Who Should Test the Requirement? A Comparative Study on Requirements Classification for Assigning Test Teams using the Pre-Trained Models

Alay Nascimento
Sidia Institute of Science and Technology
Manaus, Brazil
alay.nascimento@sidia.com

Flávia Oliveira
Sidia Institute of Science and Technology
Manaus, Brazil
flavia.oliveira@sidia.com

Leonardo Tiago
Sidia Institute of Science and Technology
Manaus, Brazil
leonardo.albuquerque@sidia.com

Lennon Chaves
Sidia Institute of Science and Technology
Manaus, Brazil
lennon.chaves@sidia.com

ABSTRACT

Analyzing software requirements is a complex task, particularly for projects with a large volume of requirements, and when conducted manually, this task is time-consuming and prone to human errors. Moreover, once the software implements the requirements, it is essential to conduct tests to ensure the correct validation. Within a software institute, each new requirement can be assigned to a test team (teams 1 and 2) responsible for ensuring coverage by updating or creating test cases. There are instances in which a requirement is not assigned to either the team or is assigned to both. Each test team is tasked with validating a specific scope of requirements, making it crucial that each requirement is analyzed and validated by an appropriate test team. If a test team fails to validate a requirement within its scope, it can result in software vulnerability. To mitigate these issues, this paper described the use of pre-trained models, such as BERT, XLNet, and ELECTRA, to automate the process of requirement classification, thereby determining which test team should validate each new requirement. We compared the models based on accuracy, precision, recall, F1-Score, and Area Under Curve (AUC) macro metrics. Notably, the XLNet model demonstrated superior performance among the models, achieving 93.16% AUC Macro, while the BERT model achieved 91.28% AUC, and the ELECTRA model achieved 90.17% AUC. We also applied the non-parametric Friedman test to statistically validate the results, followed by the Conover squared rank test, with a significance level of 0.05. The results indicate that XLNet outperformed the BERT and ELECTRA models, exhibiting a superior capacity for assigning requirements to the correct test teams. Given the promising results of this research, this study aims to demonstrate the viability of using pre-trained models as a solution to optimize the testing process in the software industry.

KEYWORDS

Software Requirements, Classification, Pre-trained Model

1 Introduction

To develop software, the following phases of the Software Development Life Cycle (SDLC) [17] are applied: requirement elicitation, project and analysis, development, testing, and maintenance [26]. The test phase is essential in the software development process, as it is in this phase that the quality, reliability, and conformity of a system with specified requirements is verified [2, 26]. In this phase, the software requirements are validated and used as a reference for the development of test cases. The process of analysis and validation of requirements already requires considerable human effort, and in cases where there is a large volume of requirements, the process becomes even more complex, consequently raising the occurrence of mistakes and inconsistencies [6, 29–31].

In the context of software testing teams within a software institute, it is crucial that the analysis and validation of the requirements might be conducted by the appropriate test team. Each team is tasked with a specific testing scope to ensure that the requirements are validated accurately. This process is currently executed manually by each team, necessitating the involvement of at least two analysts (one from each team) to analyze and manually assign the requirements to the responsible test team. Given the large volume of requirements, this process is time-consuming and susceptible to errors. Moreover, inaccuracies in the analysis and validation of requirements significantly contribute to the failure of software products [14].

To facilitate the process of assigning requirements for each test team, this paper proposes an investigation of the use of the pre-trained models, such as: BERT (Bidirectional Encoder Representations from Transformers) [7], XLNet [33] and ELECTRA (Efficiently Learning an Encoder that Classifies Token Replacements Accurately) [4]. The use of pre-trained models aims to automate the process of classifying software requirements, with the goal of identifying which test team must be responsible for the validation of the requirement. For this investigation, we built a dataset with 692 real requirements from the test teams, and then, we applied text processing techniques to the dataset, and finally, we trained the models (BERT, XLNet and ELECTRA). After training, we tested the models and evaluated based on metrics such as Accuracy, Precision, Recall, F1-Score and AUC. Lastly, aiming to statistically validate the results of this study, we performed the Friedman test [11] and Conover squared ranks test [5], with a significance level of 0.05.

2 Theoretical Reference

2.1 Requirement Classification

A requirement can be defined as a demand, feature, or attribute of a system. It can be an obligation or a necessary skill for a user to solve a problem or reach a goal, or something that needs to be accomplished or maintained by a system or part of a system to conform to a contract, standard, specification, or other mandatory document

[8]. Requirement Engineering activity involves collecting, documenting, approving, analyzing, and controlling requirements [22], which is essential for the effectiveness of the software development process [30]. This activity has a considerable hurdle: the manual classification of software requirements, which can require plenty of effort owing to the subjectivity of comprehending and managing many requirements [30].

