Bi-LSTM-Based Model for Classifying Software Requirements

Jalil Abbas *, Zhuoxuan Hu, Saima Kanwal, Arshad Ahmad *, Ahmad Almogren, Ayman Altameem

Posted Date: 28 October 2024

doi: 10.20944/preprints202410.2129.v1

Keywords:

Bi-LSTM; software engineering; functional requirements; non-functional requirements; machine learning

Article

Bi-LSTM-Based Model for Classifying Software Requirements

Jalil Abbas [1,*], Zhuoxuan Hu [1], Saima Kanwal [2], Arshad Ahmad [3,*], Ahmad Almogren [4] and Ayman Altameem [5]

[1] School of Computer Science and Technology, Anhui University, Hefei 230039, China
[2] School of Computer and Communication, Lanzhou University of Technology, Lanzhou, 730050, China
[3] School of Computing Sciences, Pak Austria Fachhochschule: Institute of Applied Sciences and Technology, Haripur,22620 Pakistan
[4] Department of Computer Science, College of Computer and Information Sciences, King Saud University, Riyadh 11633, Saudi Arabia
[5] Department of Natural and Engineering Sciences, College of Applied Studies and Community Services, King Saud University, Riyadh, 11543, Saudi Arabia
* Correspondence: jalil02085@stu.ahu.edu.cn (J.A.); yaarshad@gmail.com (A.A.)

Abstract: In the domain of software engineering, the accurate and effective classification of requirements is of paramount importance. Proper classifications of these requirements enable developers to create robust and error-free software solution. Traditional methods of user requirements classification face the issue of the reliance on manual processes, which are time-consuming, labor-intensive, and prone to human error. The limitations of traditional methods underscore the need for more automated, scalable, and robust approaches to user requirements classification in order to meet the demands of modern software development practices. To improve the classification process, we employed a Deep Learning (DL) methodology termed the Bidirectional Long Short-Term Memory (Bi-LSTM) model to conduct feature extraction, after which we merged these feature vectors into ML classifiers. Our research methodology is structured around a five-step process. Initially, the textual input is tokenized and converted to lowercase. Subsequently, we eliminate all punctuation. The pre-processed text is then subjected to a Bi-LSTM (Bidirectional Long Short-Term Memory) model for feature vector extraction. After that, this feature vector is fed into different classifiers such as Medium KNN, Cubic KNN, Linear SVM, Quadratic SVM, and Cubic SVM and obtained an accuracy of around 99.60% to 99.80% on a publicly available dataset of requirements.

Keywords: Bi-LSTM; software engineering; functional requirements; non-functional requirements; machine learning

1. Introduction

The classification of the text refers to the categorization based on the attributes and properties belonging to each text. Text classification is utilized in various domains such as the identification of spam [1] and the categorization of news [2]. It is possible to do manual classification if the number of documents is limited. The task becomes difficult if these documents are in hundreds or thousands [3]. Manual requirement classification becomes cumbersome with increasing data volumes, necessitating efficient automated solutions. By Machine Learning (ML) and Deep Learning (DL) techniques aims to accelerate the accurate classification of Functional (FRs) and Non-functional Requirements (NFRs), ultimately contributing to improve software reliability and meeting project deadlines. Through automated classification, the goal is to empower software engineers to create efficient, error-free, and high-quality software within limited timeframes. The same is the case with software development requirement classification as the development of high-quality software is considered a costly and time taking task as it addresses the real-world problem [4]. There involve some software engineering tasks such as requirement identification, analyzing, designing, and final implementation. The requirements of the software, which are considered major properties of the

 doi:10.20944/preprints202410.2129.v1

software, are also constrained by various factors. There are some other factors such as the tool of development, techniques, competency of a developer, and timeline that also add an additional layer of complexity. Due to these complexities, there come some defects in the software development life cycle (SDLC) [5]. One of the key challenges in requirements classification is accurately identifying relevant stakeholders during the elicitation phase, as misidentification can lead to incomplete or incorrect requirements, ultimately causing project failure. A systematic literature review highlights 17 methodologies for stakeholder identification, emphasizing its importance in ensuring high-quality requirements classification [6]. According to authors of study [7], there is a fact that requirement engineers and software users use various terminologies and structures of sentences for a description of same kind of requirements. The high orders of inconsistency in the elicitation of requirements make the classification of requirements prone to error. There should be some optimal way to find automated classification. Inadequate collection, misunderstanding of requirements, inefficient architecture, and bad practices of coding may also create problems [8]. It will be very helpful in making good quality software as well in a limited time if these tasks are automized [9-12]. Identification of software requirements is considered a key task in the development of software. The Software Requirement Classification task is based on the classification of requirements of software [13]. The classification of the requirements can be considered as two categories as FRs and NFRs. Generally, the provision of the service that is considered mandatory is called FRs such as the function or behavior of a system and the quality of providing these services is referred as NFRs. The NFRs are based on performance, security, usability, and reliability.

The user of software delivers their requirements in natural language and typically in words. After gathering of these requirements, this is the responsibility of analyst to extract the software requirements from the unstructured and ambiguous data to make the designer understand. The manual extraction of each requirement from ambiguous language is considered as daunting task. This difficulty comes due to ineffective extraction of features by human being. It is obvious that, if the FRs and NFRs are not extracted correctly, it will result in the failure of the project or software. Therefore, it is important to extract the complete and accurate FRs and NFRs in limited time at initial stage of SDLC. This is the reason that manual extraction of requirement is considered inaccurate and ineffective. The accurate and faster classification of requirement can be performed with the help of Artificial Intelligence (AI) techniques. Currently, ML and DL have transformed personal computers in hard working and intelligent assistants. These assistants are helping people in different domains such as industry, medicine and in software engineering as well. Hence, it can be easily stated that these methods can help software engineers to successfully classify the software requirements.

The primary contribution of this paper is the introduction and utilization of the Bi-LSTM model [14], an advanced type of Recurrent Neural Network (RNN). Traditional RNNs are adept at recognizing patterns in sequences of data but struggle with learning from data points that are far apart due to issues like vanishing and exploding gradients. The Bi-LSTM model addresses these limitations by incorporating both bidirectional processing and long-term memory capabilities, making it particularly effective for sequence-based classification tasks [15]. This approach enables the network to remember information over extended periods and capture dependencies within the data more effectively.

This paper develops the Bi-LSTM Model specifically for classification tasks, where the principal objective is to determine the specific category or class to which a particular input belongs. Conventional machine learning (ML) techniques often face difficulties in handling such tasks due to their limited ability to manage temporal dependencies and contextual information within sequences. The Bi-LSTM model overcomes these hurdles by processing data in both forward and backward directions, thus utilizing the entire context of the sequence. This bidirectional approach, combined with the memory retention properties of LSTM networks, ensures a more comprehensive understanding of the sequence, leading to improved classification accuracy.

