification *accuracy*, than the standard prompt template used in BERT-MLM (see the example in Figure 5)?
C2. How to attain a prompt strategy preserving the *transfer-ability* and *generality* of a pretrained language model, suiting the characteristics of unseen requirements?
C3. How to combine self-learning with prompt-learning for the *automation* of labeling the large-scale requirement samples, even when most of them are unseen before?

For these ends, in this paper, we present the approach PRCBERT with flexible prompt templates and combine self-learning strategy to address these challenges. The workflow and technical details of PRCBERT are depicted and elaborated in §3 and §4, respectively.

3 APPROACH OVERVIEW AND KEY SOLUTIONS

In this section, we will introduce the overview of PRCBERT (including the training and inference step) and key solutions.

3.1 Overview

Following modern models' workflow to address a classification task, PRCBERT consists of three phases: fine-tune training, inference, and auto-labeling (see Figure 3). The input of the whole approach is a dataset of requirement samples (e.g., a well-labeled dataset such as PROMISE). On this given dataset, PRCBERT will apply the proposed prompt templates to train and update its learnable parameters, then apply this model for requirement inference (prediction). Last, PRCBERT combines self-learning with this fine-tuned model to iteratively auto-label the unseen requirement samples (e.g., those from StackOverflow). Each phase is introduced below:

(1) Train Step. Before inputting the train samples into PRCBERT, we duplicate each sample appended with K different prompt templates and then shuffle the duplicated train set. After getting the classification confidence from the binary classifier in PRCBERT, we calculate the training objective loss (see §4.4) and update the gradient of its learnable parameters through back propagation. During training step, we iterate the loss calculation and gradient descent operation until PRCBERT convergences or the train epochs reach the predefined maximum threshold.

(2) Inference Step. The middle part of Figure 3 is the inference step of PRCBERT. Different from the duplicating and shuffling operations of training step, during inference we simultaneously input the duplicated K assertions of a given input sample to PRCBERT and then get the corresponding K classification confidences, the class which owns the maximum softmax value over the K classification confidences corresponds to the predicted label of the requirement sequence. Hence, the output of this step is the predicted label for each requirement sample.

(3) Auto-label Step. The bottom part of Figure 3 illustrates how PRCBERT auto-labels unseen requirements via combining zero-shot learning with self-learning strategy. That is, we first directly apply the model of PRCBERT from step (2), which is fine-tuned on a labeled requirement dataset (such as PROMISE), to zero-shot on another requirement dataset (e.g., NFR-SO) for unseen requirements classification. Then, different from the active learning strategy [20, 26] turning to the help of time-consuming human annotation, we select the samples with higher confidence for iteratively updating PRCBERT to further boost the zero-shot performance of PRCBERT, called self(-supervised) learning [34].

3.2 Key Solutions to Challenges

We highlight the key solutions in PRCBERT to address the aforementioned challenges in §2.3 as follows.

Flexible Prompt Templates for C1 and C2. Instead of using the standard prompt template with a mask token [M], we convert the problem of K-class classification into K times of binary classification, and then instantiate each category to get the corresponding prompt templates (see Figure 5). The simple but effective prompt template is "This is <requirement category name> Requirement.". Note that, though the number of our proposed flexible prompt templates equals the number of classes, PRCBERT can be directly applicable to a classification problem with any number of classes (not necessary to additionally adapt the classification model).

Zero-shot + Self-learning for C3. In addition to our proposed flexible prompt templates, we adopt a self-learning strategy to further boost the performance of auto-labeling the unseen requirements. As shown in the bottom part of Figure 3, we iterate the below steps to auto-label the unseen requirements:

(a) *getting the coarse train set*: we obtain the dataset in two ways, random sampling from the original requirement dataset at the beginning or updating by the following refined train set during self-learning iteration;
(b) *refining the train set*: to get the refined train set, first, we should acquire the prediction confidence of each sample in coarse train set via the frozen parameters PRCBERT (at beginning initialized with the PRCBERT fine-tuned on another requirement dataset, during iteration we will apply the updated PRCBERT whose parameters are frozen), then filter the sample whose corresponding maximum confidence is less than the predefined threshold. Note that, we empirically regard the predicted labels with high confidence as ground truth (i.e., a requirement sentence labeled with the predicted category) for the subsequent training step;
(c) *fine-tuning PRCBERT based on the refined train set*: in this step, we perform supervised training and update PRCBERT based on the refined train set.

Until no sample is filtered or a predefined maximum of epochs is reached, the self-learning process terminates.

4 TECHNICAL DETAILS OF PRCBERT

To better understand our proposed PRCBERT, we will elaborate on the major steps in this section, including the transformer architecture, sigmoid binary classifier, input vector representation, training objective, and the auto-label algorithm.

4.1 Transformer Encoder

Transformer [51] was first proposed by Ashish Vaswani in 2017. This model is completely different from traditional recurrent and convolutional neural networks. Only based on the attention mechanism [5], the transformer can implement semantic interaction between all sequence text subwords (also called tokens) without any attenuation (due to the gradient vanishing and explosion caused by the long relative distance between subwords, previous LSTM-based language models can process 200 context words on average [14, 27]).

Typically, a transformer consists of an encoder and a decoder, since only the text classification task is involved in this paper, we will merely discuss the encoder part here. As shown in Figure 4, a transformer [51] encoder is composed of a stacked L-layer subblock, each subblock possesses multi-head self-attention and FFN (feed-forward network) two sublayers.

Multi-Head Self-Attention. In this sublayer we first should project $H^{l-1} \in \mathbb{R}^{len \times d_{model}}$ into Q, K, V through three independent projection networks, where the len means the predefined maximum sequence length, d_{model} is the model embedding dimension, and $W_Q, W_K, W_V \in \mathbb{R}^{d_{model} \times d_{model}}$ are three projection networks weight matrixes, respectively.

$$\begin{aligned} Q &= H^{l-1} W_Q \\ K &= H^{l-1} W_K \\ V &= H^{l-1} W_V \end{aligned} \tag{1}$$

Then, as shown in Equation 2: carrying out the multi-head self-attention mechanism, where h represents the number of heads (similar to convolutional kernel [40], to some extent, the larger number of heads assigned, the more semantic features can be learned); $d_k = \frac{d_{model}}{h}$; $Q_i, K_i, V_i \in \mathbb{R}^{len \times d_k}$ are the i-th slice of Q, K, V. In each concatenated head attention computation, dot-product $Q_i K_i^T$ means semantic interaction between sequence subwords, $\frac{1}{\sqrt{d_k}}$ is the scaling factor, and the softmax function is to get the normalized semantic weights, this square matrix represents the degree of influence between the semantics of each subword in the sequence.

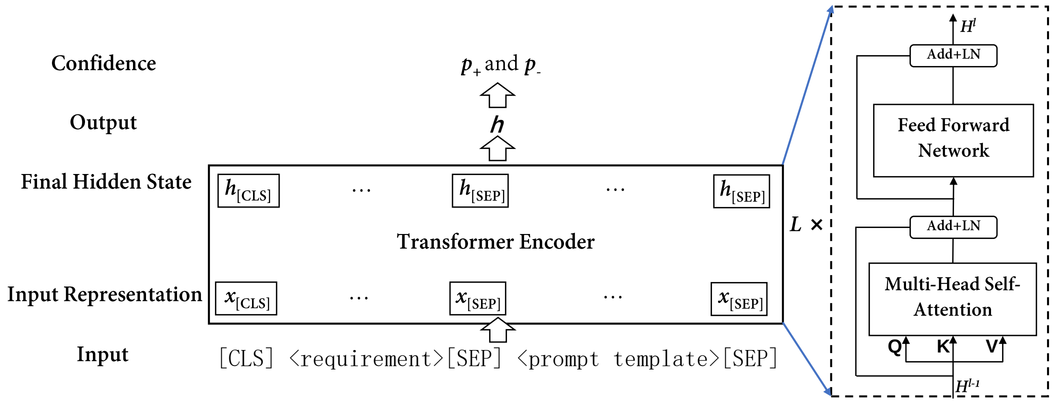
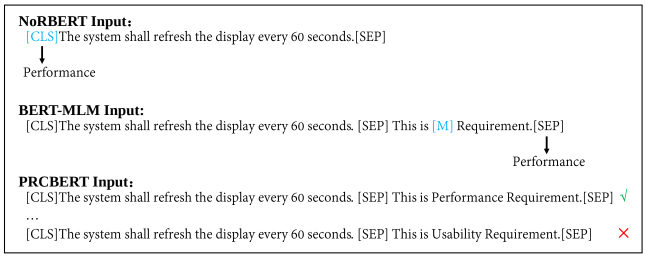