To overcome this obstacle, Fadhlurrohman et al. [9] presented the categorization of software requirements as a new strategy for generating test cases through text classification. The proposal is to classify the requirements in two categories: "none" and "both", this way demonstrating the existence or nonexistence of preconditions in software requirements. The Naive Bayes algorithm, a probabilistic classification algorithm, was used to achieve this goal, obtaining a result with 0.86 precision [9].

In another approach, Surana et al. [29] proposed an intelligent conversational chatbot that uses Machine Learning (ML) and Artificial Intelligence (AI) to talk with people involved in the software development process, collect requirements based on interaction, and classify these requirements into Functional Requirements (FR) and Non-Functional Requirements (NFR) [22]. To perform the classification, they used the algorithms Multinomial Naïve Bayes [24] and Support Vector Machine [20]. The Multinomial Naïve Bayes presented a relatively better performance, satisfying the need to classify requirements into Functional and Non-Functional [29]. To solve this problem, Tasnim et al. [30] proposed an Long Short-Term Memory (LSTM) model [35] based on attention. The results confirm that the model is efficient in classifying requirements, with a precision of 98% for the test dataset [30].

2.2 Pre-trained Models

A pre-trained model is characterized by a model trained with a large volume of non-labeled data for one or more tasks, so that later on, this model can be adjusted or subjected to Transfer Learning for specific tests using a considerably smaller volume of data for training [3]. Many natural language models have been pre-trained and used for applications in many fields within AI, specifically for Natural Language Processing (NLP) [19]. BERT [7] is an example of a natural language model known for being trained by masked token prediction. BERT is designed to pre-train deep bidirectional representations from unlabeled text by jointly conditioning on both the left and right contexts in all layers. The pre-trained BERT model can be fine-tuned with only one additional output layer to create state-of-the-art models for a wide range of tasks, such as question answering and language inference [7].

In contrast to the BERT model, the ELECTRA model [4] corrupts this by replacing some tokens with plausible alternatives sampled from a small generator network instead of masking the input. In this manner, the model becomes discriminative, predicting whether each token in the corrupted input is replaced by a generator sample [4]. Another natural language model is XLNet [33], which enables learning bidirectional contexts and has an autoregressive formulation that incorporates concepts from Transformer-XL [33].

Zhang et al. [36] applied BERT and XLNet to perform the task of sentiment analysis for software engineering. The results indicate that fine-tuning these models allows them to surpass the existing tools for this task by 6.5-35.6% in terms of macro/micro-averaged F1 score [36]. Fikriansyah et al. [10] evaluated the performance of ELECTRA for sentiment analysis after fine-tuning. These results highlight the importance of fine-tuning the ELECTRA model for text classification [10].

2.3 Pre-trained Models for Requirements Classification

With the emergence of pre-trained natural language models, it was possible to reduce manual effort in requirement classification tasks within Requirements Engineering. Previous research has investigated many pre-trained models destined for text classification tasks in the Software Engineering field [16]. In particular, Kici et al. [16] performed an empirical analysis of multilabel classification of software requirements using the BERT, DistilBERT, ALBERT, and XLNet models. The results reveal that the five models can classify requirements [16].

Subahi [27] presents an approach based in the fine-tuning of the BERT, aiming to classify non-functional software requirements. The results proved that this approach is efficient for classifying software requirements [27].

Khan et al. [15] proposed fine-tuning of the models XLNet, BERT, Distil BERT, Distil Roberta, ELECTRA-base, and ELECTRA-small to automate the identification and classification of Non-Functional Requirements. The study analyzed these models and concluded that the XLNet model presented a better performance, reaching 0.91489 in Accuracy, Precision, Recall and F1-Score, highlighting the importance of performing fine-tuning in the XLNet model, facilitating the process of software development, and optimizing the use of resources [15].

3 Requirements Analysis Process

This Section describes the workflow for receiving, analyzing, and classifying the requirements within the software institute. This process is conducted by 2 test teams, each of which is responsible for a specific group of requirements. It is possible that both teams validate the same requirement, as a team may validate it in a regression test, whereas the other team validates the requirement in an acceptance test. Team 1 validates the requirements of sanitation and regression test scopes. Team 2 validates the requirements of the acceptance tests. The requirements analysis process occurs in the following manner:

(1) The client communicates the requirement for the software institute.
(2) The requirement is documented via an internal tool for controlling requirements.
(3) One analyst from each of the test teams performs the analysis and determines if the requirement is applicable for their team's test scope. The requirements are then classified and assigned according to the following labels:
    - **Team 1:** Requirements that are validated by the test scope of team 1 (in the sanity and regression test scopes);
    - **Team 2:** Requirements that are validated by the test scope of team 2 (in the acceptance test scope);
    - **None:** Requirements that are validated by neither team;
    - **All:** Requirements that are validated at the same time by the test scopes of team 1 and team 2.
(4) If the classification is "Team 1", "Team 2" or "All", the analyst:
    - Identifies an existing test case that is associated to the validation of the requirement and updates the test case, or
    - Creates a new test case for validation.

