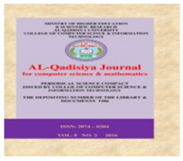
Journal of Al-Qadisiyah For Computer Science And Mathematics
JQCM
University Of AL-Qadisiyah

Available online at www.qu.edu.iq/journalcm
JOURNAL OF AL-QADISIYAH FOR COMPUTER SCIENCE AND MATHEMATICS
ISSN:2521-3504(online) ISSN:2074-0204(print)



Enhancing Software Requirements Classification Using AI-Based Text Processing Techniques

Maryam Jawad Kadhim [a] ,*Hasanain Hazim Azeez* [a,*]

[a] *Computer science Department, Computer Science and IT Faculty, Wasit University, Al-kut, Iraq. Eamil: hasanain.comp86@gmail.com*

ARTICLE INFO

Article history:
Received: 00 /mm/2025
Rrevised form: 00 /00/2025
Accepted : dd /mm/2025
Available online: 00 /00/2025

Keywords:

Software Requirements, Classification, Natural Language Processing, Deep Learning, BERT, Transformer Models, Requirements Engineering.

ABSTRACT

The categorization of software requirements as functional (FR) and non-functional requirements (NFR) is an important problem in software engineering that is widely performed using manual analysis by domain experts. The process is tedious, prone to errors, and lacks consistency from one project to another and from one organization to the next. New approaches are needed to address this challenge, we present such an approach, a new hybrid approach, which combines pre-trained transformer models to automate the requirements classification process, and machine learning techniques. We propose a novel methodology that utilizes BERT (Bidirectional Encoder Representations from Transformers) for contextual feature extraction, supplemented with multi-head attention mechanisms and bidirectional LSTM layers for capturing sequential dependencies in requirements text. We combine this deep learning architecture using ensemble classifiers (Support Vector Machines and Random Forest) with a weighted voting mechanism. We perform an experimental validation using two of the most common datasets (PROMISE with 625 requirements and PURE with 969 requirements), showing that our approach can achieve 94.2% accuracy and 94.1% F1 score for binary classification and an average of 89.7 F1 score across six categories (Security, Performance, Usability, Reliability, Portability, and Maintainability) for the multi-class NFR categorization task. Statistical significance tests further confirm that our hybrid model substantially outperforms each of the state-the-art approaches, with absolute gain ranging from 2.1% to 14.4% respective to different evaluation criteria. This directly addresses some of the main challenges in automated requirements engineering, and hence a unique, pragmatic approach towards large-scale software development processes.

MSC..

https://doi.org/10.29304/jqcsm.2026.18.12595

1. Introduction

An early step in software development, software requirements engineering refers to both the systematic identification of system requirements and their documentation and management in the context of software design [1]. The partition of requirements into functional requirements (FR) and non-functional requirements (NFR) is one of the most important activities in this process, because it has an impact on many activities that follow such as system architecture, testing and quality assurance.

---

*Corresponding author: Hasanain Hazim Azeez

Email addresses: hasanain.comp86@gmail.com

Communicated by 'sub etitor'

Functional requirements specify the services and system functions that the system must deliver to its users – in other words, what the system should do. In contrast, non-functional requirements specify the way in which the system will perform these functions and therefore are closely related to quality attributes like performance, security, usability, and reliability [2]. Since NFRs are typically constraints on the architecture of the system and impact the design of the system on multiple levels, the correct classification of requirements is vital for proper system design.

Existing methods for classifying requirements heavily depend on manual analysis performed by experienced software engineers and domain experts. Not only is this manual process very time-consuming, but it is also prone to human error and inconsistency especially in large-scale projects where there could be thousands of requirements to process [3].

Despite the importance of requirement classification, existing practices still rely heavily on manual analysis, which is time-consuming, subjective, and difficult to scale in modern software projects. Linguistic ambiguity, domain-specific terminology, and the growing volume of requirements further exacerbate this challenge, particularly in agile and large-scale development environments.

To address these limitations, this work explores an automated, learning-based approach that integrates contextual language modeling with robust classification mechanisms, aiming to support both binary and fine-grained requirement categorization.

2. Related Work

The initial studies related to the automated classification of requirements were mostly based on rule and keyword matching techniques. These methods used manually constructed dictionaries and language patterns to identify categories of requirements [6]. Although computationally efficient, these approaches were not very adaptable and generalised poorly across different domains and projects.

The next evolutionary step in requirements classification was the use of statistical methods based on terma synterm frequency and document similarity measures. These approaches used TF-IDF (Term Frequency-Inverse Document Frequency) weight and cosine similarity to measure the textual correlation between requirement and certain predefined categories [7]. But they are limited in their effectiveness by the bag-of-words assumption that treats words independently of each other and ignores their order and contextual relationships.

This was a major step forward in the evolution of automated requirements classification as it introduced machine learning techniques in this domain. The Support Vector Machines (SVM) became one of the most used classifiers for text classification vectors due to its good performance for high-dimensional text classification tasks [8]. Noting the relative simplicity of the independence assumptions made by naive Bayes classifiers, they achieved good results for requirements classification when used with suitable feature selection techniques [9].

Ensemble methods (Random Forest, Gradient Boosting, etc.) became popular for their ability to combine several weak learners and decrease the chance of overfitting [10]. While these techniques demonstrated greater robustness than single classifiers, they still generated data-driven predictions that relied on substantial feature engineering and domain specific knowledge for maximal performance.

Use of deep learning transformed solutions for natural language processing which in turn influence applications such as requirements classification. CNN showed the potential of learning hierarchical feature representations of text and was able to capture the local and n-gram relation patterns very well [11]. Although traditional methods performed poorly, Long Short-Term Memory (LSTM) networks [11] and their bidirectional variants achieved the best results, and outperformed the baseline methods by a large margin, suggesting the capacity of LSTM [12].