Furthermore, the paper elaborates on the dual processing capability of Bi-LSTMs, which consents them to utilize together prior and impending contexts, providing a richer and more nuanced analysis of the sequence. This comprehensive context utilization is critical for accurately capturing

 doi:10.20944/preprints202410.2129.v1

the intricacies of sequence-based data, which is often missed by conventional ML models. By addressing these challenges, the Bi-LSTM model enhances the performance and reliability of classification tasks, making it a robust solution for applications requiring precise and thorough sequence analysis.

The rest of the paper is organized as follows: **Section 2** reviews existing approaches and highlights the unique contributions of this research. **Section 3**,details the proposed framework, explaining the techniques and processes used. **Section 4.** presents the experimental results and discussion. **Section 5** discusses the threats to validity and how they were addressed. **Section 6** concludes the paper with a summary of key findings and suggestions for future work.

2. Literature Review

Timely and correct identification of requirements is very important for high quality software development. For this purpose, many researchers have carried out a plenty of work. In recent years, ML techniques have been used in Requirement Engineering (RE) activities, offering new approaches. A systematic mapping by [16] identified 57 algorithms applied in eight key RE activities, including requirements analysis and failure prediction. The authors of study [17], employed a method for classification of NFRs to help the experts in development of a good quality software. The method is based on a combination of ML attributes computation and classification methods. The approach is comprised of seven ML based techniques and four selection methods for features. The aim was to automatically classify NFRs by finding suitable pairs. The scheme was assessed for precision, F1-score, recall, accuracy and obtained results of 66, 61, 61, and 76 percent accordingly. In study [3], authors compared some of the ML based algorithms for feature extraction to classify the NFRs and FRs. Initially, the preprocessing is performed by cleaning the document. After cleaning, two methods for computation and one for selection of features is used such as Term Frequency–Inverse Document Frequency (TF-IDF), Bag of Words (BoW), and Chi Squared (CHI2). A new dataset called PROMISE_exp is used to test the scheme and obtained F1-score of 74 percent for general classification.

To classify software requirements, a new method is presented by authors [18]. Initially, the dataset called PURE is manually annotated to make a new dataset based on FRs and NFRs. The fine tuning of the model called BERT is performed and results are compared with ELMo and fastText. The assessment of the system is performed by comparing the results of PURE and document of Request for Information (RFI). The system succeeded in obtaining a maximum F1-score of 86% on PURE dataset and 80% on RFI dataset. In another method [19], DL based technique is presented by authors for classification of requirement. The task of classification is performed by exploiting five DL methods with the help of two voting algorithm of classification. The experiments were conducted by creating data from PURE dataset having 2617 FRs and 2044 NFRs and achieved promising results. Study [20] presented a scheme based on 5 ML methods and classified FRs into different 6 classes such as empowerment, solution, feature limitation, action limitation, policy, and definition. The method was tested by exploiting dataset having 600 FRs. The ML algorithms along with vector counting and TF-IDF were used for experiments. In another method [21], a technique for classification of software requirement is employed which is based on ML. They investigated the application and design of two models such as convolutional neural network (CNN) and artificial neural network (ANN). The aim was to classify NFRs into 5 different categories such as usability, security, maintainability, performance, and operability. The method was tested by using a dataset having 1000 NFRs and obtained maximum precision of 94, recall of 97, and f-score of 92 percent. In another method [22], authors applied the amalgamation of LR, NB, and SVM, with Doc2Vec for classification of NFRs and FRs. Another combination of CNN, Word2vec and the FastText method is also applied on the PROMISE dataset. The outcomes of the FastText technique were best. Authors also used the combination of LR, NB, and SVM with BoW, and TF-IDF. The combination of Word2Vec and CNN algorithms is also utilized. The outcome of LR and TF-IDF attained best performance.

In the research [5], a method for prioritization of demand is presented for development of software. Initially, a dataset in Turkish language was introduced for demand prioritization. The

 doi:10.20944/preprints202410.2129.v1

dataset was based on records of manually labeled demands obtained from an insurance company demand management system. after that, a DL method is designed for improvement and prioritization of demands for software development. It was observed through experiments that DL method perform well as compared to other ML methods. To consider maintainability as security requirement, a method for classification of software is presented by authors [23]. The data was extracted from the projects of student. The technique is also verified by utilized DOSSPRE which contains 1317 requirements for software having both FRs and NFRs. The ML such as SVM, LR, Multinomial Nave Bayes (MNB) algorithms were used for classification. The accuracy of 86% is obtained on all of classifiers for both binary class and multiclass. In another method presented in [24], an automated method for quality attributes prioritization and extraction is presented. The attributes were considered in context of development of agile based method. The method is comprised of two components such as QAPrioritiser and QAExtractor. QAExtractor is based on natural language processing (NLP) and QAPrioritiser is used for ranking of computed attributes. The method is assessed by calculating F-score, recall and precision. In study [25], a fine tuning method based on three stages is presented for prediction of software requirement. The prediction is based on priority, type and severity of the text requirement initiated by user. The method is compared with various techniques such as Sentence BERT and pooling based on word embedding. The result shows that fine tunned model can perform well for data distribution. The study [26], proposed a hybrid model, Bi-LSTM+CNN, to address the weaknesses of LSTM and CNN individually. This model incorporates an attention mechanism to enhance accuracy and reduce the number of learnable parameters. By utilizing a CNN to extract features from various sentence locations, the Bi-LSTM model effectively reduces input features, resulting in improved accuracy, especially with larger training data sizes and epochs, offering a potential solution to long-term dependency and data loss issues in existing models.

Here is the state-of-the-art table summarizing various studies, techniques/models used, key findings, research gaps, and our contribution:

The exploration of various methodologies for automated requirement classification includes leveraging ML attributes computation, employing diverse classification techniques, as well as utilizing Natural Language Processing (NLP) tools. The methods encompass both traditional ML algorithms, such as Support Vector Machine (SVM), Logistic Regression (LR), and Multinomial Naive Bayes (MNB), and advanced DL methods like Convolutional Neural Networks (CNN) and Bidirectional Encoder Representations from Transformers (BERT). It covers ensemble methods and explores the amalgamation of different techniques, such as combining LR, Naive Bayes (NB), and SVM classifiers with techniques like Bag of Words (BoW), Term Frequency-Inverse Document Frequency (TF-IDF), Doc2Vec, and Word2Vec.

Our contribution focuses on addressing these gaps by utilizing the Bidirectional Long Short-Term Memory (Bi-LSTM) model, which offers significant improvements over the techniques and models discussed in the previous studies. Bi-LSTM's ability to capture and learn from sequential data more effectively enables it to enhance classification accuracy, improve context utilization, and provide a more robust solution for handling complex data dependencies. This model's bidirectional processing capability also allows for a more comprehensive understanding of input data, leading to better performance in classification tasks. By leveraging the strengths of Bi-LSTM, we aim to overcome the limitations identified in earlier studies, providing a more scalable, generalizable, and precise approach to machine learning classification.