Figure 4: Architecture of PRCBERT


NoRBERT Input:
[CLS]The system shall refresh the display every 60 seconds.[SEP]
↓
Performance

BERT-MLM Input:
[CLS]The system shall refresh the display every 60 seconds. [SEP] This is [M] Requirement.[SEP]
↓
Performance

PRCBERT Input:
[CLS]The system shall refresh the display every 60 seconds. [SEP] This is Performance Requirement.[SEP] √
...
[CLS]The system shall refresh the display every 60 seconds. [SEP] This is Usability Requirement.[SEP] ✕

Figure 5: NoRBERT, BERT-MLM, and PRCBERT classification examples, where blue font means the input of classifier (represented by a down arrow ⇂), ✓ and ✕ mean the binary classification results, respectively.

$$
\begin{aligned}
\text{MultiHead(Q,K,V)} &= \text{Concat}(\text{head}_1, ..., \text{head}_h)W^O \\
\text{where head}_i &= \text{Attention}(Q_i, K_i, V_i) \\
\text{Attention}(Q_i, K_i, V_i) &= \text{softmax}(\frac{Q_i K_i^T}{\sqrt{d_k}})V_i
\end{aligned}
\tag{2}
$$

Feed-Forward Network. The output of the Multi-Head Self-Attention sublayer will then be input to a feed-forward network (FFN). As illustrated in Equation 3, FNN contains two linear layers with only a ReLU activation in between [51], where $W_1 \in \mathbb{R}^{d_{model} \times d_{ff}}$ and $W_2 \in \mathbb{R}^{d_{ff} \times d_{model}}$ (d_{ff} is an integral multiple of d_{model}) first increase the dimension and then reduce the dimension of sequence hidden state to achieve the main semantic information acquisition like the convolution operation [40].

$$\text{FFN}(x) = \max(0, xW_1 + b_1)W_2 + b_2 \tag{3}$$

To sum up, the calculation of the transformer encoder can be represented as Equation 4, where H^0 is the input sequence representation (see §4.3), and L is the number of encoder layers.

$$H^l = \text{subblock}(H^{l-1}),\ l = 1, ..., L \tag{4}$$

4.2 Sigmoid Classifier.

In PRCBERT, we change a K-class problem into K binary classification problems with corresponding prompt templates. Thus, we apply a sigmoid classifier rather than a softmax classifier, Equation 5 shows the confidence calculation that the assertion is predicted as correct, and the contrary confidence of assertion which is predicted as false equals $1 - S(x)$.

$$S(x) = \frac{1}{1 + e^{-x}} \tag{5}$$

The reason of applying sigmoid function is to normalize the linear layer outputs to effectively execute the backward gradient propagation for parameters updating.

4.3 Input Representation

In this paper, we exploit the pre-trained BERT [15] and RoBERTa [35] transformer-based language models to initialize our PRCBERT respectively without learning from scratch. Transformer-based model's original input is a sequence of subwords (using an unsupervised BPE algorithm to divide subwords[2]), BERT applies MLM (masked language model) and NSP (next sentence prediction) auxiliary tasks to capture the input sentence pair semantics, while the input of RoBERTa [35] is only a single sentence (RoBERTa has removed the next sentence prediction auxiliary task).

When using BERT initialization, the language model input (in this paper, also called assertion) is as "[CLS] <requirement text> [SEP] This is <requirement category name> Requirement. [SEP]", where the special token [CLS] appended in the front of the original input is used to represent the entire input sequence semantics, and the appended [SEP] is a separator token of each sentence. Following that used in BERT [15], for each subword w_t in the input sequence, its representation x_t (as shown in Equation 6) is composed of the corresponding token embedding, position embedding, and segment embedding. when initialized with RoBERTa, compared with BERT, we replace the first [SEP] token of its input with "\n" and don't use the segment embedding any longer.

$$x_t = TE_{w_t} + PE_{w_t} + SE_{w_t} \tag{6}$$

Therefore, the PRCBERT input representation H^0=$[x_1, x_2, ..., x_{len}]$, where len is the predefined maximum length of the input sequence (in this paper we set it 512). When the length of the input sequence is less than len, we will use consecutive special tokens [PAD] to pad the sequence length, and if the input sequence is longer than len (in this paper, all of our requirements are less than 512), we will truncate the first len subwords (including the [CLS] and [SEP] tokens).

[2]BPE algorithm repository https://github.com/google/sentencepiece

Algorithm 1: Auto-label algorithm

```
   Input: $PRCBERT$, the PRCBERT fine-tuned on PROMISE
   Input: $dataset$, a new unlabeled requirement dataset
   Input: $threshold$, a float whose range is in (0, 1)
   Output: $output$, classification result of $dataset$
1  $coarseSet \leftarrow$ random_sample($dataset$)
   /* self-learning */
2  while True do
3      $refinedSet \leftarrow \emptyset$
4      foreach max_confidence in PRCBERT.inference(coarseSet) do
5          if max_confidence > threshold then
6              $refinedSet += (requirement, predicted\_label)$
7          end
8      end
9      if $coarseSet == refinedSet.requirements$ then
10         break;
11     end
12     $PRCBERT \leftarrow PRCBERT$.train($refinedSet$)
13     $coarseSet \leftarrow refinedSet.requirements$
14 end
15 $result \leftarrow PRCBERT$.inference($dataset$)
16 return $result$
```

4.4 Training Objective

As the description of the training step in §3, PRCBERT changes the K-class classification problem into K 2-class judgment problems by duplicating the sample sequence with K corresponding flexible prompt templates. The training objective loss of PRCBERT is the cross-entropy of all duplicated assertions (see Equation 7):

$$\mathcal{L} = -\sum_{i=1}^{N}\sum_{j=1}^{K} \log p_{(i,j),y_{(i,j)}}, y_{(i,j)} \in \{+,-\} \tag{7}$$

where N and K are the numbers of requirements in the train set and the number of classes respectively; label + and − represent the corresponding assertion is True and False, respectively; $y_{(i,j)}$ is the true label of the j-th assertion of the i-th requirement; in addition, $p_{(i,j),y_{(i,j)}}$ denotes the probability that the j-th assertion of the i-th requirement is predicted to be $y_{(i,j)}$ through PRCBERT.

4.5 Auto-label Algorithm

In this subsection, we will introduce our auto-label algorithm which is integrated with an unsupervised self-learning strategy in detail.

Algo. 1 is our proposed algorithm for auto-classification of unlabeled requirement datasets. This algorithm has three inputs: the PRCBERT which is already fine-tuned on PROMISE, an unlabeled requirement *dataset*, and a float value called *threshold*. In addition, its output *result* is the predicted labels of each requirement in *dataset*. At the first line, we should initialize the *coarseSet* by randomly sampling from *dataset*; In lines 3-8, we execute the refine operation and filter the sample whose corresponding *max_confidence* outputted by PRCBERT is less than *threshold*; then as shown in line 12, based on the *refinedSet*, the PRCBERT updates its inner parameters via gradient descent; we will iterate the self-learning operation until no requirements are filtered out (see lines 9-11).