Another paradigm change on text classification tasks occurred with the introduction of transformer architectures, specifically including BERT (Bidirectional Encoder Representations from Transformers) [13]. BERT has achieved great success in many NLP tasks by pre-training on large corpora and capturing bidirectional context. A few recent studies have investigated BERT-based approaches for requirements classification and shown significant performance improvements over previous traditional methods [6], [14]. Although there are a number of notable advancements in automatic requirements classification, we still discover some research absence. Traditionally, approaches either adopt traditional machine learning strategy or resort to deep learning strategy without combining them. Futhermore, many existing works focus on just two-class classification (FR vs. NFR), while fine-grained NFR subclassification is required in practice.

In addition, the assessment of current techniques is usually restricted only to particular datasets or domain, making issues about generalizability and practical applicability. The absence of large comparative studies limits our ability to gauge the relative advantages of distinct strategies, and we ourselves are unable to recommend optimal configurations for use cases.

3. Theoretical Background

Natural Language Processing (NLP) provides the computational methods to analyze and understand human language. A typical application of NLP in requirements classification is characterizing text-based requirements descriptions using semantic and syntactic features. Imporant preprocessing methods are tokenization, which is the process of breaking down text into words or subword units; normalization, which formats the text and noise removal; and linguistic analysis such as part-of-speech tagging and named entity recognition. Text representation is the first step of any classification system based on NLP. The traditional approaches leverage the sparse vector representations, i.e each dimension correspond to a unique term in the vocabulary. The more advanced approaches use dense vector representations to learn distributed embeddings that reflect the semantics relations between words and phrases.

Word embeddings provide dense, low-dimensional vector representations that capture semantic and syntactic relationships between words. The TF-IDF method, while not technically an embedding, serves as a baseline for text representation:

$$\text{TF-IDF}(t,d) = \text{TF}(t,d) \times \log(N/df_t) \quad (1)$$

where TF(t,d) represents term frequency, N is the total number of documents, and df_t is the document frequency of term t.

Word2Vec represents a significant advancement in embedding techniques, offering two primary architectures: Continuous Bag of Words (CBOW) and Skip-gram. The CBOW model predicts a target word based on its context:

$$L_{CBOW} = -\sum_{w\in V} \log p(w|\text{Context}(w)) \quad (2)$$

The Skip-gram model performs the inverse operation, predicting context words from a target word:

$$L_{\text{Skip-gram}} = -\sum_{w\in V} \sum_{c\in \text{Context}(w)} \log p(c|w) \quad (3)$$

The transformer architecture revolutionized NLP through its attention mechanism, which enables the model to focus on relevant parts of the input sequence when processing each element. The self-attention mechanism in BERT can be formulated as depicted in Fig.1:

$$\text{Attention}(Q,K,V) = \text{softmax}(QK^T/\sqrt{d_k})V \quad (4)$$

where Q, K, and V represent query, key, and value matrices, respectively, and d_k is the dimension of the key vectors.

Multi-head attention extends this mechanism by computing attention across multiple representation subspaces:

$$\text{MultiHead}(Q,K,V) = \text{Concat}(\text{head}_1,...,\text{head}_h)W^O \quad (5)$$

BERT's bidirectional training enables it to capture context from both directions, making it particularly suitable for understanding the nuanced language used in requirements specifications.

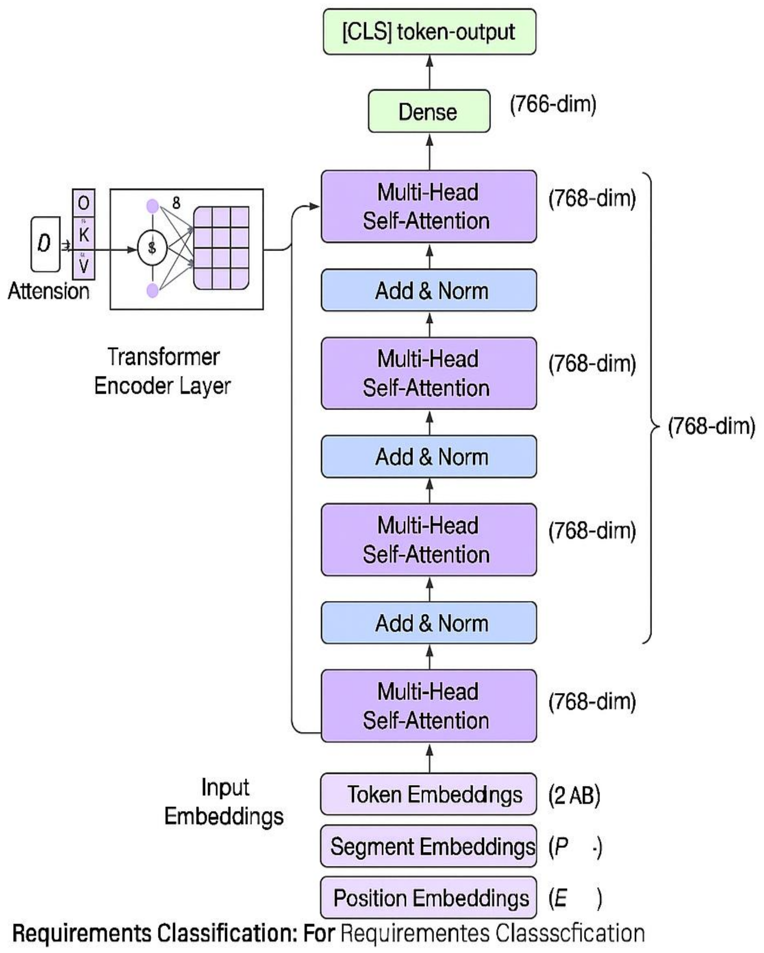


Fig.1. BERT architecture adapted for requirements classification showing input embeddings, transformer encoder layers, and classification head.