These methods are assessed using metrics like precision, recall, F1-score, and accuracy to gauge their effectiveness in classifying requirements. The utilization of advanced techniques like BERT, CNN, and DL methods enables accurate categorization, which in turn enhances software reliability and quality. However, the drawbacks are also acknowledged, including challenges related to imbalanced data, feature extraction, and the complexity of requirement classification due to the diversity of terminologies and sentence structures. This research aims to address these limitations and provide insights into the optimal utilization of automated techniques to attain accurate and efficient requirement classification, thus contributing to the advancement of software engineering practices.

 doi:10.20944/preprints202410.2129.v1

The state-of-the-art studies presented in the Table 1 summarize various techniques and models used in machine learning and deep learning for classification tasks, with each study contributing unique insights and advancements in the field. Despite these contributions, several research gaps have been identified, including limited precision and recall rates, challenges in pre-processing and feature extraction, complexity in managing multiple deep learning methods, handling of imbalanced data, and the need for better generalization across datasets.

Table 1. Summary of ML Classification Techniques and Research Gaps.

Study	Techniques/Models	Key Findings	Research Gaps
Haque, and Rehman [15]	Combination of ML attributes computation and classification methods	Achieved precision 66%, F1-score 61%, recall 61%, ,accuracy 76%	Limited precision and recall rates
Dias and Cordeiro [3]	Comparison of ML algorithms for feature extraction	F1-score 74% for general classification using PROMISE_exp dataset	Challenges with preprocessing and feature extraction
Ivanov et al.[16]	Fine-tuned BERT compared with ELMo and fastText	F1-score 86% on PURE dataset, 80% on RFI dataset	Need for better generalization across datasets
Khayashi et al,[17]	DL methods with voting algorithm	Promising results with PURE dataset having 2617 FRs and 2044 NFRs	Complexity in managing multiple DL methods
Baker et al, [19]	CNN and ANN models	Maximum precision 94%, recall 97%, F1-score 92%	Handling of imbalanced data
Tiun et al, [20]	Combination of LR, NB, SVM with Doc2Vec and Word2Vec	FastText technique showed best outcomes	Complexity in integrating various techniques
Tunali and Volkan [5]	DL method for demand prioritization	DL method performed better than other ML methods	Scalability and generalization to different domains
Kadebu et al,[21]	Classification using SVM, LR, MNB	Accuracy 86% for both binary and multiclass classification	Requirement for larger and diverse datasets
Ahmed et al,[22]	Automated method for quality attributes prioritization and extraction	Assessed using F-score, recall, and precision	Improvement in context-aware classification
Yildirim et al,[23]	Fine-tuning method for software requirement prediction	Fine-tuned model outperformed other techniques	Need for robust evaluation metrics
Jang et al,[24]	Hybrid model Bi-LSTM+CNN with attention mechanism	Improved accuracy with larger training data sizes and epochs	Addressing long-term dependency and data loss issues

3. Proposed Framework

In this research, a Bi-LSTM model is used for requirements classification. The proposed model is based on some basic steps such as Tokenization the input, converting to lower case, removing the punctuation, feature extraction and classification. After the preprocessing of the data, the Bi-LSTM model is used to compute features, classifying the sequence into NFR or FR and training the model using carefully selected hyper-parameters to ensure optimal performance. Finally, the computed feature vector is fed into multiple classifiers for final classification as illustrated in Figure 1.

 doi:10.20944/preprints202410.2129.v1

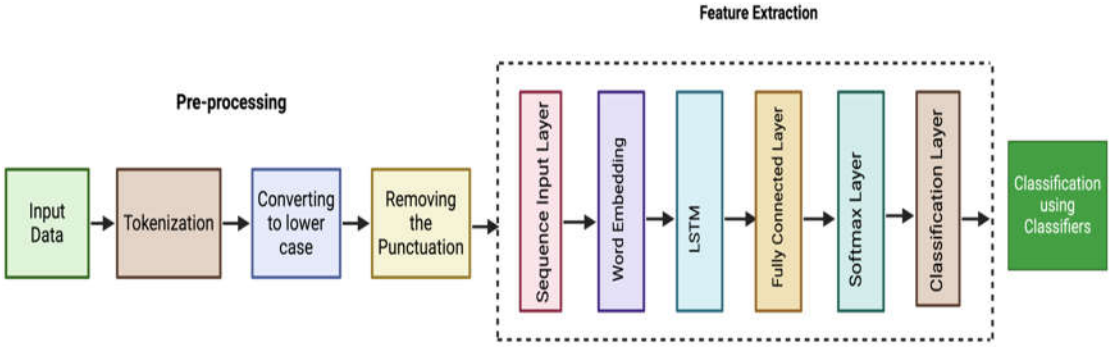


Figure 1. Proposed Architecture.

A. Data Pre-processing

In this section, a datasets available at [27] is obtained then we prepared a programming environment on the computer for the analysis and process of the dataset. In the preprocessing phase, we've selected techniques that are particularly effective for NFR classification. We also make sure that these techniques work well with our deep neural network. The benefits of employing these techniques include an increase in model accuracy and a decrease in both resource consumption and computation time. After preprocessing the data is divided into training and validation sets. The detail of these preprocessing steps is mentioned in Figure 2.

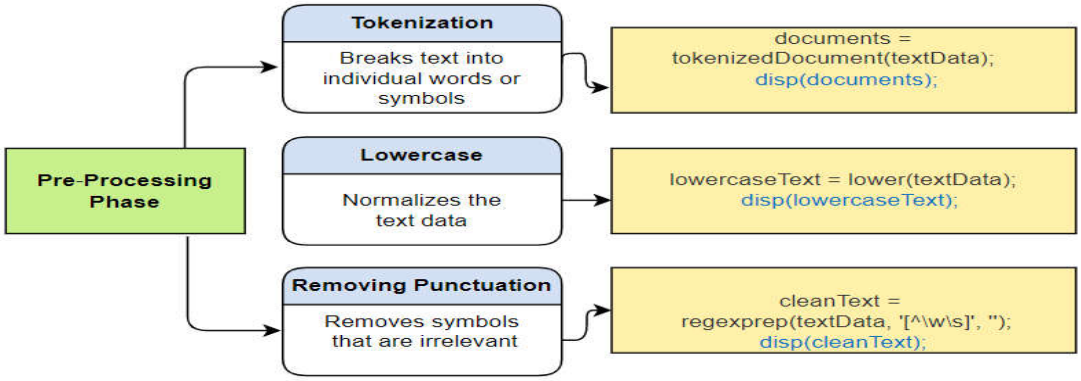


Figure 2. Preprocessing Phase.

B. Extraction of Features Vector by employing Bi-LSTM Model

The Bi-LSTM model is a powerful type of neural network used for sequential data processing, and is utilized for extraction of the features from input data. The Bi-LSTM model is based on the six layers as illustrated in Figure 3. These layers are Sequence Input, Word Embedding Layer [28], LSTM [29], Fully Connected [30], Softmax [31], and classification output.

