An approach to identifying functional and non-functional requirements in IT-project management using deep learning models

Ruslan Bahrii[1], Tetiana Skrypnyk[1], Bohdan Romanov[1,*], Elena Zaitseva[2], Houda El Bouhissi[3] and Volodymyr Lytvynenko[4]

[1]Khmelnytskyi National University, Khmelnytskyi, Ukraine
[2] Zilina University, Univerzitná 8215, 010 26 Žilina, Slovakia
[3]University of Bejaia, 06000, Bejaia, Algeria
[4]Taras Shevchenko National University of Kyiv, 64/13, Volodymyrska str., Kyiv, 01601, Ukraine

Abstract

The accurate identification of functional (FR) and non-functional (NFR) requirements is critical for software success, yet unsupervised methods using general-purpose language models often suffer from a "semantic gap," leading to poor clustering performance. In this work, we propose a domain adaptation approach for the RoBERTa model to enhance unsupervised requirements clustering. We investigate the effectiveness of standard fine-tuning compared to contrastive learning and conduct a detailed analysis of layer-wise informativeness. Our results demonstrate that domain adaptation significantly improves clustering quality, increasing the F1-Score from ≈ 0.55 to ≈ 0.81, with the final 12th layer providing the optimal representation. Furthermore, while both fine-tuning methods achieve similar accuracy, contrastive learning proves superior in identifying anomalous requirements, effectively isolating ambiguity as outliers. We conclude that domain adaptation effectively bridges the semantic gap, offering a robust tool for automating requirements analysis in IT project management.

Keywords

Software requirements classification, unsupervised machine learning, clustering, RoBERTa, domain adaptation, contrastive learning, requirements ambiguity detection

1. Introduction

In modern software development, the accurate and effective identification of functional and non-functional requirements is critically important for the success of any project. Incorrect or incomplete classification of these requirements in the early stages can lead to significant financial losses, product release delays, and ultimately, the creation of software that does not meet user expectations [1]. Therefore, the relevance of automating this process using deep learning methods is rapidly growing [2].

Furthermore, in the current era characterized by global digitalization, this software engineering task is closely correlated with global humanitarian and environmental goals. The solution to the problem of identifying functional and non-functional requirements using deep learning correlates with the Sustainable Development Goals (SDGs) adopted by the UN, not directly, but as a fundamental accelerator tool that increases the efficiency of creating solutions to achieve these goals [3]. This can be traced through the following logical connection: Effective Software Development → High-Quality and Reliable Software → Scalable Solutions for SDGs. In the context of specific goals, this has a direct impact on Goal 9 (Industry, Innovation and Infrastructure). This goal is directly related to building resilient infrastructure and fostering innovation. Automating requirements analysis using deep learning is itself an innovation in the software development process. It allows for the creation of more complex and

ExplAI-2025: Advanced AI in Explainability and Ethics for the Sustainable Development Goals, November 07, 2025, Khmelnytskyi, Ukraine

*Corresponding author.
bahriiro@khmnu.edu.ua (R. Bahrii); skrypnykt@khmnu.edu.ua (T. Skrypnyk); romanovba@khmnu.edu.ua (B. Romanov); elena.zaitseva@fri.uniza.sk (E. Zaitseva); houda.elbouhissi@gmail.com (H. E. Bouhissi); vollyt@knu.ua (V. Lytvynenko)
0000-0001-5219-1185 (R. Bahrii); 0000-0002-8531-5348 (T. Skrypnyk); 0009-0007-2495-9362 (B. Romanov); 0000-0002-9087-0311 (E. Zaitseva); 0000-0003-3239-8255 (H. E. Bouhissi); 0000-0002-1536-5542 (V. Lytvynenko)

reliable software systems (e.g., for managing smart cities, logistics, industrial processes) faster and at a lower cost. This strengthens the technological infrastructure and the innovative potential of the economy. Automating the identification of requirements with deep learning is not just a technical improvement. It is a strategic step that enables the creation of higher-quality, more reliable, and safer software tools. These very tools are the foundation for developing innovative solutions aimed at overcoming the global challenges formulated in the Sustainable Development Goals.

Functional requirements define what a system should do, describing its specific functions and behavior. In contrast, non-functional requirements establish how the system should perform these functions, defining its attributes such as performance, security, reliability, and usability. Understanding this difference is fundamental to building a quality software product [4, 5].

Traditionally, the analysis and classification of software requirements are performed manually by business analysts and system engineers. This approach, though time-tested, has a number of significant drawbacks: it is labor-intensive and costly, subject to human error, prone to incomplete identification, conflicting requirements, and the dynamic nature of requirements, among others. This is where Deep Learning and Natural Language Processing (NLP) technologies play a key role [6]. These methods allow for the creation of models capable of automatically analyzing and classifying textual information with high accuracy. The main advantages of using deep learning for identifying functional and non-functional requirements include: automation and speed, increased accuracy, objectivity, and early detection of problems [7].

The application of methods such as convolutional and recurrent neural networks allows for effective work with textual data, identifying keywords, phrases, and semantic connections that indicate whether a requirement belongs to the functional or non-functional type [5]. Thus, the relevance of identifying functional and non-functional requirements by means of deep learning is driven by the urgent need to increase the efficiency, accuracy, and speed of the software development process. Automating this critically important stage not only saves time and resources but also significantly enhances the quality of the final product, ensuring it meets both functional expectations and requirements for performance, security, and reliability.

The main contribution of this research is a comprehensive analysis of approaches to the unsupervised clustering of software requirements based on the domain adaptation of language models. The work quantitatively demonstrates that fine-tuning is a critical step that significantly improves clustering quality compared to using the base model. The research includes a deep analysis of the informativeness of each layer of the fine-tuned model, based on which the best way to form vector representations is determined, and a conclusion is drawn about the optimality of using the final layer. Furthermore, by comparing different fine-tuning methods, the research shows that contrastive learning is an effective tool for the automatic identification of semantically ambiguous and anomalous requirements, which is important for improving their quality in the early stages of development.

The rest of the paper is structured as follows. Section 2 presents an analysis of related work in the field of software requirements classification. Section 3 provides a detailed description of the proposed research methodology. Section 4 presents and discusses the results of the conducted experiments. Finally, Section 5 formulates the general conclusions of the work and outlines directions for future research.

