Hybrid Multiple Deep Learning Models to Boost Blocking Bug Prediction

Zhihua Chen, Xiaolin Ju*, Haochen Wang, Xiang Chen

School of Information Science and Technology, Nantong University, China
zhihuachen19@gmail.com, ju.xl@ntu.edu.cn, haochenwang@ntu.edu.cn, xchencs@ntu.edu.cn

Abstract

A blocking bug (BB) is a severe bug that could prevent other bugs from being fixed in time and cost more effort to repair itself in software maintenance. Hence, early detection of BBs is essential to save time and labor costs. However, BBs only occupy a small proportion of all bugs during software life cycle, making it difficult for developers to identify these blocking relationships. This study proposes a novel blocking bug prediction approach based on the hybrid deep learning model, a combination of Bi-directional Long Short-Term Memory (Bi-LSTM) and Convolutional Neural Network (CNN). Our approach first extracts summaries and descriptions from bug reports to construct two classifiers, respectively. Second, our approach combines two classifiers into a hybrid model to predict the blocking relationship of each blocking bug pair. Finally, our approach produces a report of identified blocking bugs for developers. To investigate the effectiveness of proposed approach, we conducted an empirical study on bug reports of seven large-scale projects. The final experimental results illustrate that our approach can perform better than the recent state-of-the-art baselines. Precisely, the hybrid model can predict BB better with an average accuracy of 57.20%, and an improvement of 73.53% in terms of the F1-measure when compared to ELBlocker. Moreover, according to the bug report's description, BB can be predicted well with an average accuracy of 49.16%.

Keywords: Blocking bug, Deep learning, Bug report analysis

1 Introduction

Software today plays a significant role in controlling the behavior of many systems. A failure caused by the software operation can have catastrophic consequences, including property damage, financial loss, and serious injury or death [1-2]. Bug tracking systems such as Bugzilla report numerous software bugs. Among these bugs, there is a particular association named blocking bug pairs (BBP), which means one unfixed bug blocks another bug from being fixed [3]. Blocking bugs have higher complexity than non-blocking bugs. Although the blocking bugs that make up the BBP occupy a small proportion of all bugs, the unfixed bug of BBP in upstream projects will prevent fixing the bug in downstream projects [4]. Blocking bug require more time and effort in software maintenance. A previous study showed that fixing blocking bugs usually takes 2 to 3 times longer and more lines of code than fixing non-blocking bugs [5-6].

Many approaches may effectively predict software fault proneness [7-8], but they cannot separate blocking bugs from other bugs. As a result, efforts have been devoted to automatically identify blocking bugs in order to alleviate the impact of blocking bugs. In previous studies, Garcia et al. applied various machine learning techniques to build classifiers for identifying blocking bugs from all reported bugs [5]. Later, Xia et al. proposed ELBlocker which divided the training data into multiple disjoint sets, built a separate classifier for each group, combined these classifiers, and automatically determined an appropriate imbalanced decision boundary (or threshold) to distinguish blocking bugs (BBs) from non-blocking bugs [9].

In recent years, deep learning has succeeded in many natural language processing tasks, especially in bug report analysis during software development. For instance, Huo et al. proposed NP-CNN which used lexical information and program structure information to learn unified features from the natural language and source code and then automatically located potential bug source code based on bug reports [10]. We aim to investigate the effectiveness of identifying blocking bugs by applying hybrid deep learning techniques. Specifically, we apply two deep learning techniques (i.e., Bi-LSTM and CNN) to construct a hybrid prediction model. We propose Bi-LSTM to perform bugs classification by bug reports. Bi-LSTM is inherently suitable for such analysis because they can capture both forward and backward correlated information within blocking-bug pairs. we extract the bug's summary and description from the bug report to build the training data set. We do this by assuming that the summary and description can represent the characteristics of the examined bugs. We suppose that these features can be applied to train our hybrid model and are conducive to more accurate blocking bugs identification.

To evaluate the effectiveness of our approach, we conducted an empirical study on seven large open-source projects. At the same time, four evaluation indicators are used to measure the performance of our approach.

The main contributions of our study can be summarized as follow:

- A novel blocking bug prediction model based on hybrid deep learning techniques (i.e., Bi-LSTM and CNN).

*Corresponding Author: Xiaolin Ju; E-mail: ju.xl@ntu.edu.cn
DOI: 10.53106/160792642022092305018

- A strategy to couple the classifiers based on Bi-LSTM and CNN.
- An empirical study that compared our proposed approach with the two baselines (i.e., ELBlocker and Garcia's approach) on five open-source projects, which indicate that our approach can achieve significant performance improvements over compared baselines.

2 Background

This section introduces the characteristics of blocking bugs and techniques applying deep learning to classify bug reports. More specifically, we first described the components of the bug report and the characteristics of the blocking bug. Then, we analyze related work employing deep learning techniques to manipulate bug reports and classify bugs.

2.1 Blocking Bugs

A bug report summarizes the necessary information needed to document, report, and fix bugs in software. Usually, the bug report consists of free text and non-textual information (i.e., code segments, screenshots or device logs, stack traces of program execution, error messages, etc.) that outline information about causes or seen failures [11]. Semantically speaking, a bug report usually contains several descriptive characteristics of the detected bug to point out precisely what is considered faulty. Generally, the free text includes a summary, a description, and some comments. The summary and description present the detailed preliminary information of the reported bug. These comments are usually posted by developers who have an interest in resolving the assigned bug [12].

Among all types of bugs, blocking bugs are a particular type of bug attracting the attention of more and more researchers. Specifically, the blocking bug is a bug that prevents other bugs from being fixed before it has been fixed and consequently needs more effort to fix it. Compared with non-blocking bugs, blocking bugs usually take two to three times longer to be fixed [4]. Studying the characteristics of blocking bugs is conducive to early detection of blocking bugs and giving more maintenance resources in time. Therefore, research on blocking bugs is essential for improving the efficiency of software maintenance and ensuring software quality.