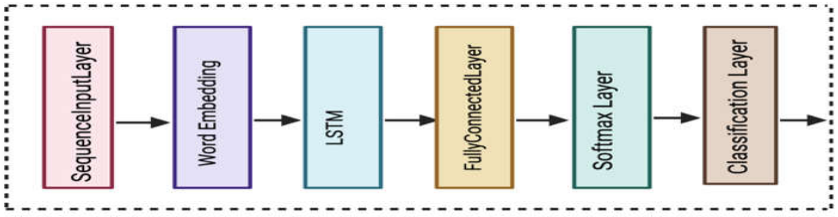


Figure 3. Bi-LSTM Layers.

The model is trained using the training dataset. For training the model, hyper-parameters such as the number of LSTM units in each layer, learning rate, batch size, and the number of epochs are specified. These parameters are finalized after extensive experiments. Sequence Input Layer and Word Embedding Layer, these two layers form the foundational part of Bi-LSTM model for feature

 doi:10.20944/preprints202410.2129.v1

extraction. We used input layer to capture the raw text sequences, and the embedding layer converted these sequences into meaningful dense representations that can be processed by the subsequent Bi-LSTM layers.

LSTM networks are considered an improvement over traditional Recurrent Neural Networks (RNNs), especially in tasks involving learning from sequences of data. Traditional RNNs struggle with long-term dependencies due to the vanishing gradient problem. The gating mechanisms in LSTMs allow for focusing on the most relevant features in a sequence. The LSTM is based on four gates such as (i) input gate, (ii) forget gate, (iii) output gate, and (iv) cell candidate gate. In the context of an LSTM network, the four gates along with the two key additional components Cell State and the LSTM Output work together to regulate the flow of information through the LSTM cell and the details are provided in Table 2.

Table 2. LSTM Network Detail.

LSTM Layer	Role	Formula
Input gate (i_t)	Determines which parts of the current input x(t) are relevant to be added to the cell state.	$i_t = \sigma\,(W_i\,[h_{t-1}, x_t] + b_i)$
Forget gate (f_t)	Decides which parts of the previous cell state C(t−1) should be forgotten.	$f_t = \sigma\,(W_f\,[h_{t-1}, x_t] + b_f)$
Output gate (O_t)	Determines which parts of the cell state will be outputted in the LSTM output h_t	$O_t = \sigma\,(W_o\,[h_{t-1}, x_t] + b_o)$
Cell candidate gate (c_t)	Generates a candidate vector of values that could be added to the cell state. The tanh function helps to keep these values normalized.	$c_t = \tanh\,(W_c\,[h_{t-1}, x_t] + b_c)$
Cell State(s_t)	Updates the cell state by combining the old state (modulated by the forget gate) and the new candidate values (modulated by the input gate).	$s_t = F \odot s_{t-1} + I_t \odot c_t$
LSTM Output (h_t)	Determines the final output of the LSTM unit at time t. The output gate controls which parts of the cell state are outputted.	$h_t = O_t \odot \tanh(c_t)$

Here, W_x and b_x represents the weights and biases of the respective gates.

$$W = \begin{bmatrix} RW_{input} \\ RW_{forget} \\ RW_{cell\ candidate} \\ RW_{output} \end{bmatrix}, \qquad bi = \begin{bmatrix} b_{input} \\ b_{forget} \\ b_{cell\ candidate} \\ b_{output} \end{bmatrix}$$

σ is the sigmoid function that squishes values between 0 and 1, effectively deciding how much of the past information to keep.

h_{t-1} represents the output of the LSTM block at time step t-1

x_t represents the time at current time step t

⊙ Represents the Hadamard product (element-wise multiplication) and it is used as the operation for combining the outputs of the gates with the cell state or candidate values.

In the LSTM, time steps of layers of LSTM are utilized to compute the feature vectors and give us input to the next block. The n^{th} block output is used to predict the next class label, in which the hidden unit is given to the fully connected, SoftMax, and the output. After extracting feature vectors with the Bi-LSTM, we fed these vectors into different classifiers, including Medium KNN (MKNN), Cubic KNN (CKNN), Linear SVM (LSVM), Quadratic SVM (QSVM), and Cubic SVM (CSVM). Each of these classifiers has its own approach to making predictions based on the feature vectors. Each classifier is trained on the training data and evaluated on the testing data. The performance of each

 doi:10.20944/preprints202410.2129.v1

classifier is assessed, and the best classifier is selected for the specific task. This process allows us for comparing different classification methods and choosing the one that works best.

4. Experimental Results

In this study, we rigorously evaluated the performance of a Bi-LSTM model for feature extraction from our dataset, followed by the application of these features to various classifiers to assess their efficacy in predictive analytics. Our experimental framework was structured to first fine-tune and train the Bi-LSTM model, ensuring optimal feature extraction, and then to employ these features in different classification algorithms. The results section is presented in two main parts: firstly, the training and optimization of the Bi-LSTM model, and secondly, the classification results obtained from deploying various classifiers , including Medium KNN (MKNN) [32], Cubic KNN (CKNN) [33], Linear SVM (LSVM) [34], Quadratic SVM (QSVM) [35], and Cubic SVM (CSVM) [36] using the extracted features. The Bi-LSTM model was trained on a comprehensive dataset [27]. The dataset is comprised of a total of 12 classes and are represented in Table 3 along with their class symbols.

Table 3. Dataset Classes used for Experiment.

Class Name	Class Symbol	Class Name	Class Symbol
Functional	F	Operational	O
Availability	A	Performance	PE
Fault Tolerance	FT	Portability	PO
Legal	L	Scalability	SC
Look & Feel	LF	Security	SE
Maintainability	MN	Usability	US

Dataset was divided into a training set and a testing set in a 70: 30 ratio. This means that 70% of the data was used for training the Bi-LSTM model, and the remaining 30% was used for evaluating the model's performance. The 10-fold cross-validation method was utilized in which the training data was further divided into 10 equal parts (folds). The model was trained and evaluated 10 times, each time using a different fold as the test set and the remaining nine folds as the training set. This helped in assessing the model's generalization performance. A learning rate of 0.001 was used during the training of the Bi-LSTM model. The learning rate is a hyper-parameter that controls the step size at which the model's parameters are updated during training. A smaller learning rate can lead to slower but more stable convergence during training. The entire experimentation and analysis were carried out using MATLAB 2021A. The experiments were conducted on a computer system with specific hardware specifications. We used a Core i7 system with 16GB of RAM (Random Access Memory) and an NVIDIA 940MX GPU (Graphics Processing Unit) with 4GB of memory. The GPU can significantly speed up DL tasks due to its parallel processing capabilities. For computing features we used Matcovnet [37] as the DL toolbox. This toolbox is likely used to implement and train the Bi-LSTM model and perform feature extraction.

During of our proposed Bi-LSTM model, we undertook a series of extensive experiments to select the optimal hyper-parameters for training. This involved conducting various experiments, each designed to evaluate the effectiveness of varying hyper-parameter configurations. The results of four experiments are organized in Table 4.