2. Related works

Recently, there has been a rapid development of deep learning methods for the automatic classification of software requirements (SR) into functional (FR) and non-functional (NFR). This process is critical for successful development, as NFRs define the quality attributes of a system, such as performance, security, and reliability [8]. The following is an overview of key modern approaches, their advantages, disadvantages, and existing research gaps.

Research in this field has gone through several stages of deep learning architecture development. Initially, CNN and RNN architectures (including their variants LSTM and GRU) were actively used for re-

Table 1
Advantages and disadvantages of existing approaches.

Approach	Advantages	Disadvantages	Key publications
CNN/RNN/LSTM	Automatic feature learning, outperforms traditional models	Limited context, problems with long-range dependencies	Khayashi et al. [9]
Hybrid models (CNN+RNN)	Combine the advantages of both architectures	Increased complexity, still inferior to transformers	Rahman et al. [10]
Transformers (BERT, RoBERTa)	Deep contextual understanding, SOTA accuracy, effective learning	High resource requirements, "black box" problem	Luo et al. [11], Xu [2]
Large Language Models (LLMs)	Generation and classification of requirements, zero/few-shot learning	Prone to "hallucinations", high cost of use	Almonte et al. [15], Ebrahim et al. [16]

quirements classification. CNN-based models effectively extracted local features (keywords and phrases), while RNN/LSTM models were proficient at capturing sequential dependencies in the requirement text [9]. Often, these architectures were combined to leverage the advantages of both approaches [10]. The advantages of these approaches include a significant improvement in accuracy compared to traditional machine learning methods (e.g., SVM or Naive Bayes) and the ability to automatically learn features, which eliminates the need for manual feature engineering [6]. Disadvantages include limited contextual understanding, as these models struggled with long and complex sentences due to the "vanishing gradient" problem and their limited ability to consider the full context of a requirement. Additionally, the sequential nature of RNNs slows down the training process and prevents effective parallelization.

The emergence of transformer architectures, particularly BERT (Bidirectional Encoder Representations from Transformers), was revolutionary for natural language processing and, specifically, for requirements classification. The mechanism of fine-tuning pre-trained models such as BERT, RoBERTa, and DeBERTa for the specific task of FR and NFR classification has become widely adopted [2, 11, 12]. Approaches like NoRBERT and PRCBERT demonstrate the highest accuracy to date on standard datasets such as PROMISE [11]. The advantages of this approach are: (1) Deep contextual understanding – thanks to the attention mechanism and bidirectional training, BERT considers the context of each word from both sides, allowing for a better understanding of the requirement's semantics [13]; (2) Transfer Learning – BERT is pre-trained on vast amounts of general-language texts, which allows it to achieve high accuracy even when fine-tuned on relatively small datasets of requirements [11] and (3) Higher accuracy – BERT-based models consistently demonstrate the best results, achieving F1-scores of over 90% in binary classification tasks (FR/NFR) [11].

The disadvantages include: (1) High computational requirements – fine-tuning and using large transformer models require significant computational resources (GPUs); (2) The "black box" problem – the complexity of the architecture makes it difficult to interpret why the model made a particular decision, which is critical for responsible systems [14] and (3) Sensitivity to data quality – the effectiveness of the models heavily depends on the quality and size of the data on which fine-tuning is performed (existing datasets, such as PROMISE, are relatively small and may contain outdated requirements [11]).

The latest step in the evolution of these approaches has been the application of Large Language Models (LLMs), such as the latest generation models (e.g., the GPT-4 series, Google Gemini, Anthropic Claude). Unlike BERT-like architectures, which are predominantly used for classification, LLMs are capable of performing tasks in a "zero-shot" or "few-shot" manner, as well as generating requirements or answering questions about them [15]. However, their use is associated with challenges such as a propensity for "hallucinations" (generating false information) and the high cost of use via API [16]. Table 1 summarizes the advantages and disadvantages of the existing approaches.

Despite the impressive results of transformer models, a major gap in current research lies in the so-called "semantic gap." Most works rely on standard models pre-trained on general-purpose text corpora (e.g., Wikipedia, news), which fail to capture the specific semantics and nuances of terminology inherent to the field of software engineering [17, 18]. For example, the word "performance" in a general context might refer to an artist's show, whereas in software requirements, it has a clear technical meaning (response time, resource usage). Terms like "scalability," "maintainability," and "security" have complex, multifaceted meanings that cannot be fully grasped without specialized knowledge. A model that does not "understand" the intricacies of the subject domain is forced to rely only on superficial syntactic patterns, which reduces its reliability [19].

Promising directions for solving this problem include: (1) creating domain-specific language models – developing corresponding models by pre-training or further fine-tuning existing transformers on vast amounts of texts from the software engineering domain (scientific articles, documentation, real-project requirements, Stack Overflow discussions) [17]; (2) Integration with Knowledge Graphs – combining text embeddings with embeddings obtained from knowledge graphs that formally describe software engineering concepts and the relationships between them (e.g., the hierarchy of NFR types according to the ISO 25010 standard) [20] and (3) Structure-aware Embeddings – developing methods that consider not only the text of a requirement but also its structure or relationship with other requirements, simultaneously learning representations of words and the structure of the corpus itself [20]. Thus, the future breakthrough in the accuracy and reliability of automatic requirements classification lies not so much in creating new architectures as in enriching existing models with domain knowledge through more informative and contextually-aware embeddings.

Of all the listed directions, the most direct and common method in practice for creating a domain-specific model is fine-tuning an existing powerful general-purpose language model on relevant data. This approach, known as domain adaptation, allows for the specialization of the model to the nuances of a specific industry without requiring training from scratch. However, while most research applies domain adaptation to create supervised classifiers, which require labels for each new dataset, in real-world scenarios, data are often unlabeled. Therefore, an important research question arises: how effective is domain adaptation for improving the quality of the vector representations themselves so they can be used for unsupervised clustering? Such an approach not only allows for the automation of processing new, unlabeled requirements but also for the investigation of their natural semantic structure.

To address this problem and improve the quality of unsupervised approaches, the goal of the research is formulated: to improve the quality of software requirement clustering through the domain adaptation of a pre-trained language model and the subsequent analysis of its internal representations. To achieve this goal, the following tasks are formulated: (1) to compare the quality of clustering before and after fine-tuning; (2) to conduct an analysis of the informativeness of the fine-tuned model's layers to determine the most informative layers and the best way to aggregate them; (3) to compare the effectiveness of standard fine-tuning and contrastive learning as tools for identifying and characterizing groups of semantically ambiguous requirements.