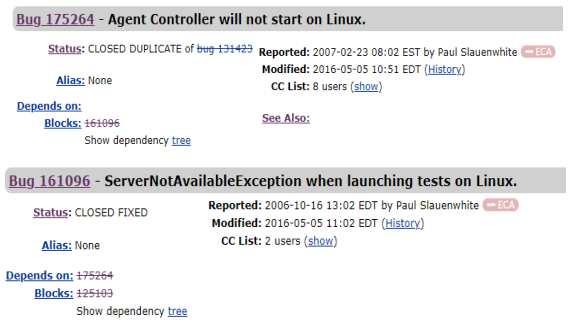


Figure 1. An example of a blocking bug pair in Eclipse's bug report with BugID=175264, BugID=161096.

An example of two blocking bug reports in Eclipse is shown in Figure 1. From these bug reports, we can obtain the following observations: Blocking bugs usually take a long time to fix. For example, Bug 175264 was opened on February 23, 2007, and was fixed on May 05, 2016. It took nine years to repair the bug (BugID=175264). At the same time, this bug (BugID=175264) also blocked Bug 161096. Bug 161096 was created on October 16, 2006 and was repaired on May 05, 2016. That is to say. It took more than 10 years to repair this bug (BugID=161096).

2.2 Bug Report Analysis applying Deep Learning

In recent years, deep learning models have become popular and have achieved great success in many natural language processing tasks. For example, Johnson et al. employed Convolutional Neural Networks (CNN) to provide an alternative mechanism to effectively use the word order of text classification by directly embedding short text regions [13]. To alleviate the difficulty of learning the long-term dynamics of the Recurrent Neural Network (RNN) for text processing, Long-Short-Term Memory (LSTM) integrates memory units to know when to forget the past. The status of and when to update the current situation to get new information was proposed [14-17]. We use Bi-directional Long Short-Term Memory (Bi-LSTM) and Convolutional Neural Network (CNN) in our approach. Next, we will introduce these two methods.

Bi-LSTM, a variant of Recurrent Neural Network (RNN), is a two-way Long Short-Term Memory network (LSTM). As we all know, the basic idea behind Recurrent Neural Network (RNN) is to use the information present in a given sequence and calculate the output. However, the text information with a long distance cannot be used by RNN. LSTM solves the memory problem with a directional unidirectional propagation structure to alleviate this issue. When processing text sequences, the model can use only the previous text to predict the subsequent text, and the prediction of the previous part cannot be given by analyzing the content after the prediction. However, this unidirectional communication structure has some shortcomings in predicting text content. Therefore, we utilize Bi-LSTM to solve this issue. It considers both forward and reverses semantics, which largely compensates for the weaknesses of LSTM and improves text analysis and prediction powers.

Researchers initially proposed CNN (Convolutional Neural Network) in the context of image classification. However, it has also been widely used in text classification tasks and has shown promising results recently. It first performs a multi-layer convolution operation and then converts the convolution output of each layer with a nonlinear activation function. Each local input area is connected to a neuron output in the convolution process. Different convolution kernels are applied to each layer, and each type of convolution kernel can be considered as a feature extracted, and then multiple features are added. CNN is a network with incomplete connection and weight sharing, which reduces the complexity of the model to a certain extent. The existence of the convolutional layer enables CNN to capture the local spatial correlation better, and the pooling layer in the neural network can significantly reduce the amount of calculation of the model [18]. Word2Vec turns natural language text into a vector pattern that can be processed by neural networks,

enabling CNN to obtain local information in the text and effectively improve the performance of automatic bug classification [19].

3 Our Approach

The overall goal of our work is to provide an effective solution for distinguishing blocking bugs from non-blocking bugs. We instantiate the learning task by proposing a hybrid deep learning model — a combination of Bi-directional Long Short-Term Memory (Bi-LSTM) and Convolutional Neural Network (CNN). We constructed two weak classifiers and combined them through a fully connected network using the bug report's summary and description as training data. In this way, the model achieves better performance. As shown in Figure 2, our approach mainly consisted of three phases: **Pre-Processing**, **Model Construction** and **Prediction Report**.

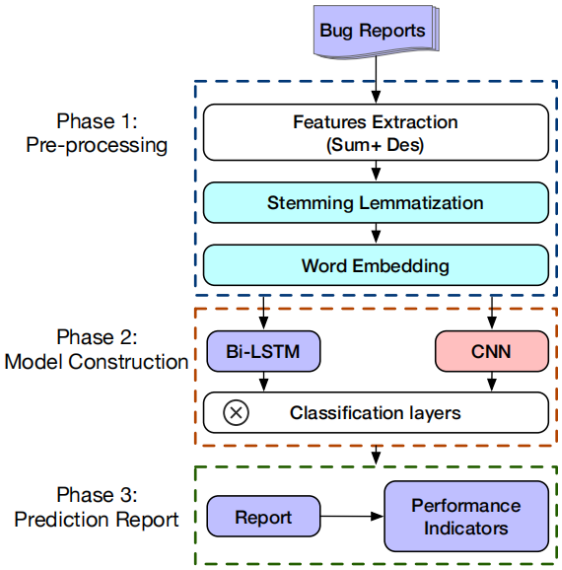


Figure 2. The framework of our approach

Next, we will introduce the framework of our approach in the following subsections.

3.1 Pre-Processing

We extract the summaries and descriptions from the bug reports. Summary and description are an important part of bug reports and contain rich textual information. In Garcia's empirical research, it is shown that textual information in bug reports is an important factor in understanding bug characteristics, representing a rich source of unstructured information [5]. Accordingly, we use these two parts of data (i.e., summaries and descriptions) as the primary input of our hybrid model.

To employ the text content as the input of the neural network, they must be processed earlier to be a more acceptable form. There are two steps for data pre-processing as follow:

First, we conduct a stemming morphological restoration of the text. The words in the bug summary and description usually have multiple morphological variants, and different morphological variants of the same word have similar characteristic interpretations in most cases. In addition, these morphological variants are considered equivalent in the feature learning of neural networks. Therefore, we use stemming analysis to merge word forms in feature learning. In this way, to a certain extent, the ambiguity generated by the neural network for feature learning of different word forms with the same semantics is eliminated.