(5) Lastly, the analyst adds to the documented requirement a comment indicating which technical team is responsible for validating the requirement.

This workflow presents the following problems:

- Errors due to the subjectivity of the interpretation of the requirement;
- Time spent analyzing manually;
- Dependence of the analyst's expertise.

To solve these problems, this study proposes the automation of the requirement classification step (item 3) through the fine-tuning of pre-trained models (BERT, XLNet, and ELECTRA). The next section details how this automation was performed.

4 Methodology

With the goal of validating the proposition of using pre-trained models, in this section we present the necessary steps to evaluate the viability of the use of these models for classifying requirements in the software institute. The objective of this study is to identify the most adequate model for correctly assigning the requirement for the test team responsible for its validation.

In Figure 1, the steps for evaluating the training of the models are presented, including the development of the dataset, preprocessing of data, training of the models with the data, evaluation and testing of the models, and planning of the experimental design.

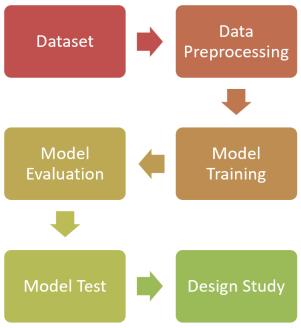

Figure 1: Step-by-Step of the Pre-Trained Model Evaluation

4.1 Dataset

As a first step in the methodology, we built a dataset. This dataset was built based on real requirements from the software institute, which were already analyzed and validated by the test team. In summary, 692 requirements[1] are collected from 2023 to 2024. These requirements were then labeled among the 4 classes, as presented in Section 3. Table 1 presents the number of requirements divided by class.

Table 1: Requirements Class by Teams

Class	Quantity
All	205
Time 1	190
None	177
Time 2	120

[1]Due to non-disclosure agreement, we can not share the list of requirements.

4.2 Data Preprocessing

In the second step, the dataset was preprocessed. In particular, we applied regular expression techniques to remove accents and stop words and stemming [23], as well as to normalize the dataset.

4.2.1 Data Cleaning. Regular expressions were applied to remove special characters, line breaks, punctuation, digits, double spaces, and hyperlinks, ensuring that only the relevant tokens remained for the model's learning. We removed stop words to reduce words that did not make sense of the text. Stemming implements a reduction of words to their roots, helping to compare morphologically associated words [15].

4.2.2 Data Balancing. Regarding the number of requirements by class, we have an unbalanced dataset. To solve this issue, synthetic data were generated based on the synonyms of the original basis and added. These synonyms are generated using the WordNet framework [18]. With the balancing of the base, a new number of requirements by class was obtained, as shown in Table 2. For these classes, it was necessary to perform integer coding using the LabelEncoder framework [1]. Then, we started the training of the model by dividing the dataset into 80% for training and 20% for validation through stratification [21], and the tokenizers of each model were used [4, 7, 33].

Table 2: Requirements Class by Teams after Balancing

Class	Quantity
Time 1	727
Time 2	727
None	727
All	727

4.3 Training

After stratification and codification, we organized and stored the data samples and their respective labels in the dataset. The variants of each model were loaded as shown in Table 3. The following settings were defined for the training parameters: 13 epochs, batch size of 16 for training and 32 for validation, and learning rate of 5e-5. To avoid overfitting the models, we applied Early Stopping [34] during training whenever 5 epochs went by without improvements in the validation loss.

The approach employed in this study was the fine-tuning of the pre-trained models, that is, adjusting the weights in the multilabel classification task. This approach consists of adding a linear layer and updating the weights of the model and layer [28].