3. Methods

This section details the approach developed for the automatic unsupervised clustering of software requirements. The core of this research is a comparative analysis that aims to quantitatively evaluate the impact of a language model's domain adaptation on the separability of functional (FR) and non-functional (NFR) requirements classes in the vector space. The proposed approach is a sequential process that is applied to both the base and the fine-tuned models. The entire process can be divided into the following stages: generation of multi-layer vector representations, combination of layers and formation of the final vector, clustering, and final quality evaluation. The general scheme of this approach, illustrating the sequence of key stages, is depicted in Figure 1.

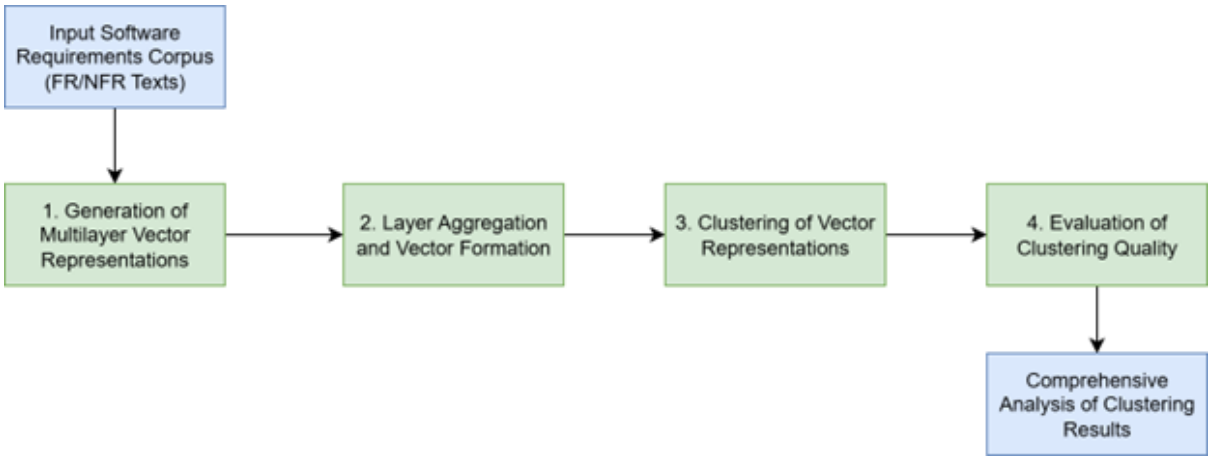

Figure 1: General scheme of the approach to unsupervised classification of software requirements.

3.1. Experimental design and model preparation

To convert textual requirements into vector representations, RoBERTa (A Robustly Optimized BERT Approach) [21] was chosen as the base architecture. This model is an ideological successor to BERT and was selected for its advantages proven in the original research. Specifically, RoBERTa was trained on a much larger dataset and used more advanced training methods (e.g., dynamic masking without the Next Sentence Prediction task), which generally leads to the creation of higher-quality and more semantically rich embeddings. In this study, its base version, roberta-base, is used.

The subsequent experiment is based on a comparative analysis of the effectiveness of two domain adaptation approaches for this model compared to its base state. The first approach is standard fine-tuning for classification, where the model is adapted on a separate domain-specific set of labeled data to solve the binary FR/NFR classification task. Additionally, a more advanced method of domain adaptation is investigated – contrastive learning [22]. Its goal is not to train the model to predict class labels, but to directly optimize the structure of the vector space. For this, pairs of requirements are formed from the training data ("positive" – from the same class, and "negative" – from different classes), and the model learns to minimize the distance between similar ones and maximize it between different ones. It is expected that such an approach will force similar requirements to group into denser and more clearly separated clusters. Thus, the subsequent stages of the method are applied in parallel to three states of the model (base, fine-tuned for classification, and contrastively fine-tuned) for an objective comparison of their effectiveness.

3.2. Input corpora and data preparation

The experiment uses two separate, publicly available datasets to avoid any overlap between training and testing data. For fine-tuning and obtaining a domain-adapted model, the PROMISE_exp dataset [23] is used. This corpus is an extension of the well-known PROMISE dataset and consists of 969 software requirements collected from 49 different projects. The dataset is relatively balanced, containing 444 functional (FR) and 525 non-functional (NFR) requirements, making it a quality source for training the model to distinguish between these two classes.

For the main clustering experiment and final evaluation, a different dataset is used – the "FR_NFR_dataset" [24]. This dataset contains a total of 6118 software requirements (3964 FR and 2154 NFR), collected from existing repositories and open-source project documentation. It is important to note that the class labels are not used during the feature extraction and clustering stages; they serve exclusively for the final evaluation of the unsupervised algorithm's performance. Since the original dataset is imbalanced (the number of FRs significantly exceeds NFRs), which could lead to a bias in the clustering results toward the dominant class, a balancing procedure was performed. To ensure

the objectivity of the experiment, a balanced subset was created from the original corpus. From each class, 1000 requirements were randomly selected. Thus, the final experimental corpus used in this work consists of 2000 software requirements, containing 1000 functional and 1000 non-functional examples. This approach ensures that the evaluation of clustering quality will not be skewed due to class imbalance.

3.3. Method steps

Step 1: Generation of Multilayer Vector Representations The first stage of the process is the generation of multilayer vector representations (embeddings) from the prepared textual requirements. For this, the pre-trained language model roberta-base [21] from the Hugging Face Transformers library is used. The representation generation process begins with tokenization, which converts each textual requirement into a standardized sequence of numerical tokens. An important detail of this process is that the tokenizer automatically adds a special service token [CLS] (Classification) to the beginning of each sequence.

The tokenized texts are fed into the input of the corresponding model version (base or fine-tuned), which operates exclusively in inference mode for this stage. This means that during the generation of representations, the model's weights are not changed. A key feature of the approach is that outputs are collected from the entire depth of the architecture for analysis: from the initial embedding layer (layer 0) and from each of the 12 transformer encoder layers (layers 1-12). This process of multilayer extraction is schematically depicted in Figure 2.