Second, we convert the text into a sequence as an input for neural networks. When applying a machine learning model to natural language processing, the first task is to find the correct representation of the word. The vector representation of words is beneficial to capture the semantics of words in different natural language processing tasks. Word vector representation methods are LSA, Word2Vec, and GloVe. Naili et al. applied LSA, Word2Vec, and GloVe to their work [20]. Their comparative experiment found that the three methods depend on the language used, but Word2Vec provides the best word vector representation. Another commonly used word vector representation is One-Hot Encoding. One-Hot Encoding applies a random algorithm, which can easily lead to the loss of information between words. Moreover, it will create an extensive and sparse word vector matrix when it encodes a large amount of text.

Accordingly, we applied the Word2Vec to convert text data into word vectors in our work by considering that Word2Vec solves matrix dimension disaster well by mapping each word to a shorter word vector.

3.2 Model Construction

In the work of Wen et al., the investigation of how the sequence length affects RCNN with Highway Networks indicates that the deep learning model can learn an acceptable representation for long texts [21]. A comparative analysis of CNN and RNN in the work of Adel et al. revealed that CNN is significantly better than RNN when dealing with longer sentences [22]. An empirical study by Akhter et al. shows that CNN is better than LSTM, Bi-LSTM, and CLST in completing long text classification tasks [23]. Illuminated by the above studies, we employ Bi-LSTM to process short-length summary texts and CNN to process long-length descriptions and build a combined model of Bi-LSTM and CNN.

We instantiate the learning task by combining Bi-LSTM with CNN. Considering that the prediction result given by Bi-LSTM for the bug pair is R_{LSTM}, and the prediction result given by CNN for the bug pair is R_{CNN}, we obtain two weaker classifiers through training. After that, we add the weighted sum of the values given by the two classifiers, as shown in Equation (1), and combine the two classifiers through the training of the fully connected network to obtain a more comprehensive and robust model.

$$R = w_1 * R_{LSTM} + w_2 * R_{CNN} + b \quad \textbf{(1)}$$

where R_{LSTM} and R_{CNN} are the prediction result of the new bug pair obtained by the two classifiers, weight w_1 and w_2 are the importance of the two classifiers in the new prediction result, and b is the bias term.

Our hybrid model can be more stable and robust to predict the blocking pairs after training all its hyper-parameters. Taking a new bug b_i for an example, we pair it with all the existing bugs and then use the trained classification model to predict whether there is a blocking relationship between b_i and all other bugs.

3.3 Prediction Report

Our approach employs the trained model to classify the bug pairs, give the prediction results of the relationship between the bugs, and identify whether they are blocking bug pairs with a blocking relationship. In previous work (ELBlocker and Gar.), Precision, Recall, and F1-measure were used to evaluate the model's performance. We refer to their approach and use these three performance indicators and Accuracy indicators to analyze the prediction results of the model to evaluate the performance of our model.

4 Empirical Setup

The primary goal of our work is to predict whether there is a blocking relationship between bugs by applying a hybrid model of Bi-LSTM and CNN. To evaluate the effectiveness and efficiency of the proposed model, we applied it to the bug pairs corresponding to seven open-source projects. In our empirical study, we want to investigate the following research questions:

RQ1: How much improvement can our approach achieve over state-of-the-art approaches?

RQ2: Which deep learning model (Bi-LSTM, CNN or hybrid model) performs better in identifying blocking bugs?

RQ3: What is the computational cost of the building model and predicting block bugs of our approach?

4.1 Experimental Subjects

We obtain a total of 129178 bug reports from seven open-source projects (such as Eclipse[1], Mozilla[2], NetBeans[3], Chromium[4], OpenOffice[5], RedHat[6], Gentoo[7]). These projects are mature, long-standing open-source projects with a large number of bug reports. Six projects (i.e., Eclipse, Mozilla, NetBeans, OpenOffice, RedHat, and Gentoo) use Bugzilla as an issue tracking system. In Bugzilla, there is a "Blocks" field in the bug report. This field is used to display the bug blocked by this bug. Therefore, we use "Blocks" in bug reports to determine whether there is a blocking relationship between bug pairs. Chromium is Google's issue tracking system, and there is a "Blocking" field in its bug reports. This field has the same function as "Blocks" in Bugzilla. We use this field to determine the blocking relationship between bug pairs.

We extracted the summary and description of each bug report as the factors that we used to distinguish between blocking bugs and non-blocking bugs. Table 1 summarizes the characteristics of the seven open-source projects used in our empirical study. To some extent, there are enough bug reports in these open-source projects, which also contain many blocking bugs. The percentage of bugs with blocking relationships ranges from 5.54% to 25.58%, and the total percentage is 7.84%.

Table 1. Statistics of the collected bug reports

Projects	Bugs	Blocking bugs	Non-blocking bugs
Eclipse	56652	4315 (7.62%)	52337 (92.38%)
Mozilla	5832	1492 (25.58%)	4340 (74.42%)
NetBeans	8047	531 (6.60%)	7516 (93.40%)
Chromium	8757	534 (6.10%)	8223 (93.90%)
OpenOffice	9998	554 (5.54%)	9444 (94.46%)
RedHat	29,892	2148 (7.19%)	27744 (92.81%)
Gentoo	10000	555 (5.55%)	9445 (94.45%)
Total	129178	10129 (7.84%)	119049 (92.16%)

4.2 Performance Measures

Our approach will classify the bug pair with the trained model and give the prediction result of the relationship between bugs and identify whether they are the blocking-bug pair with the blocking relationship. We use a confusion matrix to evaluate the effectiveness of our trained model [5]. The confusion matrix stores the correct and incorrect decisions made by the classifier as shown in Table 2.