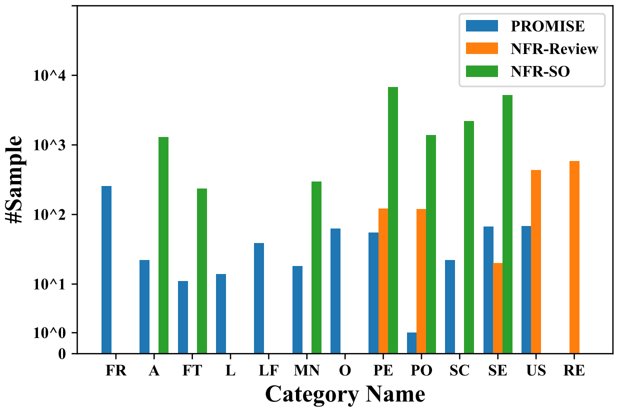


Figure 6: Samples distribution of PROMISE, NFR-Review, and NFR-SO; where F: functional, A: availability, FT: fault-tolerance, LF: look & feel, MN: maintainability, O: operational, PE: performance, PO: portability, SC: scalability, SE: security, US: usability, RE: reliability

Finally, Algo. 1 outputs the predicted labels which are inferred by the well-trained PRCBERT.

5 EVALUATION

In this section, we first list some research questions (RQs) in evaluation, introduce the three different requirement datasets, then depict the experimental setup in detail, and finally answer the RQs.

5.1 Research Questions

We aim to answer these three research questions (RQs):

RQ1. What are the best model architecture and parameter choices of the proposed PRCBERT for requirement classification?

RQ2. How accurate is PRCBERT in classifying requirements in PROMISE, NFR-Review and NFR-SO? When compared with NoRBERT and BERT-MLM, can PRCBERT be better?

RQ3. How do the capabilities of generalization and transfer-ability of PRCBERT? How effective is Algo. 1 in enabling PRCBERT to auto-label unseen requirements?

5.2 Datasets

We introduce the three used requirement datasets as follows.

PROMISE. PROMISE is a requirement dataset crowdsourced by 15 teams in 2005, which widely appears in various requirement classification literature, even in 2020, it is still used in [23]. As shown in Figure 6, the PROMISE requirement dataset can be divided into two parts: FR (functional requirement, 255 samples) and NFR (non-functional requirement, 370 samples). In addition, PROMISE has 11 NFR categories (label classes).

NFR-Review. This requirement dataset is mentioned and released in [53], which consists of 1278 NFR user review sentences from iBooks and WhatsApp two popular Apps (therefore we name it NFR-Review). Figure 6 shows that NFR-Review has a total of 1278 samples of 5 NFR categories: security, portability, performance, reliability, and usability.

Table 2: Statistics of PROMISE, NFR-Review, and NFR-SO

Item / Dataset	#Sample	#Class	Avg. Word	Avg. Char
PROMISE	625	12	22	120
NFR-Review	1278	5	14.5	70
NFR-SO	17434	7	11	70

NFR-SO. NFR-SO (nonfunctional requirements from StackOverflow) is a labeled dataset that we crawled (using BeautifulSoup4[3]) from the real-world technical forum StackOverflow, each sample of NFR-SO is the interrogation content tagged by one of the 7 NFR categories (including availability, performance, maintainability, portability, scalability, security, and fault-tolerance). Normally, their tags are credible, as they are chosen or added manually by the human questioners. Still, there are some samples with two or more tags among the seven, and we will filter them as we are forming a dataset for single-label classification. Then, we ensure the correctness of sample labels as much as possible through auxiliary manual inspection. Finally, we collect NFR-SO that contains 17434 samples (each sample corresponds to the question part, excluding the subsequent answers) in total[4].

Table 2 shows the statistic information of the three datasets, where NFR-Review is 2x larger than PROMISE, and NFR-SO is ~28x larger than PROMISE. In addition, only PROMISE contains the functional requirements (255 samples); Both the average number of words and characters of PROMISE requirements are the longest.

5.3 Experimental Setup

Parameters Setting. The pre-trained transformer-based language models are all downloaded from the model hub [5] maintained by huggingface, including BERT-base, BERT-large, RoBERTa-base, and RoBERTa-large, whose parameters are following that in BERT [15] or RoBERTa [35]. Besides, the maximum input sequence length len equals 512. During training, we set the batch size as 8, learning rate as 5e^{-6}, epochs as 32, and all the train and inference steps are executed on a 24GB NVIDIA GeForce RTX 3090. Moreover, the optimizer we adopt is AdamW [36], and its inner parameters are $\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 1e^{-6}$, and $weight\_decay = 0.0$, respectively.

Evaluation Baselines. To reflect the characteristic of PRCBERT through a fair comparison, we use three baseline models: NoRBERT [23], BERT-MLM, and Trans_PRCBERT. **NoRBERT** directly applies the sequence representation ($h_{[\mathrm{CLS}]}$) outputted from a transformer-based language model to a K-class classification neural network. Different from NoRBERT, **BERT-MLM** combines the original requirement text with a standard prompt template (e.g., This is $[M]$ Requirement.) as the input sequence, after getting the final hidden state of this sequence from a transformer, BERT-MLM feeds the vector representation of the masked token $[M]$ ($h_{[\mathrm{M}]}$) to a K-class classification neural network to restore its target class. **Trans_PRCBERT** is previously fine-tuned on PROMISE and will be subsequently trained on NFR-Review or NFR-SO, rather than directly trained on NFR-Review or NFR-SO like PRCBERT. The corresponding classification examples are illustrated in Figure 5.

[3]BeautifulSoup4 Homepage, https://www.crummy.com/software/BeautifulSoup/

[4]We have released the NFR-SO dataset in anonymous website https://sites.google.com/view/prcbert/home

[5]https://huggingface.co/models

Table 3: The binary classification performance of PROMISE (K = 2, FR or NFR)

Model	Pooler	FR (255)			NFR (370)		
		P	R	$F1$	P	R	$F1$
PRCBERT with BERT-base	first	0.9246	0.9137	0.9191	0.9293	0.9595	0.9441
	mean	0.9141	0.9176	0.9159	0.9128	**0.9622**	0.9368
PRCBERT with BERT-large	first	0.9186	0.9294	0.9240	0.9368	**0.9622**	0.9493
	mean	0.9195	0.9412	0.9302	0.9267	0.9568	0.9415
PRCBERT with RoBERTa-base	first	0.9249	0.9176	0.9213	0.9207	0.9730	0.9461
	mean	**0.9498**	0.8902	0.9190	0.9289	0.9541	0.9413
PRCBERT with RoBERTa-large	first	0.9198	**0.9451**	**0.9323**	0.9440	0.9568	0.9503
	mean	0.9195	0.9412	0.9302	**0.9468**	**0.9622**	**0.9544**
NoRBERT with BERT-large[1]	first	0.92	0.88	0.90	0.92	0.95	0.93

[1] The experimental result is from NoRBERT [23].

Evaluation Metrics. In this paper, we use the F1 score (as shown in Equation 8) to measure the classification performance of a specified class. TP_i means the number of samples that both the predicted classes and their target classes all are i-th class, while FP_i is the number of samples whose predicted classes are i-th class but their target classes are non-i-th classes, and FN_i is the number of samples whose target classes are i-th class but their predicted classes are non-i-th classes.

$$P_i = \frac{TP_i}{TP_i + FP_i},\quad R_i = \frac{TP_i}{TP_i + FN_i},\quad F_i = \frac{2 * P_i * R_i}{P_i + R_i} \tag{8}$$

For the overall performance mensuration of multi-class classification, we adopt the weighted F1 score (also called w-F) as below:

$$w_i = \frac{\#sample_i}{\sum_{j=1}^{K} \#sample_j},\quad w\text{-}F = \sum_{i=1}^{K} w_i * F_i \tag{9}$$

Where w_i means proportion of the samples with i-th class label in all samples, and F_i corresponds to the F1 score of the i-th class samples classification.

To eliminate the randomness in the training process, we perform the paired T-test [29, 54] to analyze the classification performance [6] between PRCBERT and the baselines (i.e., NoRBERT and BERT-MLM).