Table 4. Parameters used in Experiments and the Corresponding Training Accuracies.

 doi:10.20944/preprints202410.2129.v1

Exper.	Batch Size	Learning Rate	Epochs	Accuracy (%)	Figure
1	16	0.001	50	61.26	4
2	16	0.001	100	66.67	5
3	16	0.001	150	72.97	6
4	16	0.001	200	66.67	7

These experiments collectively demonstrate how varying the number of epochs while holding other parameters steady can impact the training effectiveness of the Bi-LSTM model. A lot of padding is occurred while training the model which results the overhead and also affects classification. This problem is addressed by limiting the length of sequence to 60. Experiment 3 emerged as the most successful, achieving the highest accuracy among all trials. Based on these findings, we selected the hyper-parameters used in Experiment 3 for our final model configuration.

This includes setting the model with 150 hidden units, a manageable batch size of 16, a learning rate of 0.001, and a comprehensive training duration of 150 epochs. The decision to adopt these specific parameters was driven by their demonstrated ability to significantly enhance the model's performance.

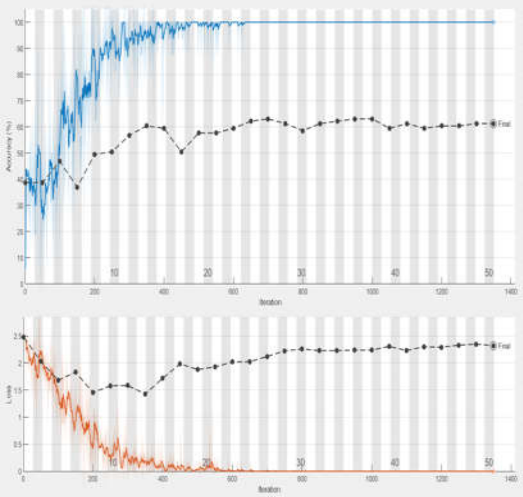


Figure 4. Training Accuracy of Bi-LSTM in Experiment 1**.**

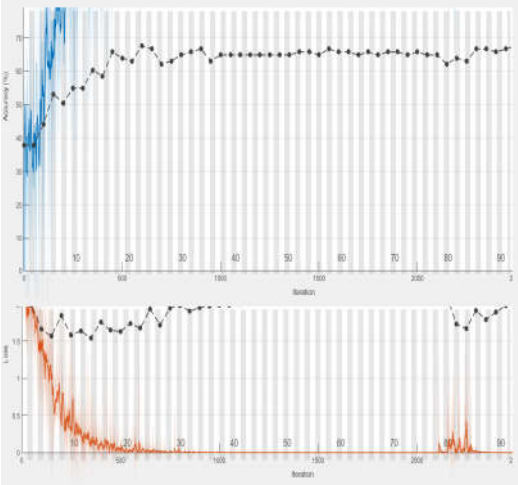


Figure 5. Training Accuracy of Bi-LSTM in Experiment 2.

 doi:10.20944/preprints202410.2129.v1

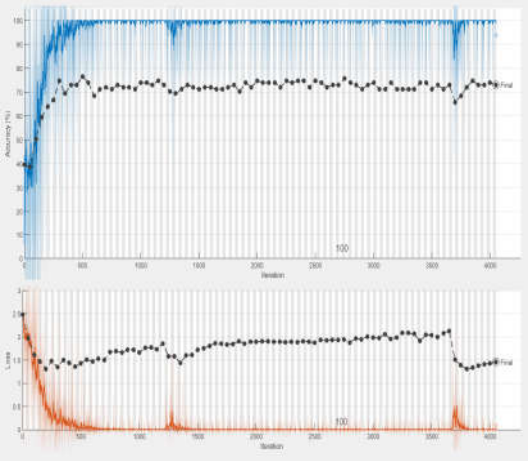


Figure 6. Training Accuracy of Bi-LSTM in Experiment 3 .

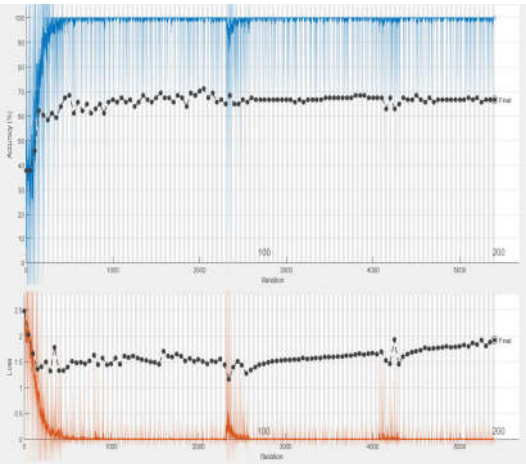


Figure 7. Training Accuracy of Bi-LSTM in Experiment 4.

After the successful extraction of features, the resultant feature vector (FV) was subjected to a series of classification tests to evaluate the efficacy of the extracted features. Utilizing MATLAB, we applied several well-established classifiers, namely Medium KNN (MKNN), Cubic KNN (CKNN), Linear SVM (LSVM), Quadratic SVM (QSVM), and Cubic SVM (CSVM). This phase was crucial to determine the predictive power of the features under various classification schemes. The computational cost, measured in seconds (sec), is also evaluated. This indicates how much time it takes for the system to perform the experiments and computations, which is important for assessing the system's efficiency and response time. When feature vectors are fed into classifiers, the classifier's performance evaluation relies heavily on metrics such as Recall (Rec), Precision (Pr), and Accuracy (Acc). These metrics are essential for understanding how well the model can classify and predict outcomes, providing insights into different aspects of its effectiveness. Table 5 summarizes the key aspects of these metrics.

TP (True Positives): Instances correctly identified as positive.
TN (True Negatives): Instances correctly identified as negative.
FP (False Positives): Instances incorrectly identified as positive.
FN (False Negatives): Instances incorrectly identified as negative.

To visually assess and compare the performance of these classifiers, scatter plots are utilized. Such plotting allows a graphical representation of the classifier's outcomes in a two-dimensional space. Each point on the scatter plot corresponds to a feature vector and is colored according to the classifier's prediction. This visualization technique enables a quick and intuitive way to understand

 doi:10.20944/preprints202410.2129.v1

how well each classifier is segregating the data points. The legend of a scatter plot helps to understand the plot by providing information about the different colors represented within the plot. The verification of the results by using namely MKNN, CKNN, LSVM, QSVM, and CSVM classifiers is presented with the help of scatter plots as shown in Figures 8–12. The detailed information about how each classifier performs in terms of these metrics and how long it takes them to compute their results is described in Table 5.

Table 5. Information about Recall,Precision and Accuracy.

Metric	Formula	Role
Recall (Rec)	TP / (TP + FN)	Measures the classifier's ability to identify all relevant instances.
Precision (Pr)	TP / (TP + FP)	Assesses the accuracy of the classifier when it predicts a positive class.
Accuracy (Acc)	(TP + TN) / (TP + TN + FP + FN)	Gives an overall idea of how often the classifier is correct.