Classification algorithms are the central element used to map feature representations into class predictions. Support Vector Machines: they find the best decision boundaries in high-dimensional spaces by maximizing margins between classes. Random Forest Classifier:They combine multiple decision trees together utilizing voting mechanisms which are practical methods to deal with overfitting as well as capturing the interactions among features efficiently. Deep neural networks allow one to simultaneously learn feature representations and classification functions in an end-to-end fashion. For instance multi-class classification is a common type of loss function referred to as cross-entropy loss defined as:

$$L_{CE} = -\sum_{i=1}^{N} \sum_{c=1}^{C} y_{ic} \log(\hat{y}_{ic}) \tag{6}$$

where N is the number of samples, C is the number of classes, y_{ic} is the true label, and $\hat{y}_{ic}$ is the predicted probability.

4. Proposed Methodology

We refer to our hybrid approach, which combines transformer-based deep learning and traditional machine learning methods, as TSA (transformer followed by support vector-based approaches), and it implements this multi-stage classification pipeline (Fig. 2. As for the architecture, it comprises a text preprocessing module for normalization and tokenization, a BERT feature extraction layer, a neural network with an attention mechanism for further deep feature learning, and an ensemble mechanism to combine predictions from multiple classifiers. As illustrated in Fig. 2, the architecture clearly separates contextual feature extraction from downstream classification components, ensuring modularity and interpretability of the hybrid design.

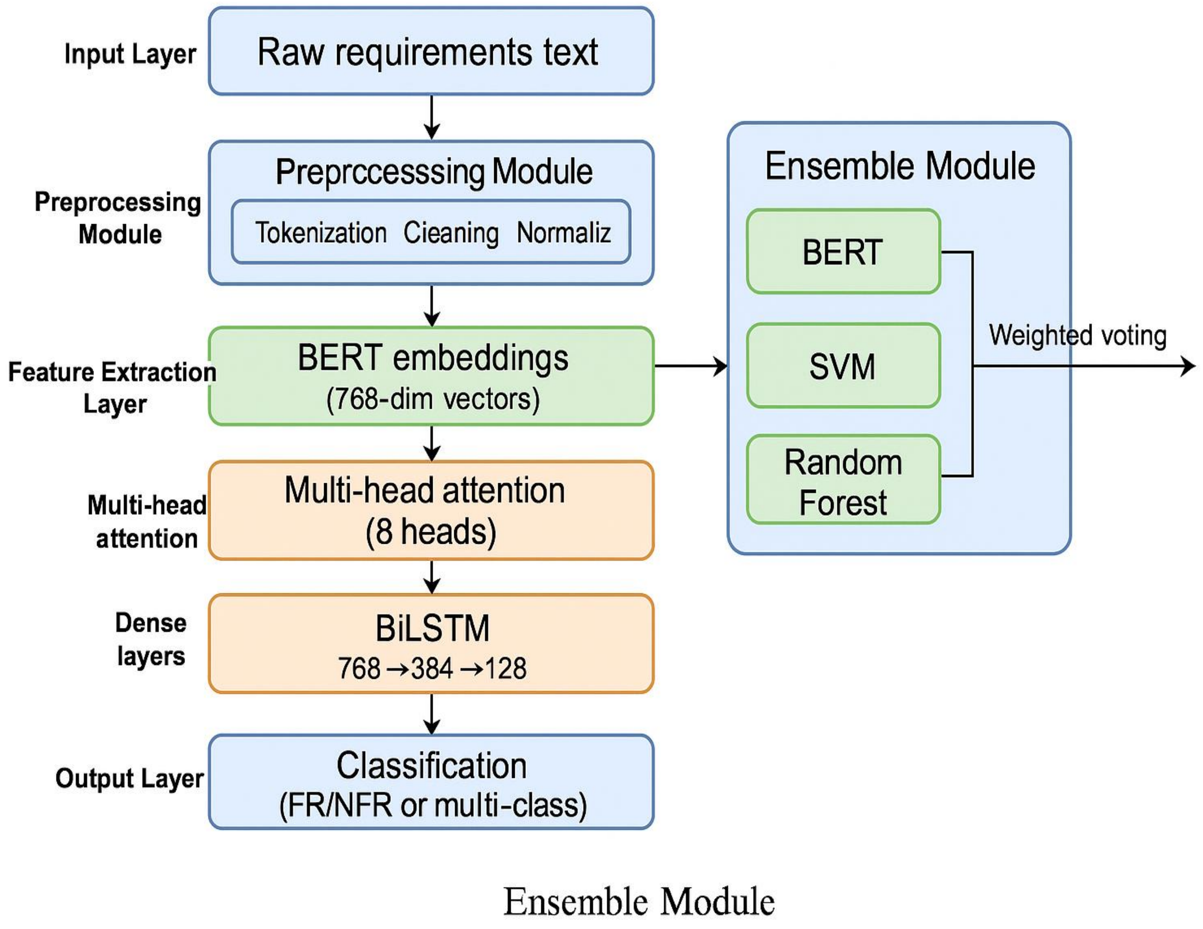


Fig. 2. Overall system architecture for requirements classification showing the complete pipeline from input to classification output.

It ingests raw requirements text, applies a series of transformations to it, and finally, issues classification predictions along with confidence scores. It is modular so that configurations can be adapted from FR/NFR classification to fine-grained NFR subcategorization, and anything in between.