Note that, all metric calculations are based on the 10-fold cross-validation output with multiple repeated executions. In addition, when fine-tuning on PROMISE, we only use the random seed 904727489, for the classification of NFR-Review and NFR-SO, the inference result is the mean of three w-F (the corresponding random seeds are 42, 930728, and 904727489 respectively).

5.4 Experimental Results

Binary Classification of PROMISE FR/NFR. The classification performance of PROMISE FR/NFR is shown in Table 3, and we obtain 8 sets of experimental results by varying the values of the three parameters (pre-trained language model, the corresponding model size, and the pooler strategy). Where the pooler strategy is the sequence vector representation extraction method that changes

[6]we perform the paired T-test with the aid of the Microsoft Excel spreadsheet program.

Table 4: The classification performance (F1 score) of PROMISE NFR subclasses (K=10)

NFR	**PRCBERT**								**NoRBERT**[1]
	BERT-base		BERT-large		RoBERTa-base		RoBERTa-large		BERT-large
	first	mean	first	mean	first	mean	first	mean	
A (21)	0.9767	0.9268	0.8947	0.8780	0.9302	0.9268	0.9767	**1.0000**	0.78
FT (10)	0.8889	0.7368	0.9412	0.9412	0.8889	**1.0000**	**1.0000**	**1.0000**	0.60
L (13)	0.9630	0.6842	0.9630	**1.0000**	0.8889	0.9600	0.9600	0.9630	0.83
LF (38)	0.9333	0.8974	0.9589	0.9333	0.9189	0.9189	**0.9600**	0.9367	0.80
MN (17)	0.7692	0.8571	0.8649	0.8824	0.7879	0.8125	0.8824	**0.9412**	0.53
0 (62)	0.9280	0.9268	0.8730	0.9106	0.8527	0.8800	0.9440	**0.9449**	0.81
PE (54)	0.9434	0.9074	0.9423	0.9434	0.9174	0.9159	0.9455	**0.9615**	0.90
SC (21)	0.8636	0.8718	0.8780	0.8636	0.8500	0.8718	**0.9091**	**0.9091**	0.76
SE (66)	0.9559	0.9774	0.9697	0.9701	0.9552	0.9697	0.9774	**0.9851**	0.91
US (67)	0.9197	0.9420	0.9412	0.9630	0.9489	0.9130	0.9630	**0.9701**	0.86
w-F	0.9263	0.9127	0.9278	0.9360	0.9083	0.9165	0.9544	**0.9613**	0.82

[1] The experimental result is from NoRBERT [23].

sequence's final hidden state into output (see Figure 4), including first ($h_{first} = h_{[\text{CLS}]} = H_0^L$) and mean ($h_{mean} = \text{avg}(H^L)$).

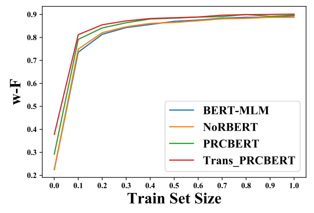
For the FR classification task, the poorest performance is RoBERTa-base with a mean pooling strategy, its F1 score still can reach 91.9% (at this time, the highest F1 score of NoRBERT is only 90%). In the classification of NFR, BERT-base with the mean pooling has the lowest performance, its F1 score is 93.68%. Results show that our PRCBERT model can leave a performance jump than NoRBERT which directly applies the BERT model to deal with downstream tasks, even the PRCBERT initialized by the smaller BERT-base language model outperforms NoRBERT-large in functional and non-functional requirements classification tasks by 1.91% and 0.68%, respectively.

NFR Subclasses Classification of PROMISE. Table 4 shows the classification performance of the 10 PROMISE NFR subclasses (because there is only one portability requirement, which cannot be split into 10 folds, this subclass is not considered).

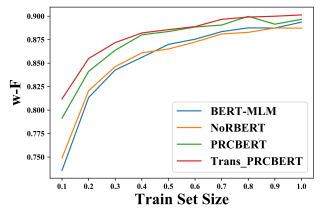
Results show that our PRCBERT with flexible prompt templates can achieve the best overall classification performance on the 10 subclasses of non-functional requirements in PROMISE, where the poorest PRCBERT initialized by RoBERTa-base model with the mean pooling strategy can also reach 90.83% weighted F1 score (8.83% higher than NoRBERT). As the size of the initialized language model increases (base to large), the performance of PRCBERT on PROMISE NFR also shows an upward trend, and the PRCBERT initialized by RoBERTa-large language model with the mean pooling strategy reaches the highest weighted F1 score to 96.16%.

Both the experimental results in Table 3 and Table 4 all show that the PRCBERT initialized by RoBERTa-large language model with the mean pooling strategy can achieve the overall best performance when performing the classification task on the PROMISE dataset. In view of the superior performance of the PRCBERT model under these parameters, in the subsequent experiments, our PRCBERT will apply these parameters (using RoBERTa-large pre-trained language model and mean pooling strategy).

Answer to RQ1: When initialized with RoBERTa-large pre-trained language model, and using the mean pooling strategy to extract sequence vector representation from the final hidden state calculated by the transformer-based model, PRCBERT can achieve the overall best classification performance.



(a) performance on different train set size of NFR-Review (0-100%)



(b) performance on different train set size of NFR-Review (10%-100%)

Figure 7: The classification performance (mean of 3 repeated experiments) of the four compared models on different train-set sizes of NFR-Review

Table 5: The classification performance (w-F) of the four models on different train-set sizes of NFR-Review (K=5)*

Train Set Size	NoRBERT	BERT-MLM	PRCBERT	Trans_PRCBERT
0.0	0.2248	0.2257	0.2915	0.3779
0.1	0.7358	0.7491	0.7915	0.8124
0.2	0.8134	0.8207	0.8413	0.8550
0.3	0.8428	0.8463	0.8638	0.8719
0.4	0.8562	0.8610	0.8802	0.8822
0.5	0.8700	0.8651	0.8837	0.8857
0.6	0.8755	0.8725	0.8885	0.8889
0.7	0.8836	0.8811	0.8906	0.8967
0.8	0.8876	0.8828	**0.8999**	0.8991
0.9	0.8874	**0.8875**	0.8891	0.9000
1.0	**0.8937**	0.8873	0.8967	**0.9014**

* Each value is the mean w-F of 3 repeated experiments (random seeds are 42, 930728, and 904727489 respectively).

Classification on NFR-Review. In order to explore the transferability and generality of our PRCBERT, we further compare the classification performance of NoRBERT, BERT-MLM, PRCBERT, and Trans_PRCBERT (PRCBERT is firstly fine-tuned on PROMISE, then trained on NFR-Review) on NFR-Review requirement dataset.

Figure 7 shows the performance curves of the four models on different train-set sizes of NFR-Review, besides, Table 5 lists the corresponding detailed statistic information of Figure 7(a), where the underline mark means the best performing model in a specified train set size, and the bold font is to highlight the highest w-F of a model among all train set sizes. Table 6 is the paired T-test result between the classification performance of the four models on different train-set sizes of NFR-Review, where the hypothesis

- H_0: data sequence A has no significant improvement than data sequence B;
- H_1: data sequence A indeed has significant improvement than data sequence B.