Table 2. Confusion matrix

		True class	
		Blocking	Non-blocking
Classified as	Blocking	TP	FP
	Non-blocking	FN	TN

We use an example to explain the semantics of each field (i.e., TP, FP, FN, and TN) in Table 2. If a pair of blocking bugs is correctly classified as having a blocking relationship, it is classified as True Positive (TP). If a pair of non-blocking bugs are classified as having a blocking relationship, they are classified as False Positives (FP). If a pair of blocking bugs is classified as a non-blocking relationship, it is classified as a False Negative (FN). Finally, if it is a pair of non-blocking bugs and is correctly classified as a non-blocking relationship, it is a True Negative (TN).

We propose ways to infer distributions of any performance measures computed from the confusion matrix in Table 2. In this paper, we calculate the Accuracy, Precision, Recall, and F1-measure measures for predicting blocking bug pairs to evaluate the performance of the prediction model:

Accuracy refers to the ratio of the number of blocking bugs correctly predicted to the total number of experimental classification bugs. It can be calculated as follows.

$$Accuracy = \frac{TP+TN}{TP+TN+FP+FN} \quad \textbf{(2)}$$

Precision is the ratio of correctly classified blocking bugs over all the bugs classified as blocking. It can be calculated as follows.

[1] http://bugs.eclipse.org/bugs
[2] https://bugzilla.mozilla.org
[3] https://netbeans.apache.org
[4] https://bugs.chromium.org
[5] https://bz.apache.org/ooo
[6] https://bugzilla.redhat.com
[7] https://bugs.gentoo.org/

$$Precision = \frac{TP}{TP+FP} \quad \textbf{(3)}$$

Recall is ratio of correctly classified blocking bugs over all of the actually blocking bugs. It can be calculated as follows.

$$Recall = \frac{TP}{TP+FN} \quad \textbf{(4)}$$

F1-measure measures the weighted harmonic mean of the Precision and Recall. It can be calculated as follows.

$$F1_{measure} = 2 * \frac{Precision*Recall}{Precision+Recall} \quad \textbf{(5)}$$

The higher the Accuracy, the more accurately our model can distinguish between blocking bug pairs and non-blocking bug pairs. Similarly, the higher the value of Precision and Recall, the better the performance of our model, but the two indicators cannot achieve the best at the same time. Combining the above two indicators, a higher F1-measure means that the model is more effective in identifying blocking bugs.

4.3 Implementation Details

We extract the summaries and descriptions from the bug reports and perform the pre-processing of the text through stemming and word vectorization. During the training process, pairs of summaries and descriptions and their associated labelsare fed to the neural networks. The model is iteratively trained to optimize the training loss. For the testing process, new bug pairs are input into the model, and the model gives the prediction results of their relationship. Through these results, various performance measures related to the model are calculated to evaluate its performance.

Experiments were carried out on each project using the proposed model and compared with previous studies (i.e., ELBlocker and Gar.) on several measures (i.e., Precision, Recall, and F1-measure). We run our proposed approach on a machine with Windows 10, 64-bit, Intel(R) Core (TM) i5-9300H CPU @ 2.40GHz and NVIDIA GeForce GTX 1650.

5 Result Analysis

5.1 Result Analysis for RQ1

During the maintenance of software systems, developers usually spend plenty of time and effort on fixing blocking bugs. An automated technique of predicting blocking bugs can detect blocking bugs as early as possible and reduce their threats. For this reason, we hope to construct a new classification prediction model to help developers distinguish between blocking bugs and non-blocking bugs. Recently, Garcia and Shihab used random forests to predict blocking bugs [5]. Xia et al. proposed the ELBlocker, which divides the data set into multiple disjoint sets, trains different classifiers, and combines them to distinguish blocking and non-blocking bugs [9]. Motivated by their work, we use Bi-LSTM and CNN to build weak classifiers and combine them to build a predictive model. Furthermore, we compare our approach with these two baselines.

To train a reliable and stable model, we perform 10-fold cross-validation with 10 repetitions for seven open-source projects, and then calculate the average of the Precision, Recall, and F1-measure of the compared models (i.e., Bi-LSTM+CNN, ELBlocker, and Gar.).

Figure 3 to Figure 5 illustrate the experimental results of our approach compared with the ELBlocker, Garcia and Shihab's approach (denoted as Gar.) on seven open-source projects. Note that: the baseline approaches in the original papers only conducts empirical research on the former five open-source projects. So we only utilized the experimental data of the baseline on five projects from the original papers. The scores of Precisions, Recall and F1-measure are 0.376 to 0.542, 0.376 to 0.680, and 0.375 to 0.589. On average, Bi-LSTM+CNN can achieve Precision, Recall, and F1-measure scores of 0.483, 0.480, and 0.477 on seven projects, respectively. Although in Figure 4, the Recall rate of our approach in OpenOffice is slightly lower than that of the two baselines. But in other cases, it has improved performance compared to other approaches.