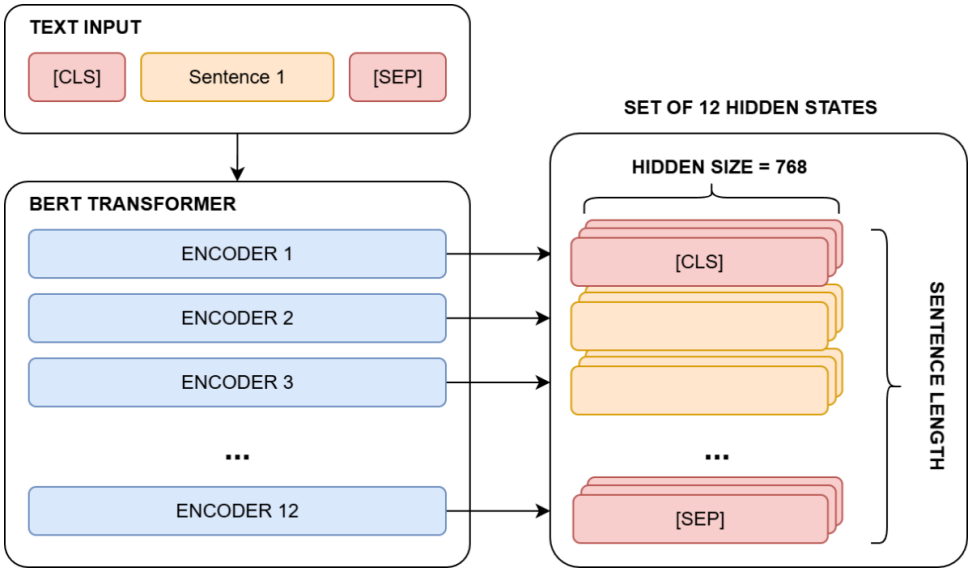

Figure 2: Schematic representation of the process of generating multilayer vector representations from the RoBERTa model.

Thus, for each requirement in the corpus, a set of vector representations is obtained at the output of this stage, which includes a separate vector for the [CLS] token from each layer. These representations, which encode information at different levels of abstraction, will serve as the basis for the next stage.

Step 2: Layer Aggregation and Vector Formation After generating multilayer representations, a systematic study is conducted to find the best way to use them. The study consists of two consecutive stages.

1. Individual analysis of layer informativeness. At this stage, the contribution of each individual layer to the quality of clustering is evaluated. For this, the [CLS] vector extracted from each of the 12 encoder layers of the respective model (base or fine-tuned) is used for clustering separately, and the quality of the resulting partition is evaluated using a set of internal and external metrics. The purpose

of this step is to create a detailed "informativeness profile" of the model, which allows for visualizing and quantitatively assessing at which levels of the architecture (lower, middle, or upper) the most useful semantic information for distinguishing between functional and non-functional requirements is contained. The results of this analysis provide the rationale for selecting strategies in the next stage.

2. Testing the hypothesis on the effectiveness of combining the best layers. Based on the results of the previous stage, which show that the highest informativeness is concentrated in the upper layers of the model, a hypothesis is tested as to whether combining these best layers can improve the result and surpass the performance of the best single layer. For this, the strategy of aggregating the last four layers (from 9th to 12th), which are the most informative according to the analysis, is investigated. Two methods of aggregating the [CLS] vectors are applied:

- Averaging produces a generalized 768-dimensional vector.
- Concatenation produces a feature-rich 3072-dimensional representation.

The results obtained in these stages are compared with the baseline approach, which is the best result from the individual analysis stage, i.e., the use of a single, most effective layer.

Step 3: Clustering of Vector Representations After a summary vector has been formed for each requirement, the next step is their grouping. Before being fed into the clustering algorithms, the vectors undergo standardization (StandardScaler). The study utilizes two clustering algorithms, each aimed at solving a separate task. For the main task of binary partitioning into FR/NFR classes, the K-Means algorithm with the parameter $k = 2$ is applied. This choice allows for the forced partitioning of the entire corpus into two clusters, which is a necessary condition for the subsequent calculation of classification quality metrics, such as the F1-Score. In parallel, to solve the task of identifying ambiguous and anomalous requirements, the density-based algorithm HDBSCAN [25] is used. The key advantage of this method is its ability not only to find clusters of arbitrary shape but also to identify "outliers" – data points that do not belong to any dense grouping. In the context of this work, these "outliers" are interpreted as anomalous or atypical requirements, which are separated into a distinct group for further analysis.

Step 4: Evaluation of Clustering Quality The final stage involves an objective evaluation of the quality of the obtained clusters to determine which vector formation strategy is the most effective. The evaluation is performed using both internal and external metrics, which together provide a comprehensive view of the partition quality.

Internal evaluation is conducted without using ground truth labels and shows the structural properties of the clusters themselves. For this, the Silhouette Score [26] is applied. This metric quantitatively evaluates how well each object fits its cluster, based on two criteria: cohesion – the measure of how similar an object is to other objects in its own cluster, and separation – the measure of how different it is from objects in other clusters. The coefficient's value ranges from -1 to +1, where: a value close to +1 indicates that the clusters are dense and well-separated from each other; a value around 0 means that the clusters overlap; and a value close to -1 suggests that the points have likely been assigned to the wrong clusters. The overall Silhouette Score for the entire dataset is the average value across all points. In scientific practice, values in the 0.5 – 0.7 range are considered to be an indication of a justified, well-defined cluster structure, while values above 0.7 indicate a strongly pronounced structure.

External evaluation uses the ground truth class labels (FR/NFR) to determine how well the clustering result corresponds to the reference distribution. Since the labels assigned by the clustering algorithm (e.g., "cluster 0" and "cluster 1") are arbitrary, a mapping step is performed before calculating the metrics. Each cluster is assigned the class label that is dominant among the points in that cluster. For example, if 95% of the requirements in "cluster 0" have the true label "FR," the entire cluster receives the label "FR." After this, the task is treated as a binary classification problem, and the F1-Score and its components are used for its evaluation: Precision shows what fraction of the items named "functional" by the clustering algorithm are indeed so. Recall shows what fraction of all "functional" requirements from the entire

dataset the clustering algorithm was able to find and place in the corresponding cluster. The F1-Score is the harmonic mean of precision and recall, which makes it an integral indicator of quality. In the research results, the F1-Score will be the key metric for comparing the effectiveness of different layer aggregation strategies. A high F1-Score will indicate that a certain strategy allows for the creation of vector representations that are not only well-separated in space but also that this separation corresponds to the human, semantic division into functional and non-functional requirements.