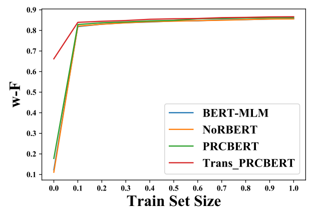
Based on hypothesis H_0, we set the hypothesized mean difference as 0, $\alpha = 0.05$. From Figure 7, Table 5, and Table 6, we can draw some conclusions:

- The classification performance of BERT-MLM has no significant improvement than that of NoRBERT;
- PRCBERT outperforms NoRBERT and BERT-MLM among all train-set sizes (PRCBERT's weighted F1 score is 2.26% higher than NoRBERT on average overall train-set sizes). In addition, PRCBERT can achieve a maximum weighted F1 score of 89.99%.
- Due to Trans_PRCBERT being previously fine-tuned on PROMISE and being able to directly be applied for classification of NFR-Review, the semantic understanding learned from

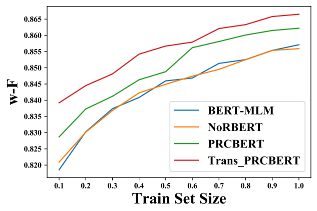
Table 6: The paired T-test between classification performance of 4 models on different train set size of NFR-Review. If the value of t Stat is larger than t Critical One-tail, the corresponding data pair will violate the H_0

Data Pair[1]	t Stat	t Critical One-tail	Obey H_0
(BERT-MLM, NoRBERT)	0.4152	1.8125	Yes
(PRCBERT, NoRBERT)	3.6506	1.8125	No
(Trans_PRCBERT, NoRBERT)	2.7740	1.8125	No
(PRCBERT, BERT-MLM)	4.1934	1.8125	No
(Trans_PRCBERT, BERT-MLM)	2.8634	1.8125	No
(Trans_PRCBERT, PRCBERT)	1.8151	1.8125	No

[1] Each object in a data pair is the weighted $F1$ score sequence corresponding to a column in the Table. 5.



(a) performance on different train set size of NFR-SO (0-100%)



(b) performance on different train set size of NFR-SO (10%-100%)

Figure 8: Classification performance (mean of 3 repeated experiments) of 4 models (NoRBERT, BERT-MLM, PRCBERT, Trans_PRCBERT) on different train set size of NFR-SO

PROMISE is preserved to a certain extent, finally manifesting as the weighted F1 score of zero-shot on NFR-Review is 37.79%. Moreover, the performance curve of Trans_PRCBERT is overall the best among all 4 models and reaches the highest weighted F1 score of 90.14%.

- The result in Figure 7(b) shows that at 10% to 40% of the train-set size, the weighted F1 scores of the four models have the fastest growth rate. When only fine-tuned on the 70% train-set, Train_PRCBERT can achieve 99.48% (0.8967/0.9014) classification performance compared with using the whole train-set, acquiring more rich semantic comprehension fast to some extent.

Classification on NFR-SO. In order to further explore the classification performance of the PRCBERT on an extremely large dataset, we continue to apply these four models (Trans_PRCBERT is now firstly fine-tuned on PROMISE, then trained on NFR-SO) to perform the classification task on the NFR-SO dataset with 17434 samples.

Figure 8 shows the performance curves of the four models on different train-set sizes of NFR-SO. Table 7 shows the corresponding detailed statistics of Figure 8(a). In addition, Table 8 lists the statistics of the paired T-test results between the classification performance of 4 models on different train-set sizes of NFR-SO (the related setting is the same as that applied in paired T-test of NFR-Review, e.g., hypothesis, hypothesized mean difference, α, etc.).

Based on Figure 8, Table 7, and Table 8, we could conclude that:

- Our PRCBERT with flexible prompt templates is able to achieve the maximum weighted F1 score of 86.22%, outperforming NoRBERT and BERT-MLM among all train-set sizes (increased by 1.08% and 1.20% on average, respectively).

Table 7: The classification performance (w-F) of the four models on different train-set sizes of NFR-SO (K=7)*

Train Set Size	NoRBERT	BERT-MLM	PRCBERT	Trans_PRCBERT
0.0	0.1109	0.1225	0.1779	**0.6620**
0.1	0.8209	0.8186	0.8287	**0.8392**
0.2	0.8301	0.8303	0.8373	**0.8445**
0.3	0.8368	0.8374	0.8412	**0.8481**
0.4	0.8423	0.8408	0.8463	**0.8542**
0.5	0.8449	0.8460	0.8488	**0.8567**
0.6	0.8474	0.8468	0.8562	**0.8579**
0.7	0.8495	0.8514	0.8581	**0.8621**
0.8	0.8525	0.8525	0.8601	**0.8633**
0.9	0.8554	0.8554	0.8615	**0.8658**
1.0	0.8559	0.8571	0.8622	**0.8665**

* Each value is the mean w-F of 3 repeated experiments (random seed are 42, 930728, and 904727489 respectively).

Table 8: The paired T-test between classification performance of the four models on different train-set sizes of NFR-SO.

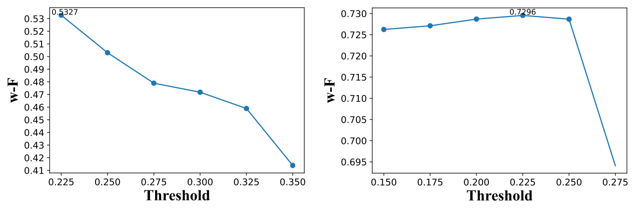
Data Pair[1]	t Stat	t Cirtical One-tail	Obey H_0
(BERT-MLM, NoRBERT)	0.1086	1.8331	Yes
(PRCBERT, NoRBERT)	13.7676	1.8331	No
(Trans_PRCBERT, NoRBERT)	15.9082	1.8331	No
(PRCBERT, BERT-MLM)	10.6173	1.8331	No
(Trans_PRCBERT, BERT-MLM)	11.7404	1.8331	No
(Trans_PRCBERT, PRCBERT)	8.1068	1.8331	No

[1] Each object in a data pair is the weighted F1 score sequence corresponding to a column in the Table 7

- Train_PRCBERT which is transferred from PROMISE to NFR-SO gains more natural language semantic understanding and achieves the best weighted F1 score of 86.65%. There is an obvious performance boost compared with the other three models (increased by 6.12% than NoRBERT, 6.01% than BERT-MLM, and 4.93% than PRCBERT on average, respectively).
- Trans_PRCBERT is able to auto-label a new dataset (NFR-SO) without any adaptation of the classification neural network which is added behind a pre-trained transformer-based language model. Results show that Trans_PRCBERT outperforms the other three models that use a random initialization for downstream classification at the beginning, and reaches a zero-shot performance of 66.20%, showing high generalization and transferability.

Answer to RQ2: PRCBERT with flexible prompt templates outperforms NoRBERT and BERT-MLM on the PROMISE, NFR-Review, and NFR-SO datasets. In PROMISE NFR classification, PRCBERT makes a 14.13% performance boost over NoRBERT. In the classification of NFR-Review and NFR-SO, the classification accuracy of PRCBERT is moderately better than NoRBERT (2.26% and 1.09%) and BERT-MLM (2.19% and 1.20%) on average among 11 different train-set sizes.

Auto-label Unseen Requirements. Given the excellent generalization and transfer-ability of Trans_PRCBERT as shown in classification on NFR-Review and NFR-SO, we apply Algo. 1 integrated with self-learning strategy to further enhance zero-shot classification performance of Trans_PRCBERT (previously pre-trained on PROMISE). To test the capability of this model in auto-labeling the



(a) performance on NFR-Review (b) performance on NFR-SO

Figure 9: The classification performance (mean of 3 repeated experiments on NFR-Review and NFR-SO with different thresholds of label confidence for the self-learning strategy in Trans_PRCBERT

Table 9: Zero-shot performance on NFR-Review and NFR-SO

Zero-shot Strategy	Dataset	*w-F*
Trans_PRCBERT	NFR-Review	0.3779
Trans_PRCBERT with self-learning enabled	NFR-Review	0.5327
Trans_PRCBERT	NFR-SO	0.6620
Trans_PRCBERT with self-learning enabled	NFR-SO	0.7296

unseen requirements, we de-label the NFR-Review and NFR-SO datasets as the input *dataset* of Algo. 1, respectively.

As shown in Table 9, Trans_PRCBERT with a self-learning strategy auto-labels the de-labeled NFR-Review and NFR-SO requirement dataset much more effectively and achieve a higher weighted F1 score of 53.27% (15.48% higher) and 72.96% (6.76% higher), existing an obvious performance improvement than directly zero-shot with PRCBERT on the de-labeled requirements). Figure 9 shows the performance (average F1 score calculated on Equation 9) growth curves with different thresholds on NFR-Review and NFR-SO. According to the Pigeonhole principle, we set the threshold starting from $\frac{1}{4}$ and $\frac{1}{7}$ (we choose its approximation of 0.15) respectively with a 0.025 (a more precise value is likely to cause overfit) step increment. Results illustrate that selecting the sample whose confidence of its predicted label is larger than 0.225 to form the initial *coarseSet* (see Algo. 1) can reach a much more promising performance gain.