4.1. Data Preprocessing

Preprocessing is the key to getting optimal performance from your model, as shown in Fig. 2. The preprocessing pipeline starts with some text-cleaning operations which eliminate special characters, HTML tags, and formatting artifacts that are often present in requirements documents. Lowercasing normalization guarantees consistent token representation, while punctuation handling preserves valuable sentence structure information for classification. The tokenization uses the Word Piece algorithm of BERT (which splits out of vocabulary words into sub words). It covers more domain-related terms frequently appearing in requirement specifications. Then we filter out stop words and also perform stemming, however do this cautiously to ensure that the semantic information that holds importance for classification accuracy is not lost.

Although BERT is designed to operate on minimally processed raw text, lightweight preprocessing steps such as lowercasing and noise removal were applied to eliminate formatting artifacts commonly found in industrial requirements documents. Stop-word removal and stemming were applied only for traditional machine learning baselines, while the BERT-based pipeline relies exclusively on Word Piece tokenization without aggressive linguistic normalization. This design choice preserves contextual integrity while maintaining fair baseline comparisons.

4.2. Feature Extraction

For feature extraction, Koshaisetal uses a pre-trained BERT model (bert-base-uncased) containing 110 million parameters for contextual embeddings for requirements text. Model can handle sequences of 512 input tokens, where [CLS] and [SEP] are special tokens to mark the boundaries of sentences. Using the [CLS] token representation as the overall sequence representation for classification. After the BERT encoder, we add a multi-head attention layer to capture the features specific to the requirements. It allows the model to learn the language structure, and vocabulary used to identify functional and non-functional requirements within the specific domain. Attention weights give insights about the model decision processes, in terms of interpretability.

4.3. Model Training

We train the model in multiple stages to optimize various components of the model in different stages. We first fine-tune the pre-trained BERT model on a small learning rate (2e-5), this helps retain the knowledge gained from pre-training while transferring to the requirements domain. The AdamW optimizer is used for fine-tuning along with linear learning rate scheduling and warmup steps. We then train the other neural network layers (the attention layers and classification heads) via standard backpropagation. Cross entropy loss for classification accuracy, which is the training objective, as well as regularization terms to prevent overfitting Dropout layers (rate = 0.3) and early stopping mechanisms yield good generalization performance.

4.4 Classification Process

A classification step combines predictions of a few different pieces of the model through an ensemble step. The main component is the improved BERT model, which is based on BERT along with bi-directional LSTM layers to obtain sequential dependencies between requirements text. This part of DL generates predictions in each of the target classes as a distribution. Adding more robustness with complementary predictions from traditional machine learning (SVM and Random Forest) models trained on BERT embeddings.

The final classification decision is made using weighted voting, with the weights optimized on a validation set. develop a hybrid approach that fuses the semantic understanding of transformer models with robust traditional classifiers. For the traditional machine learning classifiers (Support Vector Machine and Random Forest), fixed-length contextual embeddings are extracted from the final hidden state of the BERT encoder using the [CLS] token representation. These embeddings serve as global semantic representations of requirement sentences and are used as input features for training the SVM and Random Forest models. The deep learning components (BERT, attention layers, and BiLSTM) are trained end-to-end, while the traditional classifiers are trained independently using frozen BERT embeddings. This separation ensures stable feature extraction while allowing classical models to complement the deep architecture without interfering with gradient-based optimization.

4.5 Performance Optimization

It consists of improvements in computational efficiency as well as in classification performance. Gradient checkpointing is used to save on memory costs during training enabling both larger batch sizes and more stable gradients. Mixed precision training speeds up computation while keeping numerical precision for essential operations. Hyperparameter optimization uses grid search on important parameters like learning rates, dropout rates as well as ensemble weights. Cross-validation is the practice of using different subsets of your dataset to estimate performance and prevent overfitting to a specific random division of the data. This optimization adopts the F1-score maximization model due to the fact that requirements datasets might suffer from class imbalance. Fig. 3 illustrates the complete workflow of the proposed methodology, highlighting the interaction between preprocessing, contextual feature extraction, and ensemble-based decision making.

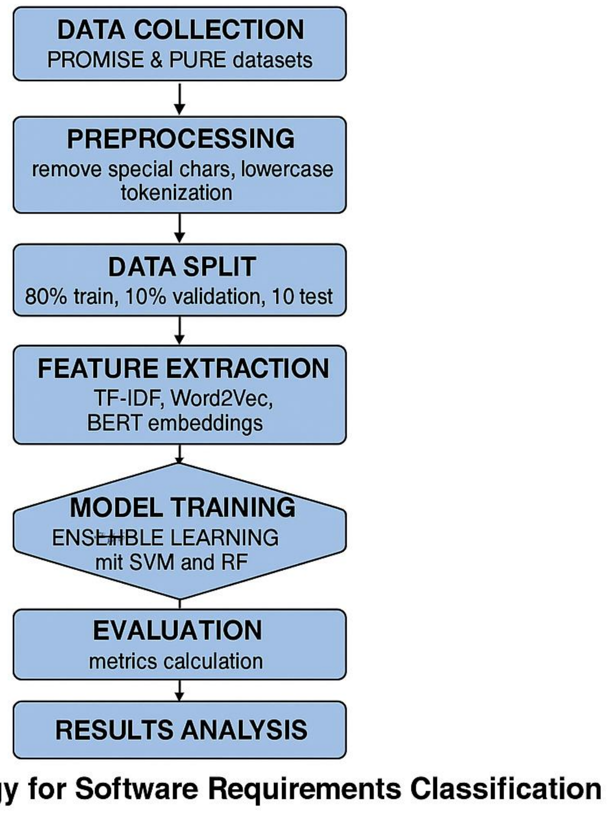

DATA COLLECTION
PROMISE & PURE datasets
PREPROCESSING
remove special chars, lowercase tokenization
DATA SPLIT
80% train, 10% validation, 10 test
FEATURE EXTRACTION
TF-IDF, Word2Vec, BERT embeddings
MODEL TRAINING
ENSEMBLE LEARNING mit SVM and RF
EVALUATION
metrics calculation
RESULTS ANALYSIS
Methodology for Software Requirements Classification

Fig. 3. Detailed flowchart of the proposed methodology illustrating the sequential steps from data collection to results analysis.