3.4. Experimental environment

The experiments were implemented in the Python programming language (version 3.x). The Hugging Face Transformers library (version 4.x) based on the PyTorch framework (version 2.x) was used for working with language models (roberta-base). The implementation of clustering algorithms (K-Means, HDBSCAN), the principal component analysis (PCA) method, and the calculation of quality metrics (Silhouette Score, F1-Score) were performed using the Scikit-learn library (version 1.x). All computations were conducted in the Google Colaboratory cloud environment using an NVIDIA Tesla T4 graphics processing unit (GPU).

4. Results and discussion

This section presents the results of the experimental study, along with their comparative analysis and discussion. The presentation of the results follows the logic of the research: first, the effectiveness of the base model is evaluated, followed by the domain-adapted model, after which a deep analysis of its layers' informativeness and an error analysis are conducted.

4.1. Effectiveness of Clustering with the Base Model

The first stage of the research evaluated the quality of unsupervised clustering using embeddings generated by the "raw," general-purpose roberta-base model without any fine-tuning. Despite testing various aggregation and pooling strategies, all of them showed extremely low results, only slightly exceeding the level of random guessing. The best stable result was obtained using the [CLS] vector from the 12th layer and the K-Means algorithm, which achieved an F1-Score of ≈ 0.55. A visual analysis of this experiment (Figure 3) demonstrates the key problem: although the K-Means algorithm formally partitions the data into two clusters (Figure 3a), this division does not correspond to the true class distribution (Figure 3b), as they are heavily intermingled in the vector space. This is confirmed both by the error analysis (Figure 3c) and the extremely low Truth Silhouette Score (≈ 0.015).

The model's inability to form semantically meaningful clusters is further confirmed by the results of the density-based algorithm HDBSCAN. When applied, about half of all requirements were classified as "noise" (outliers), indicating the absence of any dense, pronounced structure in the data. These results are a direct empirical confirmation of the "semantic gap." The embeddings from the general-purpose model proved to be unsuitable for the unsupervised separation of software requirements, which justifies the need for domain adaptation of the model.

4.2. Impact of domain adaptation on clustering quality

The second stage of the research evaluated the impact of prior domain adaptation (fine-tuning) of the model. The results demonstrated a radical improvement across all key metrics. The best strategy, as shown by further analysis, was the use of the embedding from the final, 12th layer of the fine-tuned model. With this strategy, an F1-Score of 0.81 was achieved, and the silhouette score for the true labels (Truth Silhouette Score) increased from a near-zero value (0.015) to 0.37. Figure 4 presents a visualization of the results for the best configuration (using the 12th layer of the fine-tuned model).

A comparison of the results before and after fine-tuning clearly demonstrates the effectiveness of domain adaptation. While the base model at best achieved an F1-Score of only ≈ 0.55, which barely

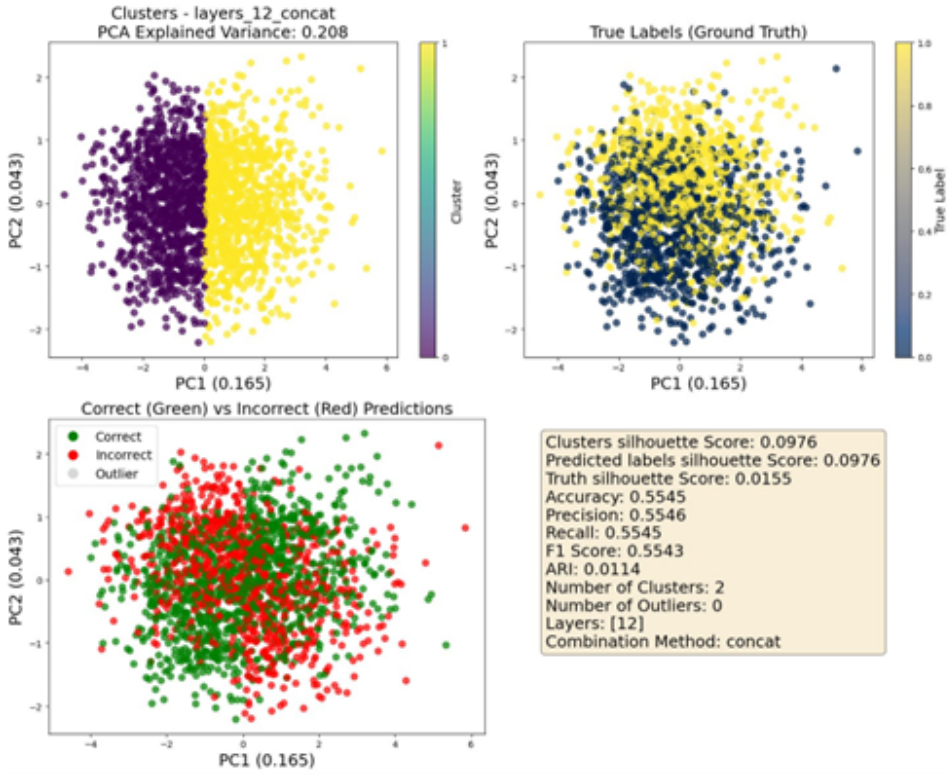

Figure 3: Visualization of clustering results for the base model (12th layer).