Answer to RQ3: Trans_PRCBERT exhibits excellent transferability and generalization capabilities, as its zero-shot performance on NFR-Review and NFR-SO is 37.79% and 66.20%, respectively. Enabling the self-learning strategy (see Algo. 1) of Trans_PRCBERT, it is able to boost the zero-shot performance on NFR-Review by 15.48%, and on NFR-SO by 6.76%.

5.5 Discussion

In this section, we identify the major threats to validity and discuss the practical applications and limitations of our proposed method.
Threats to Validity. Our experiments are based on BERT [15] and RoBERTa [35], but not on other more extensive pre-trained language models. However, existing experiments are available to prove the superiority and universality of our proposed method. Regarding the NFR-SO dataset, we collected and formed it from StackOverflow. We determined the tags of these requirement statements according to the tag category of the question and an auxiliary manual check. The accuracy of labels may be affected by the tags of StackOverflow itself. In the future work, we will consider more rigorous manual labeling and inspection to reduce the noise in the data as much as possible. In addition, the hyperparameter settings and data partitioning in these experiments may also affect the experimental results. However, we set up three random seeds for repeated experiments to minimize the contingency of the results. We also conducted paired T-test to compare the performance of various models. The experimental results prove the relative performance advantages and stability of our method.
Applications and Limitations in Practice. NLP-supported requirements engineering has been widely practiced in academia and industry, such as classifying requirements, detecting language issues, and generating domain specific languages [58]. Requirements classification is beneficial to requirements apportionment and reuse. Our proposed method could improve the performance of requirements classification in practical applications. Furthermore, our work contributes a new large dataset of non-functional requirements for developing and experimenting new requirement analysis methods. In practice, PRCBERT's self-learning strategy can be combined with an active-learning strategy to enhance its transfer-ability in the zero-shot scenarios. However, there may be a weakness (or a trade-off) in our PRCBERT, which is scarifying the inference efficiency to get a better classification performance — with the number of classes increasing, it will take more (lower than linearly because GPU can accelerate parallel computing to a certain extent) inference time to predict the class of the input sequence.

6 RELATED WORK

Our study is mainly relevant to the following two lines of research: requirement classification and BERT for requirement engineering.

6.1 Requirements Classification

The classification of requirements is an important research problem in the field of software engineering. The task classifies requirements into different categories based on application purposes. In recent years, a series of work [1, 4, 10, 50] based on supervised machine learning techniques are dedicated to the classification of non-functional requirements (NFRs). In [10, 50], Tóth, Binkhonain et al. compare the performance of a variety of NFR classification models based on machine learning through experiments. Their results show that the model based on Support Vector Machines (SVMs) achieves the best performance on a small labeled NFR dataset. In [4], Amasaki et al. use the word vector representation of NFRs and SVM to classify the requirement statements into 14 NFR categories. In [1], Abad et al. improve the performance of decision tree classifier through preprocessing and unifying the PROMISE NFR dataset. Although the above machine learning-based models have achieved good performance on small-scale labeled NFR dataset, they rely on manually-labeled training data and feature engineering, so they are difficult to be applied in large-scale requirement engineering.

Due to the limitations of machine learning methods, some work based on deep learning methods [6, 12, 23] extracts semantic information of requirement statements through a deep model to automate the classification of NFRs. In [6], CNN and word embedding are used to divide NFRs into five categories, with the F1 score ranging from 82% to 92%. In [23], Hey et al. propose NoBERT, which is an NFR classification model based on BERT, a pre-trained language model (PLM). Good results are achieved on the PROMISE NFR dataset by fine-tuning BERT on downstream requirements classification tasks. In [12], Chatterjee et al. use the Snorkel tool [44] to automatically label the unlabeled NFRs extracted from a large number of documents and then apply these data to train PLMs, such as BERT, for domain adjustment.

6.2 BERT for Requirements Engineering

In recent years, in addition to BERT-based NFR classification, BERT and other PLMs have also surpassed traditional deep learning models in many other tasks of requirements engineering, showing the best effectiveness. In [38], Mekala et al. classify the user feedback requirements collected based on BERT and word vector, so as to identify the useful requirements and useless requirements. In [19], Fischbach et al. utilize BERT and other models to detect whether causality exists in requirement statements. In [31], Lin et al. propose a novel framework called Trace BERT based on BERT to generate trace links between source code and natural language artifacts. The author compares the accuracy and efficiency of three different BERT architectures. Through transfer learning and PLMs, these three models overcome the problem of insufficient data and achieve better results than classical models such as RNN.

However, these works simply use PLMs through fine-tuning. In contrast, we apply prompt learning to reconstruct the input texts and customize flexible prompt templates to better use PLMs.

7 CONCLUSION AND FUTURE WORK

In this paper, we make the first attempt on applying prompt learning with a BERT-based pretrained language model for software requirement classification and propose flexible prompting templates by converting one multi-class classification problem into K binary classification problems. The evaluation shows the proposed PRCBERT performs significantly better than NoRBERT (BERT followed by a softmax layer for fitting K-class classification) and BERT-MLM (BERT with standard prompt templates) on the three used datasets of requirements. In addition, we collect and provide a large-scale labeled dataset, namely NFR-SO, on which Trans_PRCBERT exhibits excellent transfer- ability and generalization capabilities with good zero-shot performance (w-F score of 53.27% and 72.96% on NFR-Review and NFR-SO with self-learning strategy enabled). In future, we will apply and improve our approach to other classification problems in requirement engineering and software engineering.

ACKNOWLEDGMENTS

This work is supported by the Basic Research Program of Jiangsu Province (Grant No. BK20201192). Dr. Xue's research is also supported by CAS Pioneer Hundred Talents Program.

REFERENCES

[1] Zahra Shakeri Hossein Abad, Oliver Karras, Parisa Ghazi, Martin Glinz, Guenther Ruhe, and Kurt Schneider. 2017. What works better? a study of classifying requirements. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*. IEEE, 496–501.

[2] Zahra Shakeri Hossein Abad, Oliver Karras, Parisa Ghazi, Martin Glinz, Guenther Ruhe, and Kurt Schneider. 2017. What works better? a study of classifying requirements. In *2017 IEEE 25th International Requirements Engineering Conference (RE)*. IEEE, 496–501.

[3] Alan Akbik, Duncan Blythe, and Roland Vollgraf. 2018. Contextual String Embeddings for Sequence Labeling. In *COLING 2018, 27th International Conference on Computational Linguistics*. 1638–1649.

[4] Sousuke Amasaki and Pattara Leelaprute. 2018. The effects of vectorization methods on non-functional requirements classification. In *2018 44th Euromicro Conference on Software Engineering and Advanced Applications (SEAA)*. IEEE, 175–182.

[5] Dzmitry Bahdanau, Kyunghyun Cho, and Yoshua Bengio. 2015. Neural Machine Translation by Jointly Learning to Align and Translate. In *3rd International Conference on Learning Representations, ICLR 2015, San Diego, CA, USA, May 7-9, 2015, Conference Track Proceedings*, Yoshua Bengio and Yann LeCun (Eds.). http://arxiv.org/abs/1409.0473

[6] Cody Baker, Lin Deng, Suranjan Chakraborty, and Josh Dehlinger. 2019. Automatic multi-class non-functional software requirements classification using neural networks. In *2019 IEEE 43rd annual computer software and applications conference (COMPSAC)*, Vol. 2. IEEE, 610–615.

[7] David Soares Batista. 2016. *Large-scale semantic relationship extraction for information discovery.* Ph. D. Dissertation. INSTITUTO SUPERIOR TÉCNICO.

[8] David S Batista, Bruno Martins, and Mário J Silva. 2015. Semi-supervised bootstrapping of relationship extractors with distributional semantics. In *Proceedings of the 2015 Conference on Empirical Methods in Natural Language Processing*. 499–504.