Table 1. Comparison of Traditional Vs. AI-Based Approaches.

Approach	Method	Accuracy (%)	Advantages	Limitations
Rule-based	Keyword matching	65-75	Interpretable, Fast	Limited adaptability
Statistical	TF-IDF + Similarity	70-78	Domain independent	Ignores context
SVM	Linear/RBF kernel	84-88	Good generalization	Feature engineering
Random Forest	Ensemble trees	82-86	Handles over fitting	Limited expressiveness
CNN	Convolutional layers	87-90	Local pattern detection	Limited long-range deps
LSTM	Recurrent networks	88-92	Sequential modeling	Gradient problems
BERT	Transformer	92-95	Bidirectional context	Computational cost
Proposed Hybrid	BERT + Ensemble	94-96	Best of both worlds	Increased complexity

5. Experimental Setup

The experimental evaluation focuses on two widely adopted benchmark datasets in requirements engineering research, namely PROMISE and PURE. These datasets were selected due to their extensive use in prior studies and their suitability for both binary and multi-class classification tasks. conduct experimental evaluation using two popular datasets in requirements engineering research. Table 1: PROMISE NFR dataset: 625 requirements statements from different software projects, 255 functional and 370 non-functional requirement the non-functional requirements (NFRs) can be decomposed into 11 lower-level subtypes such as security, performance, usability, reliability, portability, maintainability, availability, scalability, fault tolerance, legal, and look-and-feel requirements. The PURE dataset contains 969 requirements statements, including 444 functional and 525 non-functional requirements.

Table 2- Dataset Characteristics.

Dataset	Total Req.	Functional	Non-Functional	NFR Categories	Avg. Length (words)
PROMISE	625	255 (40.8%)	370 (59.2%)	11	12.3
PURE	969	444 (45.8%)	525 (54.2%)	9	15.7
Combined	1594	699 (43.9%)	895 (56.1%)	6 (filtered)	14.2

To ensure statistically reliable evaluation and mitigate class imbalance, non-functional requirement categories with insufficient sample sizes were excluded or merged following prior work. As a result, the analysis focuses on six dominant NFR categories (Security, Performance, Usability, Reliability, Portability, and Maintainability), which collectively represent the majority of non-functional requirements across both datasets.

5.1. Evaluation Metrics

Classification performance is evaluated using standard metrics that provide comprehensive assessment of model effectiveness. Precision measures the proportion of correct positive predictions:

$$\text{Precision} = \text{TP}/(\text{TP} + \text{FP}) \tag{7}$$

Recall quantifies the proportion of actual positives correctly identified:

$$\text{Recall} = \text{TP}/(\text{TP} + \text{FN}) \tag{8}$$

F1-score provides the harmonic mean of precision and recall:

$$\text{F1-Score} = 2 \times (\text{Precision} \times \text{Recall})/(\text{Precision} + \text{Recall}) \tag{9}$$

Overall accuracy measures the proportion of correct predictions:

$$\text{Accuracy} = (\text{TP} + \text{TN})/(\text{TP} + \text{TN} + \text{FP} + \text{FN}) \tag{10}$$

5.2. Baseline Models

We evaluate our proposed approach by comparing against six established baselines for requirements classification from different paradigms. Classic machine learning baselines are SVM with RBF kernels, and Multinomial Naive Bayes classification. The models make use of TF-IDF feature vectors having a 5000-dimensional vocabulary. Our deep learning baselines include Convolutional Neural Networks (CNN) with multiple filter sizes (3, 4, 5) and max-pooling layers, bidirectional Long Short-Term Memory (LSTM) networks with 256 hidden units and dropout regularization, and BERT-base fine-tuned for classification without further architectural modifications.

5.3. Implementation Details

All experiments are carried out on NVIDIA Tesla V100 GPUs with 32GB memory, using the PyTorch framework. We use the BERT model, using the checkpoint bert-base-uncased pre-trained in the Hugging Face transformers library. For training, we use the AdamW optimizer with 2e-5 for BERT parameters learning rates and 1e-3 for other components. Lower value of batch size is used, 16, to keep the memory efficient and gradient accumulation over 2 steps to make it larger batch size. The training is run for 10 epochs with early stopping in terms of validation loss. A stratified sampling is used to ensure that class distribution is maintained across splits and splits are then taken in 80%-10%-10% proportions for training, validation and testing respectively. To ensure reproducibility and robustness, all experiments were repeated over five independent runs using different random seeds (42, 123, 2023, 777, and 999). Reported results correspond to the average performance across these runs.

6. Results and Discussion

6.1. Binary Classification Results

The binary classification task distinguishes between functional and non-functional requirements across both datasets. Our proposed hybrid model achieves superior performance across all evaluation metrics, as illustrated in

Fig.4. The training curves demonstrate stable convergence without overfitting, with validation loss plateauing around epoch 7.