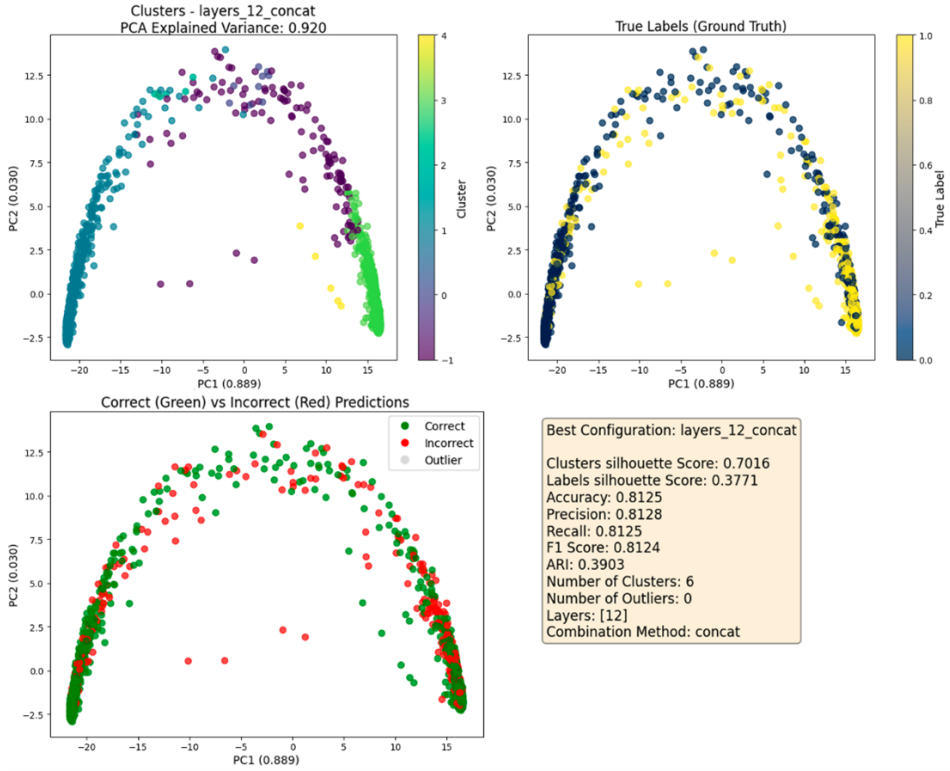

Figure 4: Visualization of clustering results for the fine-tuned model: (a) clusters found by the algorithm; (b) distribution of true labels; (c) error analysis.

exceeds a random level, the fine-tuned model shows a stable result of F1-Score ≈ 0.81. This quantitative leap is also reflected in the visual analysis. In contrast to the completely mixed structure of the classes

in the base model (Figure 3), the visualization for the fine-tuned model (Figure 4b) shows the formation of two distinct, spatially separated groups. The high F1-Score (≈ 0.81) confirms that the structure found by the unsupervised algorithm corresponds with high accuracy to the actual semantic division of requirements. The error analysis (Figure 4c) additionally shows that misclassifications (red dots) are predominantly concentrated at the boundary between the two clusters, indicating that the model makes mistakes only on the most semantically close, "borderline" examples. Thus, it can be concluded that domain adaptation is an effective method for overcoming the "semantic gap," as it allows for the creation of vector representations that form a high-quality, semantically meaningful, and spatially separated structure suitable for subsequent unsupervised clustering.

4.3. Analysis of the informativeness of the fine-tuned model's layers

To understand which layers of the fine-tuned model contribute the most to classification quality, an individual analysis ("profiling") of them was conducted. The experiment consisted of sequential clustering using the [CLS] vector taken separately from each of the 12 encoder layers. The results, presented in Figure 5, demonstrate a clear trend: the quality of clustering (as measured by the F1-Score) steadily increases with the layer level. While the lower and middle layers show low effectiveness, a sharp increase in quality is observed starting from the 9th layer, reaching its maximum at the final, 12th layer (F1-Score ≈ 0.81).

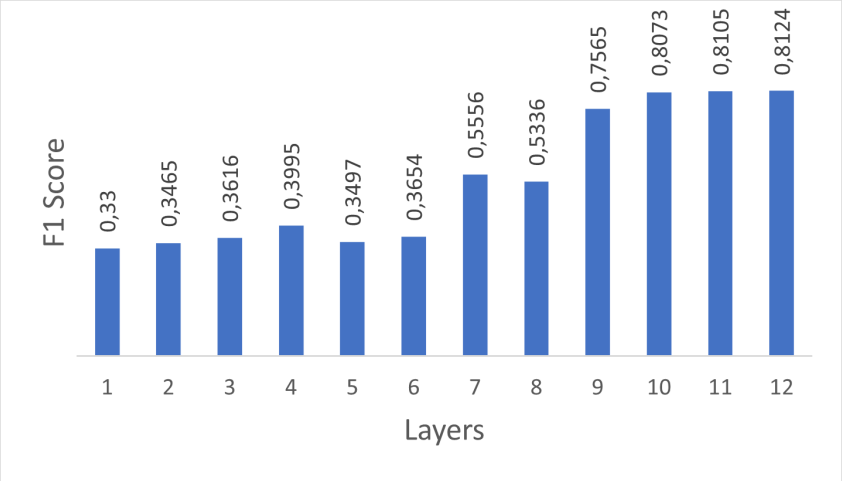

Figure 5: Dependence of clustering quality (F1-Score) on the encoder layer number of the fine-tuned model.

In the next step, based on these data, a hypothesis was tested as to whether a combination of the best layers (9-12) could surpass the result of the best single layer. The experiment showed that the strategy of aggregating the last four layers (by both averaging and concatenation) achieves an F1-Score that is statistically identical to the result of the 12th layer alone. The obtained results allow for two important conclusions. Firstly, the fine-tuning process effectively "specializes" the upper layers of the model (9-12) for the task of distinguishing FR and NFR, which is confirmed by the consistent growth of their informativeness. Secondly, a "plateau" effect is observed. The experimental results show that the final, 12th layer, aggregates all the necessary semantic information for classification so completely that complicating the model by aggregating several layers (even the strongest ones from 9-11) does not lead to an improvement in the F1-Score. Therefore, using the [CLS] vector from only the 12th layer is the most justified and effective approach.

4.4. Identification of ambiguous requirements

In addition to the quantitative quality assessment, a deep analysis of the ability of different fine-tuning approaches to identify groups of semantically ambiguous requirements was conducted. The purpose of

this analysis was to compare how standard classification fine-tuning and the more advanced contrastive learning handle the task of detecting "problematic" requirements. As shown in the previous sections, standard fine-tuning for classification achieves a high F1-Score; however, the error analysis (Figure 4c) shows that the "incorrect" predictions are located in a zone of strong class overlap, and the density-based algorithm HDBSCAN does not identify a significant number of "outliers."

The situation changes radically with the application of contrastive learning. Although the final F1-Score remains at the same high level (≈ 0.80), the structure of the vector space undergoes qualitative changes. Contrastive learning forces the model to form very dense and compact clusters for typical FR and NFR, "pushing out" all intermediate and anomalous cases into the space between them. As a result, as shown in Figure 6, the HDBSCAN algorithm now easily identifies a large and clearly defined group of 129 "outliers," which are located precisely in the "zone of semantic ambiguity" between the two main classes.