Table 3: Pre-trained Models Settings

Model	Settings
bert-base-uncased[2]	12-layer, 768-hidden, 12-heads, 110M parameters
xlnet-base-cased[3]	12-layer, 768-hidden, 12-heads, 110M parameters
electra-base-discriminator[4]	12-layer, 768-hidden, 12-heads, 110M parameters

[2]https://huggingface.co/google/electra-base-discriminator
[3]https://huggingface.co/xlnet/xlnet-base-cased
[4]https://huggingface.co/google/electra-base-discriminator

4.4 Model Evaluation

The performance of the models was measured using the following metrics: accuracy, precision, Recall, F1-Score and AUC (Area Under the Curve) [8, 13]. In particular, the accuracy measures the number of correct classifications, whereas precision measures the proportion of correct classifications among all positive classifications [8]. Accuracy is calculated through Equation 1. Precision is calculated using Equation 2.

$$\textbf{Accuracy} = \frac{(TP + TN)}{(TP + TN + FP + FN)} \tag{1}$$

$$\textbf{Precision} = \frac{TP}{TP + FP} \tag{2}$$

Recall measures the proportion of positive samples correctly identified [8], as illustrated in Equation 3.

$$\textbf{Recall} = \frac{TP}{TP + FN} \tag{3}$$

The combination of the Precision and Recall metrics can be measured through the F1-Score metrics, which is the harmonic average of Precision and Recall, identifying all positives and avoiding false positives [8]. F1-Score is calculated using Equation 4.

$$\textbf{F1-score} = 2 \times \left(\frac{Precision \times Recall}{Precision + Recall} \right) \tag{4}$$

The AUC metric calculates the performance of the models and refers to the probability that the classifier assigns a higher score to a positive example when compared to a negative example when both are chosen randomly. The AUC is calculated using the area under the ROC curve (Receiver Operating Characteristic), which correlates the performance of classifiers in many classes [13]. With the macro-average, it is possible to calculate the ROC curve for each class individually and then take the simple average of the curves. Using these metrics, it was possible to evaluate the performance of each model in the task [32]. The formula that calculates the AUC metric is represented in Equation 5.

$$\textbf{Macro-AUC}(f) = \frac{1}{K} \sum_{k=1}^{K} \frac{1}{|S_k^+||S_k^-|} \sum_{(p,q) \in S_k^+ \times S_k^-} \mathbb{I}(f_k(x_p) > f_k(x_q)) \tag{5}$$

Onde:

- TP: quantity of true positives.
- TN: quantity of true negatives.
- FP: quantity of false positives.
- FN: quantity of false negatives.
- K: total number of classes.
- S_k^+: set of indexes of the positive examples for class k.
- S_k^-: set of indexes of the negative examples for class k.
- x_p: example of input with index p, positive for class k.
- x_q: example of input with index q, negative for class k.
- $f_k(x)$: probability assigned by the model to example x for class k.
- $\mathbb{I}(\cdot)$: indicator function, that returns 1 if the condition is true and 0 if the condition is false.

4.5 Model Test

In this step, we performed the test of the models with requirements not yet known by them. Thus, the models saved after training were loaded with their respective tokenizers. The new requirements were consulted and extracted for the test, and their classes were processed.

To ensure that the dataset was balanced, only 25 requirements from each class were randomly chosen, i.e., a total of 100 requirements. The prediction in this base reproduced the same processing environment as the training. We calculated the metrics and generated a classification report by using the metrics calculated for each class.

4.6 Study Planning

4.6.1 Hypothesis. In this step we developed the hypothesis, context, variable selection and subject selection:

- **H0**: There is no difference between the models BERT, XLNet and ELECTRA in regards to the probability assigned to the correct class.
- **H1**: There is a difference between the models BERT, XLNet and ELECTRA in regards to the probability assigned to the correct class.

4.6.2 Context. We performed tests with models BERT, XLNet, and ELECTRA, and 100 new requirements were collected aiming achieve this. The goal is to evaluate how well the three models can assign the correct class for each requirement; that is, whether the model can determine which test team must validate each requirement. To achieve this, we performed an experiment to compare the probabilities of each model assigning the correct class (the test team responsible for the requirement), with the goal of checking whether there are any significant differences between them.

4.6.3 Variable selection. The independent variable is the model used in the study (BERT, XLNet and ELECTRA), and the dependent variable is the distribution of the probability[5] for assigning the correct class. For this experiment, the probabilities were calculated as follows:

(1) The models predicted the probabilities of a requirement belonging to each class for the 100 requirements used in the test dataset.
(2) The probabilities predicted for each model were stored.

These probabilities characterize how sure each model was when correctly classifying the requirement. The set of probabilities were employed during statistical tests.

4.6.4 Subject selection. The subjects of the experiments were 100 new requirements collected and stored in the test dataset, with each requirement is associated with the corresponding class.

5 Experiments and Results

In this section, we describe the experiments performed after the training of the BERT, XLNet, and ELECTRA models, as detailed in Section 4. The evaluation of the performance of each model in the training, validation, and test sets is also presented, as well as comparisons between them and the application of a statistical test to check the difference in performance among the models.