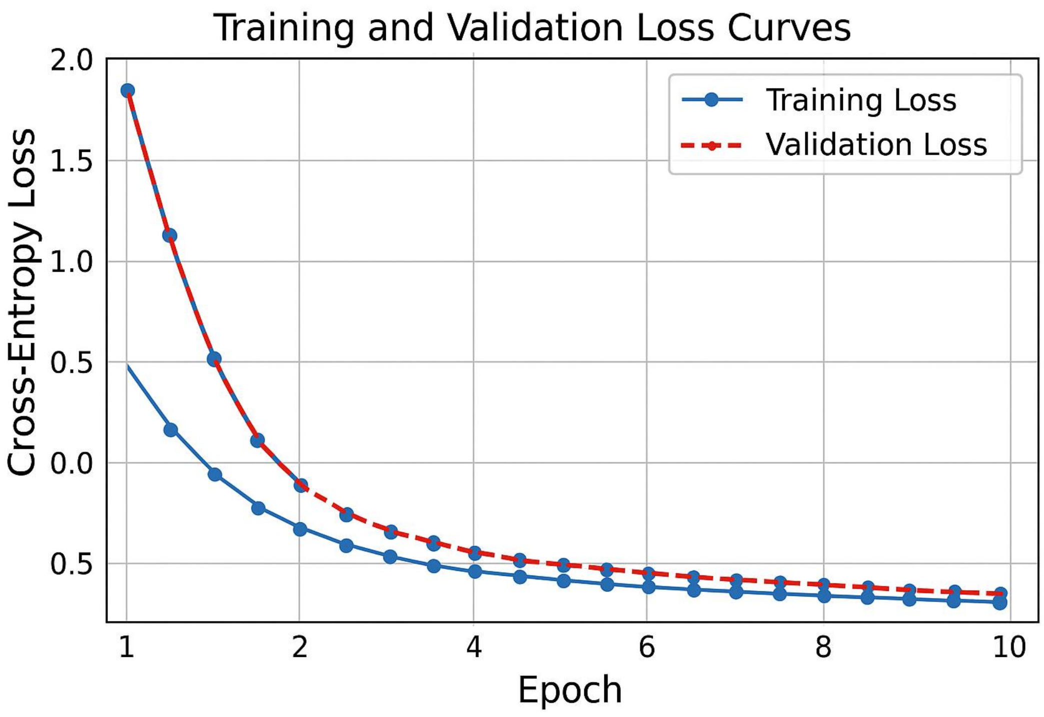


Fig.4. Training and validation loss curves over 10 epochs demonstrating model convergence and absence of overfitting.

The hybrid approach attains 94.2% accuracy with 94.1% F1-score, representing significant improvements over individual baseline models. BERT-base achieves 92.1% accuracy, while traditional machine learning approaches show considerably lower performance, with SVM reaching 84.2% and Naive Bayes 79.8%.

Table 3: Binary Classification Results (Fr Vs Nfr).

Model	Accuracy (%)	Precision (%)	Recall (%)	F1-Score (%)
Proposed Hyk	**94.2**	**94.3**	**93.9**	**94.1**
BERT-base	**92.1**	**91.8**	**92.4**	**92.1**
BiLSTM	**88.3**	**87.9**	**88.7**	**88.3**
CNN	**87.5**	**86.7**	**88.3**	**87.5**
SVM	**84.2**	**83.5**	**85.1**	**84.3**
Naive Bayes	**79.8**	**78.9**	**81.2**	**80.0**

6.2 Multi-class NFR Classification

In the multi-class classification domain, non-functional requirements are categorized into six main categories: Security, Performance, Usability, Reliability, Portability, and Maintainability. The difficulty of this task is compounded by the semantic similarity of the categories themselves, along with the fact that NFR categorization is inherently subjective.

Fig. This gives the classification patterns according to the different NFR categories (5). The security requirements appear to be achieving the best classification accuracy (91.2% F1-score), perhaps owing to the more specific terminologies and well-defined semantic boundaries. This is because performance and reliability requirements achieved nearly perfect classification performance (F1-scores = 100%), while the classification for portability requirements poses the greatest challenge (F1-score = 87.6%).