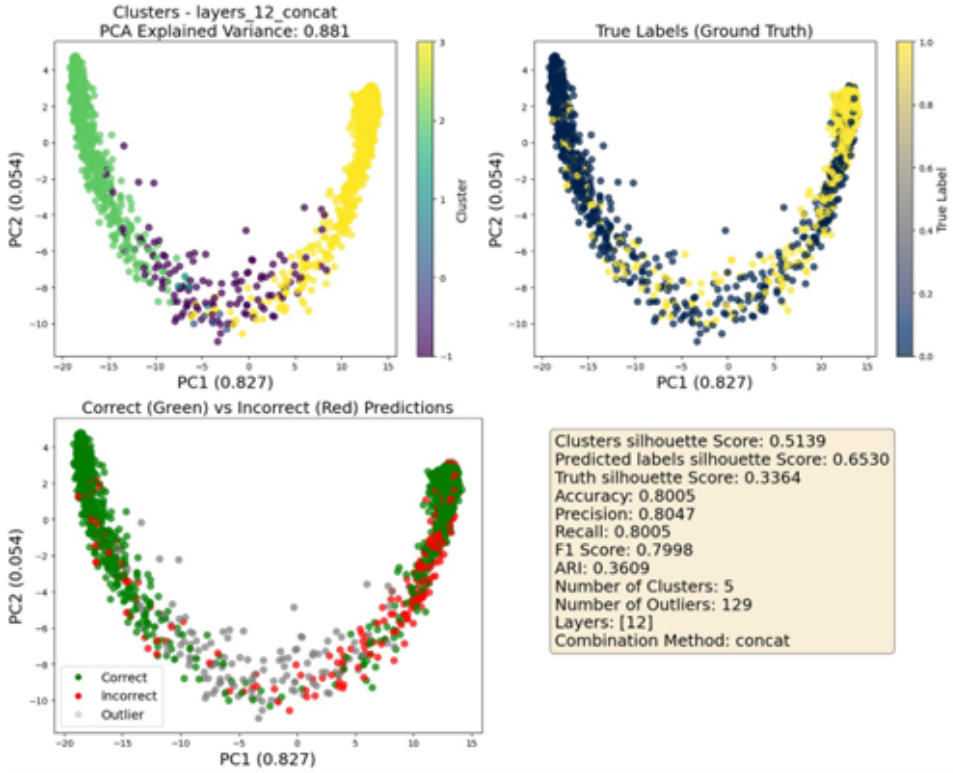

Figure 6: Distribution of classification errors and "outliers" found by the HDBSCAN algorithm after applying contrastive learning.

The comparative analysis shows that although both fine-tuning methods yield similar final accuracy, contrastive learning is a significantly more powerful tool for data quality analysis and the identification of problematic requirements. Unlike standard fine-tuning, which "hides" ambiguous cases within the main classes, contrastive learning explicitly isolates them as outliers. Thus, it can be concluded that for the task of not just simple classification, but of deep analysis and improvement of requirements quality, preference should be given to contrastive learning. It allows for the creation of an automated pipeline for detecting anomalous and ambiguous requirements, which are the first candidates for review and clarification by a business analyst, and can significantly improve the quality of specifications in the early stages of development.

4.5. Comparison with the supervised approach

In the final stage of the research, an "upper bound" of quality that could be achieved on this test corpus with the fine-tuned model was established. For this, the same fine-tuned roberta-base model was used

as a full-fledged supervised classifier with its classification head. As a result of direct classification on the test set (2000 requirements), an F1-Score of 0.82 was achieved. This result serves as a benchmark for the maximum effectiveness of this model and data. Comparing this figure with the best result from unsupervised clustering (F1-Score ≈ 0.81) leads to an extremely important conclusion. The gap between the supervised and unsupervised approaches is minimal, at only $\approx 1\%$. This indicates that domain adaptation (fine-tuning) created such a high-quality and semantically structured vector space that even a simple unsupervised algorithm, like K-Means, can find the boundary between classes within it almost as well as a classifier specifically trained for this purpose. Although the supervised approach remains the "gold standard," a properly adapted language model can serve as a powerful tool for high-quality unsupervised classification of new, unlabeled software requirements, which is a very valuable result for practical application.

5. Conclusion

This work addressed the problem of low efficiency in the unsupervised clustering of software requirements into functional (FR) and non-functional (NFR) types when using general-purpose language models. The research aimed to determine how effectively domain adaptation via fine-tuning could improve the quality of vector representations to solve this task. To achieve this goal, a comparative analysis of two states of the roberta-base model was conducted: a base model and a model fine-tuned on domain-specific data. The results showed that fine-tuning is a critically important step, which increased the clustering quality from an F1-Score of ≈ 0.55 to ≈ 0.81. Further analysis of the fine-tuned model's layers revealed that using the embedding from the final, 12th layer is the optimal strategy, and more complex aggregation methods do not lead to improved results. Furthermore, it was established that the quality of unsupervised clustering on the fine-tuned model (F1 ≈ 0.81) approaches the "upper bound" set by the supervised classifier (F1 ≈ 0.82).

The main contribution of the work is a comparative analysis of two fine-tuning paradigms. It has been shown that while standard classification fine-tuning and contrastive learning yield similar classification accuracy, contrastive learning is a significantly more powerful tool for data quality analysis and the identification of problematic requirements. Unlike the standard approach, which "masks" ambiguous cases, contrastive learning allows for their explicit isolation as "outliers," which is valuable for practical application in requirements engineering. The solution to the problem of identifying requirements using deep learning also correlates with the Sustainable Development Goals (SDGs), specifically Goal 9 (Industry, Innovation and Infrastructure), by increasing the efficiency of creating reliable software solutions.

The limitations of this study include the use of a single language model architecture (roberta-base) and specific datasets for training and testing. Furthermore, the work was conducted only on requirements written in English. The most promising direction for future research is enriching the model with linguistic information, for instance, through Multi-Task Learning by adding an auxiliary task of part-of-speech tagging. Future work may also include testing the proposed approaches on other language model architectures and for other natural languages.

Declaration on Generative AI

During the preparation of this work, the authors employed Generative AI tools to revise the final version of the manuscript. Specifically, Gemini 2.5 Pro and Grammarly were utilized to improve grammar, spelling, and overall readability. After using these tools, the authors reviewed and edited the content as needed and take full responsibility for the publication's content.