[9] Stefanie Beyer, Christian Macho, Massimiliano Di Penta, and Martin Pinzger. 2021. What Kind of Questions Do Developers Ask on Stack Overflow? A Comparison of Automated Approaches to Classify Posts Into Question Categories. In *Software Engineering 2021, Fachtagung des GI-Fachbereichs Softwaretechnik, 22.-26. Februar 2021, Braunschweig/Virtuell (LNI, Vol. P-310)*, Anne Koziolek, Ina Schaefer, and Christoph Seidl (Eds.). Gesellschaft für Informatik e.V., 27–28. https://doi.org/10.18420/SE2021_03

[10] Manal Binkhonain and Liping Zhao. 2019. A review of machine learning algorithms for identification and classification of non-functional requirements. *Expert Systems with Applications: X* 1 (2019), 100001.

[11] Tom B. Brown, Benjamin Mann, Nick Ryder, Melanie Subbiah, Jared Kaplan, Prafulla Dhariwal, Arvind Neelakantan, Pranav Shyam, Girish Sastry, Amanda Askell, Sandhini Agarwal, Ariel Herbert-Voss, Gretchen Krueger, Tom Henighan, Rewon Child, Aditya Ramesh, Daniel M. Ziegler, Jeffrey Wu, Clemens Winter, Christopher Hesse, Mark Chen, Eric Sigler, Mateusz Litwin, Scott Gray, Benjamin Chess, Jack Clark, Christopher Berner, Sam McCandlish, Alec Radford, Ilya Sutskever, and Dario Amodei. 2020. Language Models are Few-Shot Learners. In *Advances in Neural Information Processing Systems 33: Annual Conference on Neural Information Processing Systems 2020, NeurIPS 2020, December 6-12, 2020, virtual*, Hugo Larochelle, Marc'Aurelio Ranzato, Raia Hadsell, Maria-Florina Balcan, and Hsuan-Tien Lin (Eds.). https://proceedings.neurips.cc/paper/2020/hash/1457c0d6bfcb4967418bfb8ac142f64a-Abstract.html

[12] Ranit Chatterjee, Abdul Ahmed, Preethu Rose Anish, Brijendra Suman, Prashant Lawhatre, and Smita Ghaisas. 2021. A Pipeline for Automating Labeling to Prediction in Classification of NFRs. In *2021 IEEE 29th International Requirements Engineering Conference (RE)*. IEEE, 323–323.

[13] Kevin Clark, Minh-Thang Luong, Quoc V. Le, and Christopher D. Manning. 2020. ELECTRA: Pre-training Text Encoders as Discriminators Rather Than Generators. In *8th International Conference on Learning Representations, ICLR 2020, Addis Ababa, Ethiopia, April 26-30, 2020*. OpenReview.net. https://openreview.net/forum?id=r1xMH1BtvB

[14] Zihang Dai, Zhilin Yang, Yiming Yang, Jaime G. Carbonell, Quoc Viet Le, and Ruslan Salakhutdinov. 2019. Transformer-XL: Attentive Language Models beyond a Fixed-Length Context. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, Anna Korhonen, David R. Traum, and Lluís Màrquez (Eds.). Association for Computational Linguistics, 2978–2988. https://doi.org/10.18653/v1/p19-1285

[15] Jacob Devlin, Ming-Wei Chang, Kenton Lee, and Kristina Toutanova. 2019. BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding. In *Proceedings of the 2019 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2-7, 2019, Volume 1 (Long and Short Papers)*, Jill Burstein, Christy Doran, and Thamar Solorio (Eds.). Association for Computational Linguistics, 4171–4186. https://doi.org/10.18653/v1/n19-1423

[16] Edna Dias Canedo and Bruno Cordeiro Mendes. 2020. Software requirements classification using machine learning algorithms. *Entropy* 22, 9 (2020), 1057.

[17] Edna Dias Canedo and Bruno Cordeiro Mendes. 2020. Software requirements classification using machine learning algorithms. *Entropy* 22, 9 (2020), 1057.

[18] Li Dong, Nan Yang, Wenhui Wang, Furu Wei, Xiaodong Liu, Yu Wang, Jianfeng Gao, Ming Zhou, and Hsiao-Wuen Hon. 2019. Unified language model pre-training for natural language understanding and generation. *Advances in Neural Information Processing Systems* 32 (2019).

[19] Jannik Fischbach, Julian Frattini, Arjen Spaans, Maximilian Kummeth, Andreas Vogelsang, Daniel Mendez, and Michael Unterkalmsteiner. 2021. Automatic detection of causality in requirement artifacts: the cira approach. In *International*

Working Conference on Requirements Engineering: Foundation for Software Quality. Springer, 19–36.

[20] Atsushi Fujii, Kentaro Inui, Takenobu Tokunaga, and Hozumi Tanaka. 1998. Selective Sampling for Example-based Word Sense Disambiguation. *Comput. Linguistics* 24, 4 (1998), 573–597.

[21] Xi Gong, Zhenchang Xing, Xiaohong Li, Zhiyong Feng, and Zhuobing Han. 2019. Joint Prediction of Multiple Vulnerability Characteristics Through Multi-Task Learning. In *24th International Conference on Engineering of Complex Computer Systems, ICECCS 2019, Guangzhou, China, November 10-13, 2019*, Jun Pang and Jing Sun (Eds.). IEEE, 31–40. https://doi.org/10.1109/ICECCS.2019.00011

[22] Alan R Hevner, Salvatore T March, Jinsoo Park, and Sudha Ram. 2004. Design science in information systems research. *MIS quarterly* (2004), 75–105.

[23] Tobias Hey, Jan Keim, Anne Koziolek, and Walter F Tichy. 2020. NoRBERT: Transfer learning for requirements classification. In *2020 IEEE 28th International Requirements Engineering Conference (RE)*. IEEE, 169–179.

[24] ISO/IEC. 2011. Systems and software engineering – Systems and software Quality Requirements and Evaluation (SQuaRE) – System and software quality models. (2011).

[25] Rie Johnson and Tong Zhang. 2017. Deep pyramid convolutional neural networks for text categorization. In *Proceedings of the 55th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*. 562–570.

[26] Hong Jin Kang and David Lo. 2022. Active Learning of Discriminative Subgraph Patterns for API Misuse Detection. *CoRR* abs/2204.09945 (2022). https://doi.org/10.48550/arXiv.2204.09945 arXiv:2204.09945

[27] Urvashi Khandelwal, He He, Peng Qi, and Dan Jurafsky. 2018. Sharp Nearby, Fuzzy Far Away: How Neural Language Models Use Context. In *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics, ACL 2018, Melbourne, Australia, July 15-20, 2018, Volume 1: Long Papers*, Iryna Gurevych and Yusuke Miyao (Eds.). Association for Computational Linguistics, 284–294. https://doi.org/10.18653/v1/P18-1027

[28] Kyoung-jae Kim. 2003. Financial time series forecasting using support vector machines. *Neurocomputing* 55, 1-2 (2003), 307–319.

[29] Tae Kyun Kim. 2015. T test as a parametric statistic. *Korean journal of anesthesiology* 68, 6 (2015), 540.

[30] Siwei Lai, Liheng Xu, Kang Liu, and Jun Zhao. 2015. Recurrent convolutional neural networks for text classification. In *Twenty-ninth AAAI conference on artificial intelligence*.

[31] Jinfeng Lin, Yalin Liu, Qingkai Zeng, Meng Jiang, and Jane Cleland-Huang. 2021. Traceability transformed: Generating more accurate links with pre-trained bert models. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 324–335.