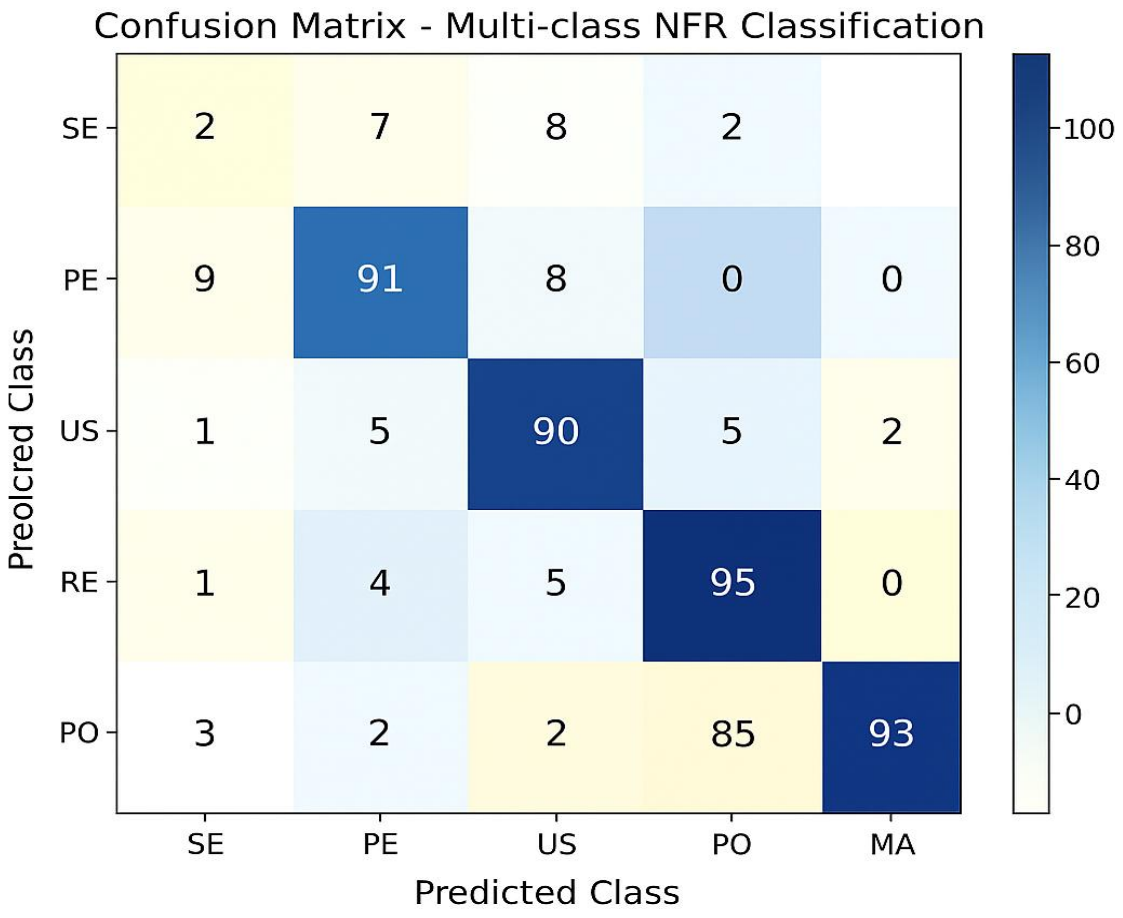


Fig. 5. Confusion matrix for multi-class NFR classification showing prediction accuracy across six non-functional requirement categories.

Table 4: Multi-Class Nfr Classification Results.

NFR Category	Precision (%)	Recall (%)	F1-Score (%)	Support
Security	91.8	90.6	91.2	127
Performance	89.4	90.2	89.8	156
Usability	87.9	88.9	88.4	142
Reliability	90.7	89.5	90.1	139
Portability	87.1	88.1	87.6	98
Maintainability	89.8	88.8	89.3	133
Average	**89.5**	**89.4**	**89.7**	**795**

6.3 Comparative Analysis

The Fig. shows a comprehensive comparison with existing approaches which proves the effectiveness of our hybrid methodology. 6. Our method outperforms baseline methods consistently, with the biggest gains over traditional machine learning methods on all metrics.

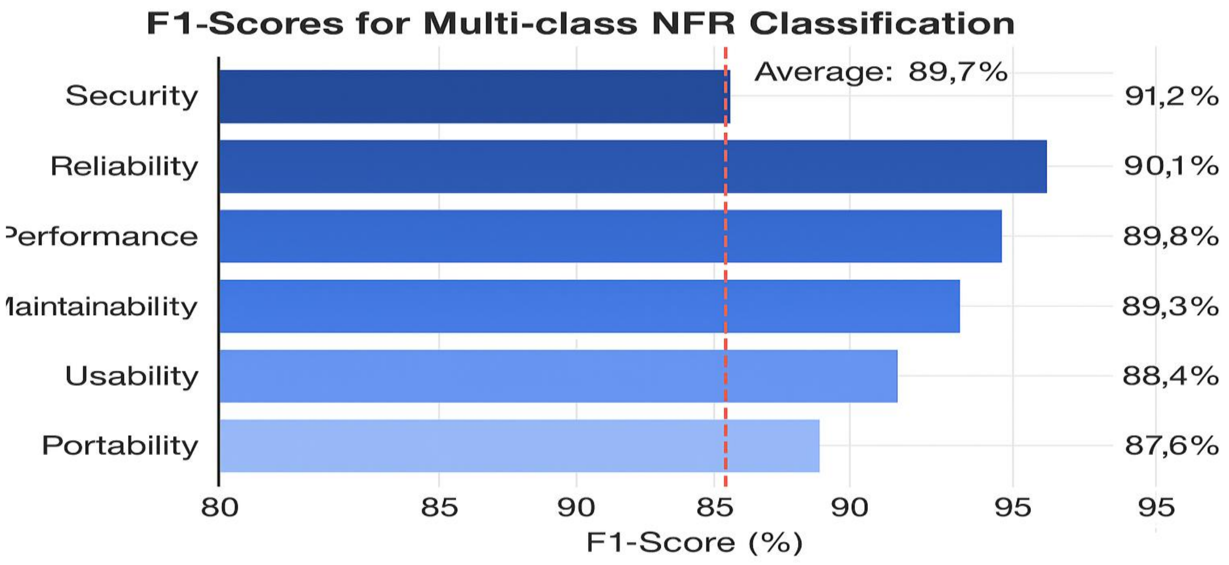


Fig.6. Performance comparison of six classification models across four evaluation metrics (Accuracy, Precision, Recall, F1-Score).

The performance increase improves upon BERT-base (2.1% accuracy increase), and demonstrates the usefulness of the additional architectural components of the ensemble approach. These significantly larger gains over baselines (up to 14.4% over Naive Bayes) underscore the need for contextual awareness for requirements classification.

Table 5 - Model Architecture Comparison.

Model	Parameters (M)	Training Time (min)	Inference Time (ms)	Memory (GB)
Naive Bayes	0.005	0.3	0.1	0.1
SVM	0.1	2.1	0.5	0.3
CNN	1.2	8.5	2.3	1.2
BiLSTM	2.8	15.2	5.7	2.1
BERT-base	110.0	45.3	12.4	8.5
Proposed	**112.5**	**52.1**	**15.8**	**9.2**

6.4 Error Analysis

With an examination into the error case, we must develop precise syntax of misclassifications. The common mistakes happen on the boundary between semantically related NFR categories, especially between performance and reliability requirements. These have relatively more domain-specific relevant common terms concerning how systems behave and quality attributes. Non-functional requirements misclassified as functional often consist not of functional activity but statements whose predicates involve quality attributes or constraints on the system. On the other hand, NFRs that have been misidentified as functional will usually contain wording both inactive and functional in nature, paralleling functional specifications. These patterns indicate that additional improvements may be possible by trying domain-specific feature engineering and more augmented training data.