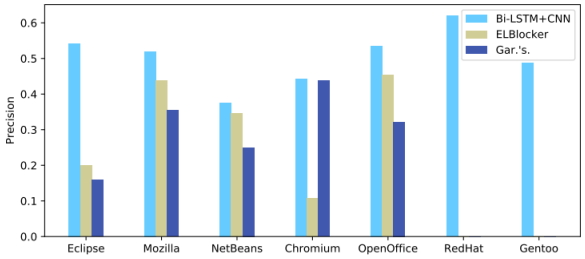


Figure 3. Precision comparison on seven subjects

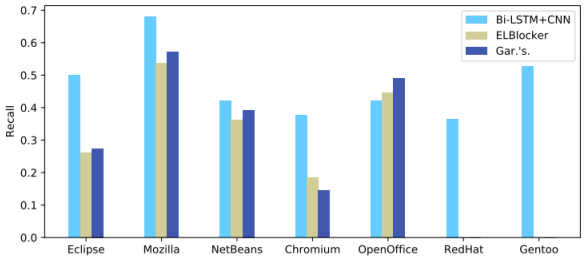


Figure 4. Recall comparison on seven subjects

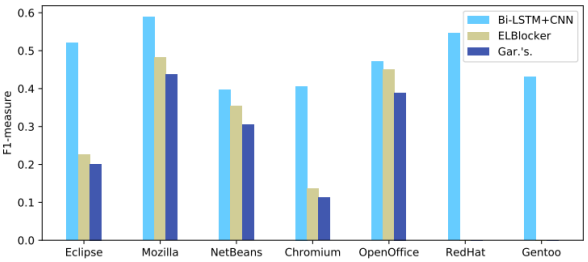


Figure 5. F1-measure comparison on seven subjects

Besides, our approach can achieve 0.512 to 0.612 of Precision, 0.423 to 0.587 of Recall, and 0.482 to 0.564 of F1-measure on the last two projects (i.e., RedHat and Gentoo). In summary, our approach can achieve performance improvement compared with the two baselines (i.e., ELBlocker and Gar.) on Precision and F1-measure.

Furthermore, we applied the paired Wilcoxon signed-rank tests to evaluate the significance of the difference between Bi-LSTM+CNN and compared ones (i.e., ELBlocker and Gar.). Table 3 show the Wilcoxon signed-rank test results of F1-measure between Bi-LSTM+CNN and ELBlocker, Bi-

LSTM+CNN and Gar., respectively. We noticed that our approach outperforms the two baseline approaches. Among them, each p-value is 0.043. The results indicate that our approach has a significant improvement compared with the other two baselines.

Table 3. Wilcoxon test of F1-measure between Bi-LSTM+CNN and ELBlocker, Bi-LSTM+CNN and Gar

	Null Hypothesis	Test	Sig.	Decision
1	The Median of differences between Bi-LSTM+CNN and ELBlocker equals 0.	Related-Samples Wilcoxon Signed Rank Test	0.043	Reject the null hypothesis
2	The median of differences between Bi-LSTM+CNN and Gar. equals 0.	Related-Samples Wilcoxon Signed Rank Test	0.043	Reject the null hypothesis

Summary for RQ1: Our approach performs better than the two compared baselines. On average, our approach achieves 73.53% and 101.44% performance improvements over the ELBlocker and Gar's approach of the F1-measure on the seven open-source projects, respectively. Furthermore, our approach improves performance significantly.

5.2 Result Analysis for RQ2

We built two classifiers and applied two deep learning models (i.e., Bi-LSTM and CNN) to identify the blocking relationship between bug pairs. Since both Bi-LSTM and CNN can predict the blocking relationship to a certain extent, although their performance is different. Now, we use a fully connected neural network (as shown in Equation (1)) to assign two weights to the two classifiers, and then combine them into a hybrid model. Then we use model evaluation measures to compare the effectiveness of a single classifier and a combined classifier. Therefore, we propose a new research question, which is to compare the performance of the Bi-LSTM model, the CNN model, and the proposed hybrid model (Bi-LSTM+CNN). Our goal is to explore which classifier can achieve better prediction performance. We perform 10-fold cross-validation with ten repetitions for each of the seven projects, and then calculate the average of the Accuracy, Precision, Recall, and F1-measure of the three classification models in the two pieces of information. Then we compared the performance of each classifier based on four measures.

Table 4 shows the comparison of Bi-LSTM and CNN on summary. It is not difficult to find from it, although there are situations where CNN is better. But overall, the effect of Bi-LSTM is better than that of CNN.

Table 5 shows the experimental results of Bi-LSTM, CNN and the hybrid model (Bi-LSTM+CNN) in different open-source projects, which uses the summary and description of the bug report to predict the blocking relationship of the bug pair. Although F1-measure can be calculated by Precision and Recall, as shown in Equation (5), we present F1-measure in Table 5 to get a more intuitive and direct explanation.

Table 4. Comparison of Bi-LSTM and CNN on summary

Project	Method	Acc	Pre	Recall	F1
Eclipse	Bi-LSTM	**0.486**	**0.360**	0.038	0.070
	CNN	0.287	0.315	**0.462**	**0.375**
Mozilla	Bi-LSTM	**0.513**	**0.556**	**0.127**	**0.207**
	CNN	0.315	0.412	0.118	0.183
NetBeans	Bi-LSTM	**0.352**	**0.231**	0.125	**0.162**
	CNN	0.276	0.125	**0.151**	0.137
Chromium	Bi-LSTM	**0.368**	**0.397**	0.152	**0.220**
	CNN	0.246	0.159	**0.177**	0.168
OpenOffice	Bi-LSTM	0.495	**0.484**	0.144	0.222
	CNN	**0.527**	0.339	**0.192**	**0.245**
RedHat	Bi-LSTM	**0.353**	**0.268**	**0.429**	**0.330**
	CNN	0.169	0.182	0.177	0.179
Gentoo	Bi-LSTM	**0.485**	**0.357**	0.364	**0.360**
	CNN	0.343	0.251	**0.389**	0.305

Table 5. Performances comparison between Bi-LSTM, CNN and Bi-LSTM+CNN

Project	Method	Acc	Pre	Recall	F1
Eclipse	Bi-LSTM	0.486	0.360	0.038	0.070
	CNN	0.539	0.514	0.462	0.487
	Bi-LSTM+CNN	**0.567**	**0.542**	**0.500**	**0.520**
Mozilla	Bi-LSTM	0.513	**0.556**	0.127	0.207
	CNN	0.524	0.501	0.618	0.553
	Bi-LSTM+CNN	**0.601**	0.520	**0.680**	**0.589**
NetBeans	Bi-LSTM	0.352	0.231	0.125	0.162
	CNN	0.451	0.344	0.367	0.355
	Bi-LSTM+CNN	**0.534**	**0.376**	**0.421**	**0.397**
Chromium	Bi-LSTM	0.368	0.397	0.152	0.220
	CNN	0.462	0.418	0.351	0.382
	Bi-LSTM+CNN	**0.529**	**0.442**	**0.376**	**0.406**
OpenOffice	Bi-LSTM	0.495	0.484	0.144	0.222
	CNN	0.527	0.498	0.394	0.440
	Bi-LSTM+CNN	**0.643**	**0.535**	**0.421**	**0.471**
RedHat	Bi-LSTM	0.353	0.268	0.429	0.330
	CNN	0.412	0.451	**0.481**	0.466
	Bi-LSTM+CNN	**0.536**	**0.620**	0.365	**0.546**
Gentoo	Bi-LSTM	0.485	0.357	0.364	0.360
	CNN	0.526	0.418	0.429	0.423
	Bi-LSTM+CNN	**0.594**	**0.487**	**0.527**	**0.431**