5.1 Performance by Dataset

As mentioned in Section 4, the dataset was divided into 80% for training and 20% for validation with 100 new requirements for testing the models. Table 4 summarizes the results for each model in the training and validation sets using accuracy, precision, recall

[5] Available at https://doi.org/10.5281/zenodo.15484550

and F1-Score metrics. Table 5 shows the performance of the models with the test set using the AUC Macro-average metric.

Table 4: Models Performance Results

Model	Dataset	Accuracy	Precision	Recall	F1-score
BERT	Training	97.33%	97.39%	97.33%	97.34%
BERT	Validation	94.32%	94.39%	94.33%	94.33%
XLNet	Training	98.49%	98.53%	98.49%	98.50%
XLNet	Validation	95.18%	95.33%	95.18%	95.21%
ELECTRA	Training	98.53%	98.57%	98.53%	98.54%
ELECTRA	Validation	94.84%	94.86%	94.84%	94.84%

Based on these results, the three models present satisfactory results in the training and validation steps, indicating their capacity to identify and learn patterns in the requirements. When evaluated in the test set with new data, the models displayed a small difference when compared to the results obtained in the validation step. As shown in Table 4, the XLNet model achieved 98.49% Accuracy and Recall metrics, 98.53% Precision, and a 98.50% F1-Score when considering the training set. For the validation set, the model obtained 95.18% Accuracy and Recall, 95.33% Precision, and a 95.21% F1-Score.

The BERT and ELECTRA models showed lower performance when compared to XLNet; however, their results were very close to each other. The BERT model obtained 97.33% Accuracy and Recall, 97.39% Precision, and 97.43% in F1-Score in the training set. In the validation set, the model obtained 94.32% Accuracy, 94.39% Precision, and 94.33% Recall and F1-Score. Meanwhile, for the training set, the ELECTRA model obtained 98.53% Accuracy and Recall, 98.57% Precision, and a 98.54% F1-Score. In the validation set, the ELECTRA model obtained 94.84% Accuracy, Recall and F1-Score and 94.86% Precision.

Table 5: Performance of models on the Test Dataset

Model	Accuracy	Precision	Recall	F1-score	AUC Macro
BERT	70.62%	74.34%	70.63%	70.63%	91.28%
XLNet	85.00%	87.04%	85.00%	85.18%	93.16%
ELECTRA	71.88%	74.06%	71.88%	71.66%	90.17%

In the test set, the BERT and ELECTRA models displayed similar performance. As shown in Table 5, the BERT model obtained 70.62% Accuracy and Recall, 74.34% Precision, 70.63% F1-Score, and 91.28% in the AUC Macro metric. The ELECTRA model obtained 71.88% Accuracy and Recall, 74.06% Precision, 71.66% F1-Score and, 90.17% in AUC Macro. However, the XLNet model obtained a better score in the metrics when compared to the BERT and ELECTRA models, with 85.00% Accuracy and Recall, 87.04% Precision, 85.18% F1-Score and 93.16% in AUC Macro.

To illustrate the performance of the three models, we generate the ROC curve presented in Figure 2. These curves represent the AUC metrics for each model. When observing Table 5, the XLNet model obtained 93.16% in the AUC Macro metric, which allows us to observe that this model is slightly better at distinguishing the requirements when compared to the BERT and ELECTRA models, which obtained 91.28% and 90.17% in the AUC Macro metric, respectively, demonstrating that these models make the correct decisions most of the time. However, we still can not conclude which model has better performance only looking for ROC curve or AUC metric.

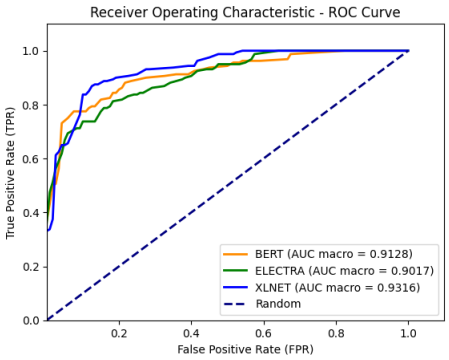

Figure 2: ROC curve produced on the test set for the BERT, XLNet and ELECTRA models

Similarly, we implemented a boxplot to illustrate the distribution of probabilities for each model's accurate class prediction. As depicted in Figure 3, the boxplot alone does not allow us to ascertain the superiority of one model over the other. Consequently, it is crucial to employ statistical tests to interpret the results thoroughly. The boxplot was implemented using JASP[6] in version 0.19.1.