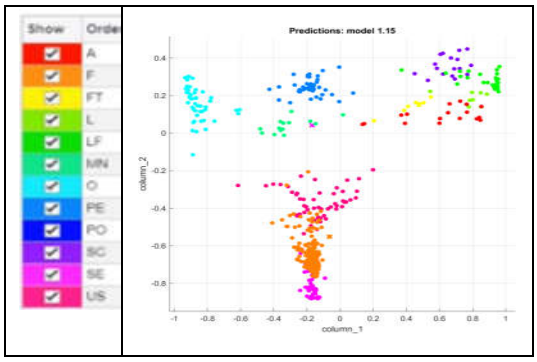


Figure 8. Scatter Plot of MKNN Results.

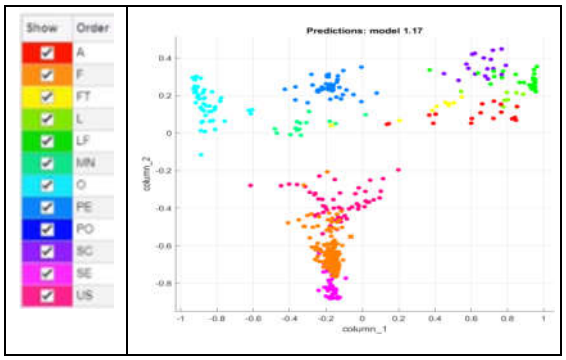
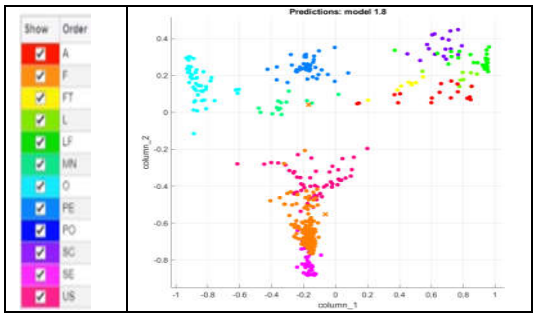


Figure 9. Scatter plot of CKNN Results.



 doi:10.20944/preprints202410.2129.v1

Figure 10. Scatter plot of LSVM Results.

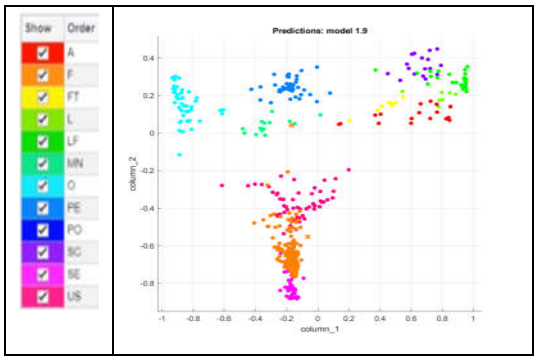


Figure 11. Scatter plot of LSVM Results.

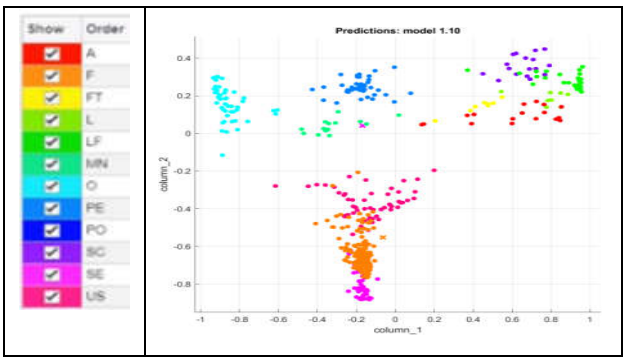


Figure 12. Scatter plot of CSVM Results.

According to the results chart Table 6, the MKNN classifier outperforms the others, achieving the highest precision and recall scores compared to other classifiers. Additionally, MKNN demonstrates competitive computational efficiency, with a relatively low time of 1.5199 seconds. Therefore, MKNN emerges as a good classifier due to its balanced performance in recall, precision and significantly faster computation time.

Table 6. Results by using the Proposed Classifiers.

Classifier	Recall	Precision	Accuracy	Time (Sec)
MKNN	91.48	91.56	99.60	1.5199
CKNN	91.48	91.43	99.60	2.3769
LSVM	91.43	91.38	99.60	10.3560
QSVM	91.43	91.38	99.80	9.5692
CSVM	91.26	90.56	99.80	7.3273

Discussion

In this study, we present a system for classification of requirements such as FRs and NFRs employing a Bi-LSTM model. Our approach involves training the model with various parameter configurations, including learning rate, epochs, and mini-batch size. Throughout our experiments, all parameters remain consistent except for the number of epochs, which varies across different trials: 50 epochs in Experiment 1, 100 epochs in Experiment 2, 150 epochs in Experiment 3, and 200 epochs in Experiment 4. While training the model, it is observed that the accuracy of the model was gradually increasing when number of epochs changes from 50 to 100. More improvement was observed when the number of epochs was increased to 150. But when the number of epochs was set to 200 resulted in the decline of system performance. During experiments, it is observed that the most affecting

 doi:10.20944/preprints202410.2129.v1

parameter for training the model is the number of epochs. Specifically, we find that the highest training accuracy of 72.97% is achieved with 150 epochs, signifying an optimal balance between training duration and model effectiveness. Consequently, we designate the parameter configuration from Experiment-3 as the final setting for subsequent requirement classification tasks. This empirical exploration elucidates the importance of fine-tuning training parameters, particularly the number of epochs, in optimizing the performance of Bi-LSTM models for requirement classification.

V. Threats to Validity

When employing a Bi-LSTM model for feature extraction followed by the utilization of different ML classifiers for feature classification, several threats to validity must be considered. Firstly, there is a risk of external validity threat if the dataset used for training and evaluation does not adequately represent the real-world distribution of feature vectors. To mitigate the risk of external validity threat associated with dataset representativeness, several measures have been implemented. The efforts have been made to collect or generate a diverse and representative dataset that reflects the real-world distribution of feature vectors. Furthermore, the dataset is split into training and testing subsets, with careful consideration given to maintaining the integrity of the distribution across these partitions. Cross-validation techniques are also utilized to assess model generalization across multiple folds of the data. Secondly, there is a potential threat to internal validity arising from the selection of hyper-parameters during the training of the Bi-LSTM model, such as the number of layers, hidden units, and learning rate, which could influence the quality and relevance of the extracted features. A systematic approach to hyper-parameter selection has been adopted, involving cross-validation techniques to ensure robustness and generalizability of the Bi-LSTM model. This process ensures thorough evaluation and comparison of various combinations of hyper-parameters, techniques such as early stopping have been implemented to prevent over fitting and enhance generalization performance. Additionally, the choice of ML classifiers introduces another layer of complexity, as different algorithms may perform variably based on the nature of the extracted features. This leads to a potential threat to construct validity, as the classifiers may not accurately capture the underlying structure of the data, resulting in biased comparisons. To mitigate the threat posed by the variability in ML classifier performance and its potential impact on construct validity, we evaluated each classifier thoroughly using standard performance metrics such as accuracy, precision, recall, and F1-score. By systematically comparing the performance of multiple classifiers on the same set of extracted features, we aim to mitigate biases and ensure a more accurate representation of the underlying data structure, thus enhancing the construct validity of our analysis. By implementing these measures, we aim to enhance the validity and generalizability of our findings when applying Bi-LSTM models for feature extraction and classification using different machine learning classifiers.