It can be found from Table 5 that in the Mozilla project, the Accuracy of CNN is slightly lower than that of Bi-LSTM. In other cases, the four measures of CNN are better than Bi-LSTM in all projects. On average, the Accuracy and F1-measure of Bi-LSTM are 0.436 and 0.224, respectively. CNN's Accuracy and F1-measure are 0.492 and 0.444, respectively. The hybrid model (i.e. Bi-LSTM+CNN) can outperform CNN and Bi-LSTM in all four measures of the project. The average Accuracy and F1-measure of Bi-LSTM+CNN are 0.572 and 0.480, respectively.

To evaluate the significance of the difference between Bi-LSTM, CNN and Bi-LSTM+CNN, we applied the paired Wilcoxon test between their F1-measure. To show that CNN is more effective than Bi-LSTM and that the hybrid model is improved compared to a single model, we made a one-tailed alternative hypothesis to verify that the F1-measure of CNN is significantly higher than that of Bi-LSTM, and the F1-measure of Bi-LSTM+CNN significantly higher than CNN.

As shown in Table 6, all the p-values are less than 0.05. We can conclude that CNN's performance is significantly better than Bi-LSTM's on seven subjects. Similarly, Bi-LSTM+CNN significantly improves performance than CNN.

Table 6. Wilcoxon test of F1-measure between Bi-LSTM+CNN and CNN, CNN and Bi-LSTM

	Null Hypothesis	Test	p-value.	Decision
1	The Median of differences between Bi-LSTM+CNN and CNN equals 0.	Related-Samples Wilcoxon Signed Rank Test	0.018	Reject the null hypothesis
2	The median of differences between Bi-LSTM and CNN equals 0.	Related-Samples Wilcoxon Signed Rank Test	0.018	Reject the null hypothesis

Summary for RQ2: The results show that, among all the projects, Bi-LSTM+CNN has the best performance, followed by CNN, and then Bi-LSTM. At the same time, the significance test shows that the performance improvement of Bi-LSTM+CNN is significant.

5.3 Result Analysis for RQ3

The efficiency of the prediction model will affect its actual application in real-world software maintenance. Therefore, we investigated the time cost of our approach. This research question reports the size of the input data fed into the neural network, model building time, and prediction time. Specifically, the model building time refers to the time required to pre-process the training data, feed it into the neural network and complete the training to convert it into a model classifier. The prediction time refers to the time for the model classifier that has completed the construction to predict the labels of the new bug reports.

Considering that the experimental platform and application technology are different, we did not compare the building time of our model with these methods. In the future, we aim to reproduce these two baselines and compare them with our proposed approach.

Table 7. Model building time and Prediction time (in seconds)

Project	Sum (MB)	Des (MB)	Model building time	Prediction time
Eclipse	6.11	21.13	2056.64	128.53
Mozilla	1.02	19.76	1945.53	134.25
NetBeans	0.86	10.05	1927.12	123.68
Chromium	1.06	12.48	1933.51	126.53
OpenOffice	0.93	10.94	1966.75	127.59
RedHat	1.01	11.56	1994.79	116.64
Gentoo	2.56	16.82	2165.23	129.31
Average	1.94	14.68	1998.51	126.65

Table 7 shows the size of the training data contained in each of the seven projects, model building time, and prediction time. On average, the summary size for model training is 1.94MB, the description size for model training is 14.68MB, the model building time is 1998.51 seconds, and the prediction time of the model is 126.65 seconds. In general, our model takes about half an hour to complete the pre-processing of the training data and the model's training. It takes about two minutes to predict the blocking relationship.

Summary for RQ3: Although our hybrid model (Bi-LSTM+CNN) takes about half an hour for training, it takes less than two minutes to predict the blocking relationship for new bug pairs using the trained model. This means that our prediction model is satisfactory to a certain extent and can indicate whether multiple bug pairs have a blocking relationship within a few minutes.

6 Related Work

6.1 Bug Report Analysis via Deep Learning

Recently, deep learning models have succeeded in many natural language processing tasks, especially in analyzing software bug reports. These bug reports analyzing work focus on detecting duplicate or similar bugs. For example, an information retrieval and classification-based model using Convolutional Neural Network (CNN) and Long Short-Term Memory (LSTM), proposed by Deshmukh et al., can achieve high accuracy of 90% to detect and retrieve duplicate or similar bugs [24]. Considering that the similarity of bug reports is reflected by some similar characteristics. Kukkar et al. proposed an automatic bug report classifying model based on Convolutional Neural Network (CNN) to extract relevant features [25]. This model identifies duplicate or similar bug reports from the text content available in the bug repository. They conducted experiments on six publicly available large-scale data sets and gave detailed experimental results. Their experimental results show that the accuracy of the proposed system reaches 85% to 99%.

Inspired by these above works, we can use deep learning techniques (such as Bi-LSTM, CNN) to process the text content in bug reports and achieve satisfactory results in bug classification. Taking it a step further, we combine Bi-LSTM with CNN to obtain the characteristics of the bug report and then apply it to predict whether the examining bug is a blocking bug in the dataset.