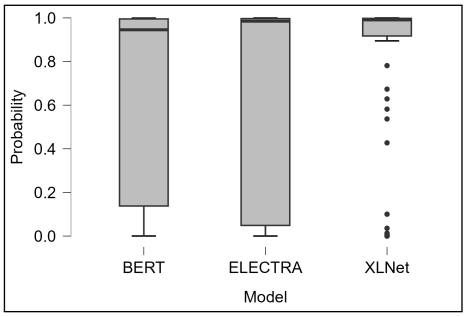

Figure 3: Boxplot of probability distribution to predict the right class

5.2 Statistical Test

The goal of the statistical test was to investigate whether there is a significant difference between the BERT, XLNet, and ELECTRA models with regard to the probability assigned to the correct class of each requirement. For the statistical test, we used a set of probability distribution to the model classify correctly each class.

Initially, we used the Shapiro-Wilk test[7] [25] to verify the normality of the probability distribution, where the test's null hypothesis refers to the population having a normal distribution. The results of the test showed $p < 0.001$ for the pairs of models, indicating that the probabilities do not have a normal distribution. Thus, the non-parametric Friedman test [11] was chosen, with $\alpha = 0.05$ significance level, to determine whether or not there is significant difference between the probabilities assigned to the correct class by each model. The test was applied to the three grouped models,

[6] https://jasp-stats.org/
[7] Performed at JASP

obtaining $p - value = 0.000000001$, demonstrating that there is a significant difference between the models.

To verify and identify which model presents a significant difference after the Friedman test, the Conover squared ranks test was applied [5] for each pair (BERT and XLNet, XLNet and ELECTRA, BERT and ELECTRA), and the Holm method [12]. We performed the statistical tests (Friedman and Conover) using Python 3.12.5[8].

The result of the Conover test is presented in Table 6. Thus, it can be concluded that between the models BERT and ELECTRA, there is no significant difference indicated through $p - value = 0.412$ however, between the pairs BERT and XLNet and XLNet and ELECTRA, there is a significant difference, with $p - value = 0.002$ and $p - value = 0.02$ respectively. Therefore, with this result, we can reject H0 and conclude, based on the Conover test, that the XLNet model is more self-assured when classifying requirements correctly.

Table 6: Result of the Conover Test

Model	BERT	XLNet	ELECTRA
BERT	1.000000	0.002928	0.412158
XLNet	0.002928	1.000000	0.025255
ELECTRA	0.412158	0.025255	1.000000

6 Discussion

Previous studies [15, 16, 27] employed pre-trained models, but with a focus on nonfunctional requirements, performing transfer learning or fine-tuning for general software specification classification [15, 16, 27]. In our study, the focus was on the real requirements of a software institute to determine the testing team to which the requirement should be directed for validation. This objective demonstrates how pre-trained models can be adapted to an industrial scenario, enabling the automation of real-world activities.

After fine-tuning, the model that performed best in all evaluation stages was XLNet [33], outperforming the other models in terms of Accuracy, Precision, Recall, F1-score, and AUC-macro. This result suggests that for specific linguistic features of requirements, which include the business rules of developed software, the model succeeded in mastering the particularities and properties of text, implying a more assertive classification. Thus, the XLNet model is the most suitable for identifying and assigning requirements to the testing team to validate it. To prove the significance of the results and discard the possibility that the superiority of one model occurred by chance, a statistical test was performed, which statistically confirmed the outstanding performance of XLNet compared with other models.

This study makes significant contributions to potential industrial use when adjusting these models to the context of requirements. The automation of the requirement classification process provides the following:

- **Acceleration of assignment**: the requirement is automatically and quickly signaled to the responsible team.
- **Reduction of manual errors**: with the reduction of human intervention, the possibility of incorrect classification is minimized.
- **Resource optimization**: by knowing exactly which team is responsible, it is possible to better plan the allocation of professionals and deadlines.

[8]https://www.python.org/downloads/release/python-3125/

7 Remarks and Future Works

This study presented an approach to requirement classification using three pre-trained models: BERT, XLNet, and ELECTRA. The main goal was to automate the process of software requirement classification in an industrial environment, directing each requirement to the responsible team between the two software testing teams. By fine-tuning each model on a set of requirements, it was found that XLNet outperformed the other models in terms of metrics such as accuracy, precision, recall, F1-score, and AUC macro. These results were confirmed through the Friedman test, which discarded the possibility of similar performance among the three models, and then through the Conover test, which identified XLNet as the best-performing model.