References

[1] D. J. Dave, Identifying Functional and Non-functional Software Requirements from User App Reviews and Requirements Artifacts, Master's thesis, Montclair State University, Montclair, NJ, 2022.

[2] H. Xu, G. Xu, Software requirements classification with deep learning: A bibliometric review, Journal of Systems and Software (2024). Preprint.

[3] I. Krak, V. Kuznetsov, S. Kondratiuk, L. Azarova, O. Barmak, P. Radiuk, Analysis of deep learning methods in adaptation to the small data problem solving, in: S. Babichev, V. Lytvynenko (Eds.), Proceedings of the International Scientific Conference "Intellectual Systems of Decision Making and Problem of Computational Intelligence" (ISDMCI 2022), volume 149 of *Lecture Notes on Data Engineering and Communications Technologies*, Springer, Cham, 2023, pp. 333–352. doi:10.1007/978-3-031-16203-9_20.

[4] E. Dias Canedo, B. Cordeiro Mendes, Software requirements classification using machine learning algorithms, IEEE Latin America Transactions 22 (2020) 1057.

[5] F. Baskoro, R. A. Andrahsmara, B. R. P. Darnoto, Y. A. Tofan, A systematic comparison of software requirements classification, Telkomnika 32 (2021) 184.

[6] K. Rahman, A. Ghani, S. Misra, A. U. Rahman, A deep learning framework for non-functional requirement classification, Scientific Reports 14 (2024).

[7] J. Nakirijja, A. Yahya, R. Samikannu, L. L. D. Bulay-og, Enhancing software requirements classification: Integrating recurrent neural networks and natural language processing for managing structural complexity, International Journal of Intelligent Systems and Applications in Engineering 12 (2024) 3110–3121.

[8] Agilitest, How to test non functional requirements?, 2025. URL: https://www.agilitest.com/cards/nfr-to-test.

[9] F. Khayashi, B. Jamasb, R. Akbari, P. Shamsinejadbabaki, Deep learning methods for software requirement classification: A performance study on the pure dataset, in: Proceedings of the International Conference on Software Engineering, Shiraz, Iran, 2022, pp. 1–9.

[10] K. A. Rahman, A. Ghani, R. Ahmad, S. H. Sajjad, Hybrid deep learning approach for nonfunctional software requirements classifications, in: 2023 International Conference on Communication, Computing and Digital Systems (C-CODE), IEEE, 2023, pp. 1–5.

[11] X. Luo, Y. Xue, Z. Xing, J. Sun, Prcbert: Prompt learning for requirement classification using bertbased pretrained language models, in: 37th IEEE/ACM International Conference on Automated Software Engineering (ASE '22), Rochester, MI, USA, 2022, pp. 1–13. URL: https://dl.acm.org/doi/10.1145/3551349.3560417.

[12] F. Yucalar, Developing an advanced software requirements classification model using bert: An empirical evaluation study on newly generated turkish data, Applied Sciences 13 (2023) 11127.

[13] H. Xu, Applying the bert algorithm for software requirements classification, in: Applying the Bert Algorithm for Software Requirements Classification, 2024, pp. 1–24. doi:10.13140/RG.2.2.33317.68320.

[14] S. Tsimenidis, Limitations of deep neural networks: a discussion of g. marcus' critical appraisal of deep learning, arXiv preprint arXiv:2012.15754 (2020).

[15] J. T. Almonte, S. A. Boominathan, N. Nascimento, Automated non-functional requirements generation in software engineering with large language models: A comparative study, in: CASCON 2025, Pennsylvania, USA, 2025, pp. 1–11.

[16] M. Ebrahim, S. Guirguis, C. Basta, Enhancing software requirements engineering with language models and prompting techniques: Insights from the current research and future directions, in: Proceedings of the 63rd Annual Meeting of the Association for Computational Linguistics (Volume 4: Student Research Workshop), Vienna, Austria, 2025, pp. 1–11. URL: https://openreview.net/pdf?id=0cxmpFiTsY.

[17] D. Braun, O. Klymenko, T. Schopf, Y. K. Akan, F. Matthes, The language of engineering: Training a domain-specific word embedding model for engineering, in: Proceedings of the 2021 3rd

International Conference on Management Science and Industrial Engineering, Osaka, Japan, 2021, pp. 8–12. URL: https://dl.acm.org/doi/10.1145/3460824.3460826.

[18] S. Phatak, How to choose the right embedding model?, 2021. URL: https://medium.com/nerd-for-tech/how-to-choose-the-right-embedding-model-487deee9d4d7.

[19] Tiger Data, General-purpose vs. domain-specific embedding models, 2024. URL: https://www.tigerdata.com/blog/general-purpose-vs-domain-specific-embedding-models.

[20] D. Lassner, S. Brandl, A. Baillot, S. Nakajima, Domain-specific word embeddings with structure prediction, Transactions of the Association for Computational Linguistics 11 (2023) 320–335.

[21] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, V. Stoyanov, Roberta: A robustly optimized bert pretraining approach, arXiv preprint arXiv:1907.11692 (2019).

[22] T. Gao, X. Yao, D. Chen, SimCSE: Simple contrastive learning of sentence embeddings, in: Proceedings of the 2021 Conference on Empirical Methods in Natural Language Processing (EMNLP), Online and Punta Cana, Dominican Republic, 2021, pp. 6894–6910. doi:10.18653/v1/2021.emnlp-main.552.

[23] M. Lima, V. Valle, E. a. Costa, F. Lira, B. Gadelha, Software engineering repositories: Expanding the PROMISE database, in: Proceedings of the XXXIII Brazilian Symposium on Software Engineering (SBES '19), Salvador, Brazil, 2019, pp. 427–436. doi:10.1145/3350768.3350776.

[24] S. Sonali, S. Thamada, Fr_nfr_dataset, 2024.

[25] R. J. G. B. Campello, D. Moulavi, J. Sander, Density-based clustering based on hierarchical density estimates, in: Pacific-Asia conference on knowledge discovery and data mining, Berlin, Heidelberg, 2013, pp. 160–172.

[26] P. J. Rousseeuw, Silhouettes: a graphical aid to the interpretation and validation of cluster analysis, Journal of Computational and Applied Mathematics 20 (1987) 53–65.