6.2 Study on Blocking Bugs

A blocking bug is a special software bug that blocks other bugs from being fixed. The study on blocking bugs originated from the earlier work by Garcia and Shihab [5]. They are the ones who first discovered the issues of maintenance aroused by blocking bugs. An empirical study of blocking bugs on six open-source projects found that blocking bugs take approximately two to three times longer to be fixed than non-blocked bugs. From then on, researchers conducted a set of empirical studies to discover the characteristic of blocking bugs, such as the amounts, distributions, and efforts of fixing, etc. [4, 6, 26-27]. For example, Ohira et al. introduced their dataset of high impact bugs which was created by manually reviewing four thousand issue reports in four open-source projects (Ambari, Camel, Derby and Wicket) [28]. The dataset is the first dataset of impact bugs, including security, performance, blocking, and so forth. The latest work by Garcia

and Shihab indicates that blocking bugs require more than 1.2 to 4.7 times of lines of code to be fixed than that of non-blocking bugs [6].

At the same time, researchers focus on identifying blocking bugs from bug reports. Garcia and Shihab also build prediction models based on decision trees to determine whether a bug will be a blocking bug or not. Furthermore, their Top Node analysis indicates that the comments, including the contents of the comment, comment size, the number of developers in the CC list, and the reporter's experience, are the most critical factors to determine blocking bugs [5-6]. They further analyzed that source code files affected by blocking bugs are more negatively impacted in cohesion, coupling complexity, and size than those affected by non-blocking bugs. In more detail, Ren et al. focus on a particular type of blocking bugs which they call Critical Blocking Bugs (CBBs), that block at least two bugs [27]. They study CBBs from five aspects: the importance, the repair time, the scale of repair, the experience of developers who repair CBBs, and the circumstance why CBBs block multiple bugs. They compared CBBs with normal blocking bugs and other bugs on various data sets. The experimental results show that CBBs are more important with longer repair time and larger repair scale, and CBBs are concentrated on parts of components of the project.

Xia et al. extend Garcia and Shihab's work by proposing a novel ensemble learning-based approach named ELBlocker, which combines multiple classifiers built on different subsets of the training set [9]. To examine the benefits of ELBlocker, they perform experiments on six large open-source projects. The ELBlocker can help deal with the class imbalance phenomenon and improve the prediction of blocking bugs. ELBlocker achieves a substantial and statistically significant improvement over the state-of-the-art methods. More specifically, our work is illuminated by ELBlocker. Cheng et al. proposed a new approach based on ensemble learning to distinguish blocking bugs and non-blocking bugs after Xia, called XGBlocker, which includes two stages: feature extraction and model construction [28]. They extracted enhanced features from bug reports and introduced XGBoost advanced algorithms to determine whether the bug was blocked. To test the performance of XGBlocker, they conducted experiments using three evaluation indicators on four open-source projects. The results indicate that XGBlocker has improved further than ELBlocker and other approaches.

6.3 Novelty of Our Study

Unlike the previous works, we only extracted the summary and description from the bug report as the training data for our hybrid model. In addition, we are also the first to apply combined deep learning techniques (Bi-LSTM and CNN) to the prediction of the blocking relationship between reported bugs.

7 Conclusion

This paper proposes a novel blocking bug prediction method, which uses a combination of different deep learning models, such as Bi-LSTM and CNN. The empirical study on seven large open-source projects with 129178 bug reports indicate that the hybrid model can achieve a substantial and statistically significant improvement over the baseline methods (i.e., ELBlocker, Gar.).

In the future, we will obtain more bug reports from more open-source projects and commercial projects as research subjects to improve the generalization ability of the trained model. In addition, we believe that more features in the bug report can help improve model performance. We will select more features into the training set for evaluation in the bug reports to improve model performance because our hybrid model has learned more features. Besides, we will consider using integrated learning methods because constructing predictive models with different features and combining them using integrated learning methods can effectively improve the performance of our model.

Acknowledgements

This work was supported in part by the National Natural Science Foundation of China under Grant 61502497, and Grant 61673384.

References

[1] W. E. Wong, X. Li, P. A. Laplante, Be more familiar with our enemies and pave the way forward: A review of the roles bugs played in software failures, *Journal of Systems and Software*, Vol. 133, pp. 68-94, November, 2017.

[2] W. E. Wong, V. Debroy, A. Surampudi, H. Kim, M. F. Siok, Recent catastrophic accidents: investigating how software was responsible, *In 2010 Fourth International Conference on Secure Software Integration and Reliability Improvement*, Singapore, 2010, pp. 14-22.

[3] V. Debroy, W. E. Wong, Insights on fault interference for programs with multiple bugs, *2009 20th International Symposium on Software Reliability Engineering*, Mysuru, Karnataka, India, 2009, pp. 165-174.

[4] H. Ding, W. Ma, L. Chen, Y. Zhou, B. Xu, Predicting the breakability of blocking bug pairs, *Proceedings of 2018 IEEE 42nd Annual Computer Software and Applications Conference*, Tokyo, Japan, 2018, pp. 219-288.

[5] H. Valdivia Garcia, E. Shihab, Characterizing and predicting blocking bugs in open source projects, *Proceedings of the 11th working conference on mining software repositories*, Hyderabad, India, 2014, pp. 72-81.

[6] H. Valdivia-Garcia, E. Shihab, M. Nagappan, Characterizing and predicting blocking bugs in open source projects, *Journal of Systems and Software*, Vol. 143, pp. 44-58, September, 2018.

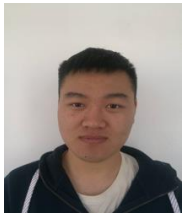
[7] Y. Li, W. E. Wong, S. Y. Lee, F. Wotawa, Using tri-relation networks for effective software fault-proneness prediction, *IEEE Access*, Vol. 7, pp. 63066-63080, May, 2019.

[8] W. E. Wong, J. R. Horgan, M. Syring, W. Zage, D. Zage, Applying design metrics to predict fault-proneness: a case study on a large-scale software system, *Software: Practice and Experience*, Vol. 30, No. 14, pp. 1587-1608, November, 2000.