6. Conclusion

In this work, our aim was to classify the user requirement in the field of RE. Traditional methods of user requirements classification often struggle to handle the complexity and variability present in natural language, making it difficult to capture context-dependent features of user requirements. Furthermore, these methods may lack scalability, as they may not be equipped to handle large volumes of requirements data efficiently. We utilized a DL technique named as Bi-LSTM model for feature extraction followed by the integration of these feature vectors into five distinct ML classifiers namely MKNN, CKNN, LSVM, QSVM, and CSVM. We observed that MKNN stands out as an effective classifier due to its balanced performance in recall, precision and notably accelerated computation time. The adoption of this technique promises to revolutionize requirement classification by enabling efficient and accurate identification of NRFs and FRs. Notably, our approach surpasses previous methodologies due to several key factors. Firstly, Bi LSTM has a bidirectional processing capability of capturing both forward and backward contextual dependencies ensured a more comprehensive understanding of the input sequence. Secondly, the utilization of multiple ML classifiers offered a robust evaluation framework, ensuring a comprehensive analysis of the model's performance across various algorithms. Furthermore, our emphasis on both performance

 doi:10.20944/preprints202410.2129.v1

metrics and computational efficiency underscores our commitment to practical applicability and scalability. Overall, our study represents a significant contribution to the domain by offering a refined methodology that outperforms previous approaches in terms of accuracy, versatility, and computational efficiency.

Future Work

Our aim is to test this technique on some other datasets exploring its adaptability across various contexts within the software engineering domain. Through persistent refinement efforts and extensive testing endeavors, our ambition is to catalyze the progression of automated requirement classification, bringing about heightened levels of accuracy and increased efficiency in software development practices.

Author Contributions: Conceptualization, Jalil Abbas, Zhuoxuan hu, & Saima Kanwal; methodoogy, Jalil Abbas.; software,. Jalil Abbas & Arshad Ahmad ; writing—original draft preparation, Jalil Abbas.; writing—review and editing, Jalil Abbas,Saima Kanwal & Arshad Ahmad.; funding acquisition: Ahmad Almogren & Ayman Altameem. All authors have read and agreed to the published version of the manuscript.

Funding: This work was supported by King Saud University, Riyadh, Saudi Arabia, through Researchers Supporting Project number (RSP2024R498).

Data Availability Statement: The data that support the findings of this study are publicly available on: https://www.kaggle.com/datasets/iamsouvik/software-requirements-dataset.

Data Availability Statement: The data used in this work are available at Kaggle.

Conflicts of Interest: The authors confirm there are no conflicts of interest associated with this research.

Acknowledgement: This work was supported by King Saud University, Riyadh, Saudi Arabia, through Researchers Supporting Project number (RSP2024R498).

References

[1] Ahmed, H.; Traore, I.; Saad, S. Detecting opinion spams and fake news using text classification. *Secur. Priv.* **2018**, *1*, e9.

[2] Mallick, P.K.; Mishra, S.; Chae, G.S. Digital media news categorization using Bernoulli document model for web content convergence. *Pers. Ubiquit. Comput.* **2023**, *27*, 1087–1102. https://doi.org/10.1007/s00779-020-01461-9.

[3] Dias Canedo, E.; Cordeiro Mendes, B. Software Requirements Classification Using Machine Learning Algorithms.Entropy 2020, 22(9), 1057. https://doi.org/10.3390/e22091057.

[4] Tunali, V.; Tüysüz, M.A.A.J.P.Ü.M.B.D. Analysis of Function-Call Graphs of Open-Source Software Systems Using Complex Network Analysis**.** *Pamukkale Univ. J. Eng. Sci.* **2020,** *26*(2), 352–358.

[5] Tunali, V.J.I.A. Improved Prioritization of Software Development Demands in Turkish with Deep Learning-Based NLP. *IEEE Access* **2022**, *10*, 40249-40263.

[6] Khan, F.M.; Khan, J.A.; Assam, M.; Almasoud, A.S.; Abdelmaboud, A.; Hamza, M.A.M. A Comparative Systematic Analysis of Stakeholder's Identification Methods in Requirements Elicitation. *IEEE Access* **2022**, *10*, 30982-31011.

[7] Abad, Z.S.H.; Karras, O.; Ghazi, P.; Glinz, M.; Ruhe, G.; Schneider, K. What Works Better? A Study of Classifying Requirements. In *Proceedings of the 2017 IEEE 25th International Requirements Engineering Conference (RE)*, Sep. 2017, pp. 496–501. IEEE. https://doi.org/10.1109/RE.2017.36.

[8] Kanwal, J.; Maqbool, O. Bug Prioritization to Facilitate Bug Report Triage. *J. Comput. Sci. Technol.* **2012**, *27*, 397-412.

[9] Sepahvand, R.; Akbari, R.; Hashemi, S.; Boushehrian, O. An Effective Model to Predict the Extension of Code Changes in Bug Fixing Process Using Text Classifiers. *Iran. J. Sci. Technol. Trans. Electr. Eng.* **2022**, *1–18*.

[10] Javidi, Z.; Akbari, R.; Bushehrian, O. A New Method Based on Formal Concept Analysis and Metaheuristics to Solve Class Responsibility Assignment Problem. *Iran J. Comput. Sci.* **2021**, *4*, 221-240.

[11] Etemadi, V.; Bushehrian, O.; Akbari, R.; Robles, G. A Scheduling-Driven Approach to Efficiently Assign Bug Fixing Tasks to Developers. *J. Syst. Softw.* **2021**, *178*, 110967. https://doi.org/10.1016/j.jss.2021.110967.

[12] Yousefi, M.; Akbari, R.; Moosavi, S.M.R. Using Machine Learning Methods for Automatic Bug Assignment to Developers. *J. Electr. Comput. Eng. Innov.* **2020**, *8*(2), 263–272.https://doi.org/10.22061/JECEI.V8I2.2853.

[13] Lima, M.; Valle, V.; Costa, E.; Lira, F.; Gadelha, B. Software Engineering Repositories: Expanding the Promise Database. In *Proceedings of the XXXIII Brazilian Symposium on Software Engineering*, Sep. 2019, pp. 427–436.

[14] Rahman, M.A.; Haque, M.A.; Tawhid, M.N.A.; Siddik, M.S. Classifying Non-Functional Requirements Using RNN Variants for Quality Software Development. In *Proceedings of the 3rd ACM SIGSOFT International Workshop on Machine Learning Techniques for Software Quality Evaluation*, Aug. 2019, pp. 25–30.