Regarding its use in the industry, the automation of the requirement classification process makes important contributions to its potential use in this environment, such as through the reduction of manual errors and speed in assigning requirements to responsible teams. The application of this solution in a real-world industrial context reinforces the potential of model-based methods in Software Requirements Engineering.

Some points should be considered for the threats to validity of this study: 1) Research was conducted at a software institute with a specific development scenario and confidential policies; 2) Models were trained based on requirements met by the testing team where this study was carried out, making it impossible to generalize to the context of the entire institute; and 3) The research does not allow generalizations because it relied on a limited dataset containing 100 requirements in the model's test set, specific to the context of the software teams at the institute where this study was conducted.

In future research, we plan to incorporate new requirements that have emerged since the last training of the model to expand the dataset, thereby integrating more recent information into the knowledge base of the model. Furthermore, we aim to address the requirements currently classified as None, which has not yet been evaluated by any testing team, by assigning them for validation to one of the two teams. Finally, we plan to conduct a controlled experiment with both testing teams to assess the advantages and disadvantages of implementing the fine-tuned model within the two teams, and we can explore the integration of this technology with real-world scenario to evaluate, for example, the analysis of hours saved.

ARTIFACT AVAILABILITY

We organized the datasets of this study in the supplementary material, which is available on: https://doi.org/10.5281/zenodo.15484550.

ACKNOWLEDGMENTS

This paper is a result of the Research, Development & Innovation Project (ASTRO) performed at Sidia Institute of Science and Technology sponsored by Samsung Eletrônica da Amazônia Ltda., using resources under terms of Federal Law No. 8.387/1991, by having its disclosure and publicity in accordance with art. 39 of Decree No. 10.521/2020. The authors extend their gratitude to the Software Engineering Laboratory for its contribution to the IT infrastructure.

REFERENCES

[1] NT Abdullaev and K Oghuz. 2023. Use of Machine Learning Models for Classification of Myographic Diseases. *Biomedical Engineering* 56, 5 (2023), 353–357.

[2] A Anand and A Uddin. 2019. Importance of software testing in the process of software development. *International Journal for Scientfic Research and Development* 12, 6 (2019).

[3] Chenyang Bu, Yuxin Liu, Manzong Huang, Jianxuan Shao, Shengwei Ji, Wenjian Luo, and Xindong Wu. 2024. Layer-Wise Learning Rate Optimization for Task-Dependent Fine-Tuning of Pre-Trained Models: An Evolutionary Approach. *ACM Transactions on Evolutionary Learning* 4, 4 (2024), 1–23.

[4] Kevin Clark, Minh-Thang Luong, Quoc V Le, and Christopher D Manning. 2020. Electra: Pre-training text encoders as discriminators rather than generators. *arXiv preprint arXiv:2003.10555* (2020).

[5] William Jay Conover. 1999. *Practical nonparametric statistics*. john wiley & sons.

[6] Richard A DeMillo. 2003. Software testing. In *Encyclopedia of Computer Science*. 1645–1649.

[7] 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, volume 1 (long and short papers)*. 4171–4186.

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

[9] Daffa Hilmy Fadhlurrohman, Mira Kania Sabariah, Muhammad Johan Alibasa, and Jati Hiliamsyah Husen. 2023. Naive Bayes Classification Model for Precondition-Postcondition in Software Requirements. In *2023 International Conference on Data Science and Its Applications (ICoDSA)*. IEEE, 123–128.

[10] Muhammad Fikriansyah, Hilal Nuha, and Muhammad Santriaji. 2023. A Deep Dive into Electra: Transfer Learning for Fine-Grained Text Classification on SST-2. In *2023 6th International Seminar on Research of Information Technology and Intelligent Systems (ISRITI)*. IEEE, 89–94.

[11] Milton Friedman. 1937. The use of ranks to avoid the assumption of normality implicit in the analysis of variance. *Journal of the american statistical association* 32, 200 (1937), 675–701.

[12] Sture Holm. 1979. A simple sequentially rejective multiple test procedure. *Scandinavian journal of statistics* (1979), 65–70.

[13] Jin Huang and Charles X Ling. 2005. Using AUC and accuracy in evaluating learning algorithms. *IEEE Transactions on knowledge and Data Engineering* 17, 3 (2005), 299–310.

[14] Azham Hussain, Emmanuel OC Mkpojiogu, and Fazillah Mohmad Kamal. 2016. The role of requirements in the success or failure of software projects. *International Review of Management and Marketing* 6, 7 (2016), 306–311.

[15] Muhammad Amin Khan, Muhammad Sohail Khan, Inayat Khan, Shafiq Ahmad, and Shamsul Huda. 2023. Non functional requirements identification and classification using transfer learning model. *IEEE Access* 11 (2023), 74997–75005.