[9] X. Xia, D. Lo, E. Shihab, X. Wang, X. Yang, Elblocker: Predicting blocking bugs with ensemble imbalance learning, *Information and Software Technology*, Vol. 61, pp. 93-106, May, 2015.

[10] X. Huo, M. Li, Z.-H. Zhou, Learning unified features from natural and programming languages for locating buggy source code, *International Joint Conferences on Artificial Intelligence Organization*, New York, NY, USA, 2016, pp. 1606-1612.
[11] T. Zhang, J. Chen, H. Jiang, X. Luo, X. Xia, Bug report enrichment with application of automated fixer recommendation, *Proceedings of the 25th International Conference on Program Comprehension*, Buenos Aires, Argentina, 2017, pp. 230-240.
[12] T. Zhang, H. Jiang, X. Luo, A. T. Chan, A literature review of research in bug resolution: Tasks, challenges and future directions, *The Computer Journal*, Vol. 59, No. 5, pp. 741-773, May, 2016.
[13] R. Johnson, T. Zhang, Effective use of word order for text categorization with convolutional neural networks, *Proceedings of the 2015 Conference of the North American Chapter of the Association for Computational Linguistics: Human Language Technologies*, Denver, Colorado, USA, 2015, pp. 103-112.
[14] T. Mikolov, M. Karafiát, L. Burget, J. Černockỳ, S. Khudanpur, Recurrent neural network based language model, *Proceedings of the 11th Annual conference of the international speech communica-tion association*, Makuhari, Chiba, Japan, 2010, pp. 1045-1048.
[15] A. Graves, *Supervised Sequence Labelling with Recurrent Neural Networks*, Ph. D. Thesis, Technische Universität München, Germany, 2008.
[16] J. Donahue, L. A. Hendricks, S. Guadarrama, M. Rohrbach, S. Venugopalan, T. Darrell, K. Saenko, Long-term recurrent convolutional networks for visual recognition and description, *Proceedings of 2015 IEEE Conference on Computer Vision and Pattern Recognition*, Boston, MA, USA, 2015, pp. 2625-2634.
[17] S. Hochreiter, J. Schmidhuber, Long short-term memory, *Neural computation*, Vol. 9, No. 8, pp. 1735-1780, November, 1997.
[18] Y. Chen, *Convolutional neural network for sentence classification*, M.S. Thesis, University of Waterloo, Waterloo, Ontario, Canada, 2015.
[19] S. Guo, X. Zhang, X. Yang, R. Chen, C. Guo, H. Li, T. Li, Developer activity motivated bug triaging: via convolutional neural network, *Neural Processing Letter*, Vol. 51, No. 3, pp. 2589-2606, June, 2020.
[20] M. Naili, A. H. Chaibi, H. H. Ben Ghezala, Comparative study of word embedding methods in topic segmentation, *Procedia Computer Science*, Vol. 112, pp. 340-349, 2017.
[21] Y. Wen, W. Zhang, R. Luo, J. Wang, Learning text representation using recurrent convolutional neural network with highway layers, *Neu-IR'16 SIGIR Workshop on Neural Information Retrieval*, Pisa, Italy, pp. 1-5, 2016.
[22] H. Adel, H. Schütze, Exploring different dimensions of attention for uncertainty detection, *Proceedings of the 15th conference of the European chapter of the association for computational linguistics*, Valencia, Spain, 2017, pp. 22-34.
[23] M. P. Akhter, J.-B. Zheng, I. R. Naqvi, M. Abdelmajeed, M. Fayyaz, Exploring deep learning approaches for Urdu text classification in product manufacturing, *Enterprise Information Systems*, Vol 16, No. 2, pp. 223-248, 2022.
[24] J. Deshmukh, K. M. Annervaz, S. Podder, S. Sengupta, N. Dubash, Towards accurate duplicate bug retrieval using deep learning techniques, *Proceedings of 2017 IEEE International Conference on Software Maintenance and Evolution*, Shanghai, China, 2017, pp. 115-124.
[25] A. Kukkar, R. Mohana, Y. Kumar, A. Nayyar, M. Bilal, K.-S. Kwak, Duplicate bug report detection and classification system based on deep learning technique, *IEEE Access*, Vol. 8, pp. 200749-200763, October, 2020.
[26] H. Ren, Y. Li, L. Chen, An empirical study on critical blocking bugs, *Proceedings of the 28th International Conference on Program Comprehension*, Seoul, Republic of Korea, 2020, pp. 72-82.
[27] X. Cheng, N. Liu, L. Guo, Z. Xu, T. Zhang, Blocking Bug Prediction Based on XGBoost with Enhanced Features, *Proceedings of the 44th Annual Computers, Software, and Applications Conference*, Madrid, Spain, 2020, pp. 902-911.
[28] M. Ohira, Y. Kashiwa, Y. Yamatani, H. Yoshiyuki, Y. Maeda, N. Limsettho, K. Fujino, H. Hata, A. Ihara, K. Matsumoto, A dataset of high impact bugs: Manually-classified issue reports, *Proceedings of IEEE/ACM 12th working conference on mining software repositories*, Florence, Italy, 2015, pp. 518-521.

Biographies



Zhihua Chen is studying for his master's degree in computer science at Nantong University. He received his B.S. degree in computer science from Nantong University. His current research interests include software testing and analysis, software measurement, defects prediction.

Xiaolin Ju received Ph.D. degree in computer science from the Chinese University of Mining Technology in 2014. He is now an associate professor with the School of Information Science and Technology, Nantong University. His current research interests are mainly in collective intelligence and software defects analysis.

Haochen Wang received the Ph.D. Degree in Electronics, Informatics and Electric Engineering from University of Pavia in 2018. He is now a lecturer in the School of Information Science and Technology, Nantong University. His current research mainly focuses on Computer Vision, Model Recognition, etc.

Xiang Chen received the M.Sc. and Ph.D. degrees in computer science from Nanjing University, China in 2011. Now he joined the School of Information Science and Technology of Nantong University as an assistant professor. His research interests are mainly in software testing.