[15] Habibullah, K.M.; Gay, G.; Horkoff, J. Non-Functional Requirements for Machine Learning: Understanding Current Use and Challenges Among Practitioners. *Req. Eng.* **2023**, *28*(2), 283–316. https://doi.org/10.1007/s00766-022-00374-5.

[16] Hassan, S., et al. A systematic mapping to investigate the application of machine learning techniques in requirement engineering activities. *CAAI Transactions on Intelligence Technology* **2024**, *1–23*. https://doi.org/10.1049/cit2.1234.

[17] Haque, M.A.; Rahman, M.A.; Siddik, M.S. Non-Functional Requirements Classification with Feature Extraction and Machine Learning: An Empirical Study. In *Proceedings of the 2019 1st International Conference on Advances in Science, Engineering and Robotics Technology (ICASERT)*, May 2019, pp. 1–5. IEEE. https://doi.org/10.1109/ICASERT.2019.8934578.

[18] Ivanov, V.; Sadovykh, A.; Naumchev, A.; Bagnato, A.; Yakovlev, K. Extracting software requirements from unstructured documents. In *Proceedings of the International Conference on Analysis of Images, Social Networks and Texts*, Cham, Switzerland, December 2021; Springer International Publishing, pp. 17–29.

[19] Khayashi, F.; Jamasb, B.; Akbari, R.; Shamsinejadbabaki, P. Deep learning methods for software requirement classification: A performance study on the pure dataset. *arXiv* **2022**, arXiv:2211.05286.

[20] Rahimi, N.; Eassa, F.; Elrefaei, L. An Ensemble Machine Learning Technique for Functional Requirement Classification. *Symmetry* **2020**, *12*, 1601. https://doi.org/10.3390/sym12101601.

[21] Baker, C.; Deng, L.; Chakraborty, S.; Dehlinger, J. Automatic Multi-Class Non-Functional Software Requirements Classification Using Neural Networks. *Proceedings of the 2019 IEEE 43rd Annual Computer Software and Applications Conference (COMPSAC)* 2019, 2, 610–615. https://doi.org/10.1109/COMPSAC.2019.00105.

[22] Tiun, S.; Mokhtar, U.A.; Bakar, S.H.; Saad, S. Classification of Functional and Non-Functional Requirements in Software Requirement Using Word2vec and FastText. *J. Phys. Conf. Ser.* **2020**, *1529*, 042077. https://doi.org/10.1088/1742-6596/1529/4/042077.

[23] Kadebu, P.; Sikka, S.; Tyagi, R.K.; Chiurunge, P. A Classification Approach for Software Requirements Towards Maintainable Security. *Sci. Afr.* **2023**, *19*, e01496. https://doi.org/10.1016/j.sciaf.2023.e01496.

[24] Ahmed, M.; Khan, S.U.R.; Alam, K.A. An NLP-Based Quality Attributes Extraction and Prioritization Framework in Agile-Driven Software Development. *Autom. Softw. Eng.* **2023**, *30*(1), 7. https://doi.org/10.1007/s10515-023-00357-7.

[25] Yildirim, S.; Cevik, M.; Parikh, D.; Basar, A. Adaptive Fine-Tuning for Multiclass Classification Over Software Requirement Data. *arXiv* **2023**, arXiv:2301.00495.

[26] Jang, B.; Kim, M.; Harerimana, G.; Kang, S.U.; Kim, J.W. Bi-LSTM Model to Increase Accuracy in Text Classification: Combining Word2vec CNN and Attention Mechanism. *Appl. Sci.* **2020**, *10*(17), 5841. https://doi.org/10.3390/app10175841.

[27] Software Requirements Dataset. Kaggle, 2022. [Online]. Available: https://www.kaggle.com/datasets/iamsouvik/software-requirements-dataset. [Accessed: Sep. 18, 2024].

[28] Peters, M.E.; Neumann, M.; Zettlemoyer, L.; Yih, W.T. Dissecting Contextual Word Embeddings: Architecture and Representation. *arXiv* **2018**, arXiv:1808.08949.

[29] Kwon, B.S.; Park, R.J.; Song, K.B. Short-Term Load Forecasting Based on Deep Neural Networks Using LSTM Layer. *J. Electr. Eng. Technol.* **2020**, *15*, 1501–1509. https://doi.org/10.1007/s42835-020-00478-0.

[30] Basha, S.; Dubey, S.R.; Pulabaigari, V.; Mukherjee, S. Impact of Fully Connected Layers on Performance of Convolutional Neural Networks for Image Classification. *Neurocomputing* **2020**, *378*,112–119. https://doi.org/10.1016/j.neucom.2019.10.008.

[31] Hu, R.; Tian, B.; Yin, S.; Wei, S. Efficient Hardware Architecture of Softmax Layer in Deep Neural Network. In *Proceedings of the 2018 IEEE 23rd International Conference on Digital Signal Processing (DSP)*, Shanghai, China, November 2018; IEEE, pp. 1–5. https://doi.org/10.1109/ICDSP.2018.8631621.

[32] Ali, A.; Alrubei, M.A.; Hassan, L.F.M.; Al-Ja'afari, M.A.; Abdulwahed, S.H. Diabetes Diagnosis Based on KNN. *IIUM Eng. J.* **2020**, *21*(1), 175–181. https://doi.org/10.31436/iiumej.v21i1.1226.

[33] Maghari, A. Prediction of Student's Performance Using Modified KNN Classifiers. In *Proceedings of the First International Conference on Engineering and Future Technology (ICEFT 2018)*, Cairo, Egypt, 2018; pp. 143–150.

[34] Chauhan, V.K.; Dahiya, K.; Sharma, A. Problem Formulations and Solvers in Linear SVM: A Review. *Artif. Intell. Rev.***2019**, *52*, 803–855. https://doi.org/10.1007/s10462-018-9614-6.

[35] Altay, O.; Ulas, M.; Alyamac, K.E. Prediction of the Fresh Performance of Steel Fiber Reinforced Self-Compacting Concrete Using Quadratic SVM and Weighted KNN Models. *IEEE Access* **2020**, *8*, 92647–92658. https://doi.org/10.1109/ACCESS.2020.2994730.

[36] Jain, U.; Nathani, K.; Ruban, N.; Raj, A.N.J.; Zhuang, Z.; Mahesh, V.G. Cubic SVM Classifier Based Feature Extraction and Emotion Detection from Speech Signals. In *Proceedings of the 2018 International Conference on Sensor Networks and Signal Processing (SNSP)*, Xi'an, China, October 2018; IEEE, pp. 386–391. https://doi.org/10.1109/SNSP.2018.00076

[37] Vedaldi, A.; Lenc, K. MatConvNet: Convolutional Neural Networks for MATLAB. In *Proceedings of the 23rd ACM International Conference on Multimedia*, Brisbane, Australia, October 2015; pp. 689–692. https://doi.org/10.1145/2733373.2807412.