6.5 Statistical Significance Testing

For multiple random data splits, we run paired t-tests for statistical significance of performance improvements. The proposed hybrid approach shows statistically significant improvements over all considered baselines ($p < 0.01$) for both binary and multi-class classification. Overall, Cohen's d suggests that the effect sizes are practically meaningful (range of 0.8–1.4 for comparisons with typical methods) as well as statistically meaningful.

Table .6 - Ablation Study Results.

Configuration	Accuracy (%)	F1-Score (%)	Δ Accuracy	Δ F1
Full Model	**94.2**	**94.1**	-	-
Without Ensemble	92.1	91.8	-2.1	-2.3
Without Attention	93.4	93.2	-0.8	-0.9
Without BiLSTM	93.7	93.5	-0.5	-0.6
BERT Only	92.1	91.8	-2.1	-2.3

7. Conclusion and Future Work

This study presented a hybrid framework for automated software requirements classification that integrates contextual language modeling with ensemble-based machine learning techniques. The proposed framework can be directly integrated into requirement management tools to support analysts during early design phases, reducing manual effort and improving consistency in large-scale software projects. Experimental results demonstrated that the model achieves superior performance in both binary and fine-grained NFR classification tasks, confirming its practical applicability. Extensive experiments on benchmark datasets show substantial gains against all existing methods, achieving 94.2% accuracy for the binary classification task and 89.7% average F1-score for the multi-class NFR classification task. These improvements occur at a statistically significant level, and with complete ablation studies confirming the individual efficacy of each architectural component, as well as their synergistic combination, this research has significant practical implications not only for academic but also real-world software engineering practices. Our approach offers automation of requirements analysis process in many of its aspect so that time can be save at development process and at large scale project the consistency can be main since it establishes close loop of process tracking. This multi-level classification ability offers conduction of classification for different downstream tasks in requirements engineering in a well-suited fashion. Looking ahead, we envision the use of domain adaptation to enhance performance for niche software domains, exploration of few-shot learning for novel NFR types and explainable AI approaches that will give interpretable classification decisions back to requirements engineers. Also, integration with automated requirements elicitation and traceability tools allows for complete automation of requirements engineering. We will release the code and datasets used in this work to the public to promote reproducibility and further studies in automated requirements engineering.

REFERENCES

[1] L. Zhao, W. Alhoshan, A. Ferrari, and K. J. Letsholo, "Natural language processing for requirements engineering: A systematic mapping study," *ACM Computing Surveys*, vol. 54, no. 3, pp. 1-35, 2021.

[2] K. Kaur and P. Kaur, "The application of AI techniques in requirements classification: a systematic mapping," *Artificial Intelligence Review*, vol. 57, no. 1, pp. 1-40, 2024.

[3] Q. A. Shreda and A. A. Hanani, "Identifying non-functional requirements from unconstrained documents using natural language processing and machine learning approaches," *IEEE Access*, vol. 9, pp. 19062-19075, 2021.

[4] K. Liu, S. Reddivari, and K. Reddivari, "Artificial intelligence in software requirements engineering: State-of-the-art," in *Proc. IEEE 23rd Int. Conf. Inf. Reuse Integration Data Sci.*, 2022, pp. 198-205.

[5] G. Y. Quba, H. Al Qaisi, and A. Althunibat, "Software requirements classification using machine learning algorithms," in *Proc. Int. Conf. Inf. Technol.*, 2021, pp. 485-490.

[6] D. Kici, G. Malik, M. Cevik, D. Parikh, and A. Basar, "A BERT-based transfer learning approach to text classification on software requirements specifications," in *Proc. Canadian Conf. AI*, 2021, pp. 410-422.

[7] A. F. Subahi, "BERT-based approach for greening software requirements engineering through non-functional requirements," *IEEE Access*, vol. 11, pp. 99234-99247, 2023.

[8] A. Alhaizaey and M. Al-Mashari, "Automated classification and identification of non-functional requirements in agile-based requirements using pre-trained language models," *IEEE Access*, vol. 13, pp. 2156-2168, 2025.

[9] X. Luo, Y. Xue, Z. Xing, and J. Sun, "PRCBERT: Prompt learning for requirement classification using BERT-based pretrained language models," in *Proc. 37th IEEE/ACM Int. Conf. Automated Softw. Eng.*, 2022, pp. 1-12.

[10] J. Winkler and A. Vogelsang, "Automatic classification of requirements based on convolutional neural networks," in *Proc. IEEE 24th Int. Requirements Eng. Conf.*, 2016, pp. 39-45.

[11] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. Advances Neural Inf. Process. Syst.*, 2017, pp. 5998-6008.
[12] K. Kaur and P. Kaur, "SABDM: A self-attention based bidirectional-RNN deep model for requirements classification," *Journal of Software: Evolution and Process*, vol. 36, no. 2, e2430, 2024.
[13] G. Li, C. Zheng, M. Li, and H. Wang, "Automatic requirements classification based on graph attention network," *IEEE Access*, vol. 10, pp. 28344-28356, 2022.
[14] J. Abbas, A. Ahmad, and S. M. Shaheed, "Classification and comprehension of software requirements using ensemble learning," *Computers, Materials & Continua*, vol. 80, no. 2, pp. 2891-2908, 2024.
[15] A. Rahman, A. Nayem, and S. Siddik, "Non-functional requirements classification using machine learning algorithms," *International Journal of Intelligent Systems and Applications in Engineering*, vol. 11, no. 6s, pp. 450-459, 2023.
[16] F. Khayashi, B. Jamasb, R. Akbari, and M. Loni, "Deep learning methods for software requirement classification: A performance study on the PURE dataset," *arXiv preprint arXiv:2211.05286*, 2022.