[32] Pengfei Liu, Xipeng Qiu, and Xuanjing Huang. 2016. Recurrent Neural Network for Text Classification with Multi-Task Learning. In *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence, IJCAI 2016, New York, NY, USA, 9-15 July 2016*, Subbarao Kambhampati (Ed.). IJCAI/AAAI Press, 2873–2879. http://www.ijcai.org/Abstract/16/408

[33] Xiaodong Liu, Pengcheng He, Weizhu Chen, and Jianfeng Gao. 2019. Multi-Task Deep Neural Networks for Natural Language Understanding. In *Proceedings of the 57th Conference of the Association for Computational Linguistics, ACL 2019, Florence, Italy, July 28- August 2, 2019, Volume 1: Long Papers*, Anna Korhonen, David R. Traum, and Llu´
is Màrquez (Eds.). Association for Computational Linguistics, 4487–4496. https://doi.org/10.18653/v1/p19-1441

[34] Xiao Liu, Fanjin Zhang, Zhenyu Hou, Li Mian, Zhaoyu Wang, Jing Zhang, and Jie Tang. 2021. Self-supervised Learning: Generative or Contrastive. *IEEE Transactions on Knowledge and Data Engineering* (2021), 1–1. https://doi.org/10.1109/TKDE.2021.3090866

[35] Yinhan Liu, Myle Ott, Naman Goyal, Jingfei Du, Mandar Joshi, Danqi Chen, Omer Levy, Mike Lewis, Luke Zettlemoyer, and Veselin Stoyanov. 2019. Roberta: A robustly optimized bert pretraining approach. *arXiv preprint arXiv:1907.11692* (2019).

[36] Ilya Loshchilov and Frank Hutter. 2019. Decoupled Weight Decay Regularization. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net. https://openreview.net/forum?id=Bkg6RiCqY7

[37] Andrew L. Maas, Raymond E. Daly, Peter T. Pham, Dan Huang, Andrew Y. Ng, and Christopher Potts. 2011. Learning Word Vectors for Sentiment Analysis. In *Proceedings of the 49th Annual Meeting of the Association for Computational Linguistics: Human Language Technologies*. Association for Computational Linguistics, Portland, Oregon, USA, 142–150. http://www.aclweb.org/anthology/P11-1015

[38] Rohan Reddy Mekala, Asif Irfan, Eduard C Groen, Adam Porter, and Mikael Lindvall. 2021. Classifying user requirements from online feedback in small dataset environments using deep learning. In *2021 IEEE 29th International Requirements Engineering Conference (RE)*. IEEE, 139–149.

[39] Ana Moreira, João Araújo, Jane Hayes, and Barbara Paech (Eds.). 2017. *25th IEEE International Requirements Engineering Conference, RE 2017, Lisbon, Portugal, September 4-8, 2017*. IEEE Computer Society. https://ieeexplore.ieee.org/xpl/conhome/8048783/proceeding

[40] Aaditya Prakash, James Storer, Dinei Florencio, and Cha Zhang. 2019. Repr: Improved training of convolutional filters. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*. 10666–10675.

[41] Alec Radford, Jong Wook Kim, Chris Hallacy, Aditya Ramesh, Gabriel Goh, Sandhini Agarwal, Girish Sastry, Amanda Askell, Pamela Mishkin, Jack Clark, et al. 2021. Learning transferable visual models from natural language supervision. In *International Conference on Machine Learning*. PMLR, 8748–8763.

[42] Colin Raffel, Noam Shazeer, Adam Roberts, Katherine Lee, Sharan Narang, Michael Matena, Yanqi Zhou, Wei Li, and Peter J. Liu. 2020. Exploring the Limits of Transfer Learning with a Unified Text-to-Text Transformer. *J. Mach. Learn. Res.* 21 (2020), 140:1–140:67. http://jmlr.org/papers/v21/20-074.html

[43] A Rakhlin. 2016. Convolutional neural networks for sentence classification. *GitHub* (2016).

[44] Alexander Ratner, Stephen H Bach, Henry Ehrenberg, Jason Fries, Sen Wu, and Christopher Ré. 2017. Snorkel: Rapid training data creation with weak supervision. In *Proceedings of the VLDB Endowment. International Conference on Very Large Data Bases*, Vol. 11. NIH Public Access, 269.

[45] Siva Reddy, Danqi Chen, and Christopher D Manning. 2019. Coqa: A conversational question answering challenge. *Transactions of the Association for Computational Linguistics* 7 (2019), 249–266.

[46] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. In *NeurIPS EMC$\hat{2}$ Workshop*.

[47] S. J. Sayyad. 2005. PROMISE Software Engineering Repository. (2005).

[48] Timo Schick and Hinrich Schütze. 2021. Exploiting Cloze-Questions for Few-Shot Text Classification and Natural Language Inference. In *Proceedings of the 16th Conference of the European Chapter of the Association for Computational Linguistics: Main Volume, EACL 2021, Online, April 19 - 23, 2021*, Paola Merlo, Jörg Tiedemann, and Reut Tsarfaty (Eds.). Association for Computational Linguistics, 255–269. https://doi.org/10.18653/v1/2021.eacl-main.20

[49] Timo Schick and Hinrich Schütze. 2021. It's Not Just Size That Matters: Small Language Models Are Also Few-Shot Learners. In *Proceedings of the 2021 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies, NAACL-HLT 2021, Online, June 6-11, 2021*, Kristina Toutanova, Anna Rumshisky, Luke Zettlemoyer, Dilek Hakkani-Tür, Iz Beltagy, Steven Bethard, Ryan Cotterell, Tanmoy Chakraborty, and Yichao Zhou (Eds.). Association for Computational Linguistics, 2339–2352. https://doi.org/10.18653/v1/2021.naacl-main.185

[50] László Tóth and László Vidács. 2018. Study of various classifiers for identification and classification of non-functional requirements. In *International Conference on Computational Science and Its Applications*. Springer, 492–503.

[51] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).

[52] Alex Wang, Amanpreet Singh, Julian Michael, Felix Hill, Omer Levy, and Samuel R. Bowman. 2019. GLUE: A Multi-Task Benchmark and Analysis Platform for Natural Language Understanding. In *7th International Conference on Learning Representations, ICLR 2019, New Orleans, LA, USA, May 6-9, 2019*. OpenReview.net. https://openreview.net/forum?id=rJ4km2R5t7

[53] Tianlu Wang, Peng Liang, and Mengmeng Lu. 2018. What aspects do non-functional requirements in app user reviews describe? an exploratory and comparative study. In *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 494–503.

[54] Claes Wohlin, Per Runeson, Martin Höst, Magnus C Ohlsson, Björn Regnell, and Anders Wesslén. 2012. *Experimentation in software engineering*. Springer Science & Business Media.

[55] Zhilin Yang, Zihang Dai, Yiming Yang, Jaime Carbonell, Russ R Salakhutdinov, and Quoc V Le. 2019. Xlnet: Generalized autoregressive pretraining for language understanding. *Advances in neural information processing systems* 32 (2019).

[56] Zichao Yang, Diyi Yang, Chris Dyer, Xiaodong He, Alex Smola, and Eduard Hovy. 2016. Hierarchical attention networks for document classification. In *Proceedings of the 2016 conference of the North American chapter of the association for computational linguistics: human language technologies*. 1480–1489.

[57] Manzil Zaheer, Guru Guruganesh, Kumar Avinava Dubey, Joshua Ainslie, Chris Alberti, Santiago Ontanon, Philip Pham, Anirudh Ravula, Qifan Wang, Li Yang, et al. 2020. Big bird: Transformers for longer sequences. *Advances in Neural Information Processing Systems* 33 (2020), 17283–17297.

[58] Liping Zhao, Waad Alhoshan, Alessio Ferrari, Keletso J Letsholo, Muideen A Ajagbe, Erol-Valeriu Chioasca, and Riza T Batista-Navarro. 2021. Natural language processing for requirements engineering: a systematic mapping study. *ACM Computing Surveys (CSUR)* 54, 3 (2021), 1–41.

[59] Peng Zhou, Wei Shi, Jun Tian, Zhenyu Qi, Bingchen Li, Hongwei Hao, and Bo Xu. 2016. Attention-based bidirectional long short-term memory networks for relation classification. In *Proceedings of the 54th annual meeting of the association for computational linguistics (volume 2: Short papers)*. 207–212.