[16] Derya Kici, Aysun Bozanta, Mucahit Cevik, Devang Parikh, and Ayşe Başar. 2021. Text classification on software requirements specifications using transformer models. In *Proceedings of the 31st Annual International Conference on Computer Science and Software Engineering*. 163–172.

[17] Yu Beng Leau, Wooi Khong Loo, Wai Yip Tham, and Soo Fun Tan. 2012. Software development life cycle AGILE vs traditional approaches. In *International Conference on Information and Network Technology*, Vol. 37. 162–167.

[18] George A Miller. 1995. WordNet: a lexical database for English. *Commun. ACM* 38, 11 (1995), 39–41.

[19] Changan Niu, Chuanyi Li, Vincent Ng, Dongxiao Chen, Jidong Ge, and Bin Luo. 2023. An empirical comparison of pre-trained models of source code. In *2023 IEEE/ACM 45th International Conference on Software Engineering (ICSE)*. IEEE, 2136–2148.

[20] William S Noble. 2006. What is a support vector machine? *Nature biotechnology* 24, 12 (2006), 1565–1567.

[21] Fabian Pedregosa, Gaël Varoquaux, Alexandre Gramfort, Vincent Michel, Bertrand Thirion, Olivier Grisel, Mathieu Blondel, Peter Prettenhofer, Ron Weiss, Vincent Dubourg, et al. 2011. Scikit-learn: Machine learning in Python. *the Journal of machine Learning research* 12 (2011), 2825–2830.

[22] Klaus Pohl. 2016. *Requirements engineering fundamentals: a study guide for the certified professional for requirements engineering exam-foundation level-IREB compliant*. Rocky Nook, Inc.

[23] Martin F Porter. 1980. An algorithm for suffix stripping. *Program* 14, 3 (1980), 130–137.

[24] Irina Rish et al. 2001. An empirical study of the naive Bayes classifier. In *IJCAI 2001 workshop on empirical methods in artificial intelligence*, Vol. 3. Seattle, USA, 41–46.

[25] Samuel Sanford Shapiro and Martin B Wilk. 1965. An analysis of variance test for normality (complete samples). *Biometrika* 52, 3-4 (1965), 591–611.

[26] Navnath Shete and Avinash Jadhav. 2014. An empirical study of test cases in software testing. In *International Conference on Information Communication and Embedded Systems (ICICES2014)*. IEEE, 1–5.

[27] Ahmad F Subahi. 2023. Bert-based approach for greening software requirements engineering through non-functional requirements. *IEEE Access* 11 (2023), 103001–103013.

[28] Chi Sun, Xipeng Qiu, Yige Xu, and Xuanjing Huang. 2019. How to fine-tune bert for text classification?. In *China national conference on Chinese computational linguistics*. Springer, 194–206.

[29] Chetan Surana Rajender Kumar Surana, Dipesh B Gupta, Sahana P Shankar, et al. 2019. Intelligent chatbot for requirements elicitation and classification. In *2019 4th International Conference on Recent Trends on Electronics, Information, Communication & Technology (RTEICT)*. IEEE, 866–870.

[30] Arpa Tasnim, Nazneen Akhter, Mohotina Khanam, and Nusrat Jahan Rimi. 2023. An Attention Based LSTM Model: Automated Requirement Classification from User Story. In *2023 International Conference on Information and Communication Technology for Sustainable Development (ICICT4SD)*. IEEE, 305–309.

[31] Michael Unterkalmsteiner, Robert Feldt, and Tony Gorschek. 2014. A taxonomy for requirements engineering and software test alignment. *ACM Transactions on Software Engineering and Methodology (TOSEM)* 23, 2 (2014), 1–38.

[32] Guoqiang Wu, Chongxuan Li, and Yilong Yin. 2023. Towards understanding generalization of macro-auc in multi-label learning. In *International Conference on Machine Learning*. PMLR, 37540–37570.

[33] 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).

[34] Yuan Yao, Lorenzo Rosasco, and Andrea Caponnetto. 2007. On early stopping in gradient descent learning. *Constructive approximation* 26, 2 (2007), 289–315.

[35] Yong Yu, Xiaosheng Si, Changhua Hu, and Jianxun Zhang. 2019. A review of recurrent neural networks: LSTM cells and network architectures. *Neural computation* 31, 7 (2019), 1235–1270.

[36] Ting Zhang, Bowen Xu, Ferdian Thung, Stefanus Agus Haryono, David Lo, and Lingxiao Jiang. 2020. Sentiment analysis for software engineering: How far can pre-trained transformer models go?. In *2020 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 70–80.