This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

DEEPSIM: Deep Semantic Information-Based Automatic Mandelbug Classification

Xiaoting Du, Zheng Zheng, *Senior Member, IEEE*, Guanping Xiao, *Member, IEEE*, Zenghui Zhou, and Kishor S. Trivedi, *Life Fellow, IEEE*

***Abstract*—Understanding and predicting types of bugs are of practical importance for developers to improve the testing efficiency and take appropriate steps to address bugs in software releases. However, due to the complex conditions under which faults manifest and the complexity of the classification rules, the automatic classification of Mandelbugs is a difficult task. In this article, we present a deep semantic information-based Mandelbug classification method that combines a semantic model with a deep learning classifier and makes use of both labeled and unlabeled bug reports. By training the bug report semantic model on millions of bug reports, each word in the text of a bug report is represented as a word embedding that preserves the semantic relationship among the words. Then, a convolutional neural network model is designed to capture the high-level features of bug reports to obtain a more accurate classification. Moreover, the effects of the semantic model size and domain on the classification results are investigated, and the quality of word embeddings is evaluated by analyzing several important parameters.**

***Index Terms*—Aging-related bug (ARB), automatic classification, bug reports, deep learning, Mandelbug.**

I. INTRODUCTION

As SOFTWARE systems are becoming increasingly large and complex, considerable efforts are being directed toward software development and maintenance procedures [1], [2]. It is well known that bugs inevitably appear in all stages of the software lifecycle [3]. Hence, understanding the characteristics of bug types is of practical interest to developers implementing appropriate countermeasures, such as fault tolerance mechanisms or software verification and validation strategies, in current and future software releases [4], [5].

However, Antoniol *et al.* [6] and Herzig *et al.* [7] found that amounts of bug reports were mislabeled in bug tracking systems, which could lead to negative consequences to bug prediction models build on them. Herbold *et al.* [8] confirmed the conclusion obtained by Herzig *et al.* [7] in their work. In order to distinguish actual bugs from nonbugs, researchers have conducted many studies [9]–[14]. For example, in [14], Herbold *et al.* trained a new classification model for both titles and descriptions, and then combined the results of both models. In addition to predicting actual bugs, understanding the specific types of bugs is also an important task. It is known that fixing and removing bugs in the operational phase according to the commits are considerably cost-expensive [15]. However, if a failure cannot be reproduced, it is difficult to diagnose and isolate the potential bugs. On the basis of whether a bug can be consistently manifested under well-defined conditions, researchers have classified bugs into Bohrbugs and Mandelbugs [16]. A Mandelbug does not consistently manifest; under exact conditions, a Mandelbug sometimes (but not always) leads to failure. By contrast, a Bohrbug manifests consistently under a well-defined set of conditions. Accordingly, classification of Mandelbugs is beneficial for understanding their characteristics and, thus, help reproduce them [17], [18].

Unfortunately, the classification of Mandelbugs poses several challenges. Due to the complex fault manifestation conditions of Mandelbugs, users tend to comprehend Mandelbugs in different ways and consequently describe them in various styles in bug reports. For example, consider two MEM bugs (MEM bug, i.e., a type of bug that causes error to accumulate due to improper memory management), for which the summaries are "ext3 memory leaks (size-64 objects)" and "Memory usage doubles after more than 20 hours of uptime." The first reporter directly stated the cause of the error, whereas the second reporter simply described the phenomenon he/she observed for a bug of the same type. Thus, the bug classification procedure is a highly time-consuming process due to the varying styles of their description.

In addition, the complexity of the classification rules presents a major barrier for the classification of Mandelbugs. For example, the subclasses of Mandelbugs include aging-related bugs (ARBs) and nonaging-related Mandelbugs (NAMs), and each of these subclasses can be further classified into several subtypes with their own characteristics and corresponding classification

Manuscript received December 7, 2020; revised April 20, 2021; accepted August 14, 2021. This work was supported in part by the National Natural Science Foundation of China under Grant 61772055, Grant 61872169, and Grant 62002163; in part by the Natural Science Foundation of Jiangsu Province under Grant BK20200441; and in part by the Open Research Fund of State Key Laboratory of Novel Software Technology under Grant KFKT2020B20. Associate Editor: J. Baik. *(Corresponding author: Zheng Zheng.)*

Xiaoting Du, Zheng Zheng, and Zenghui Zhou are with the School of Automation Science and Electrical Engineering, Beihang University, Beijing 100191, China (e-mail: xiaoting_2015@buaa.edu.cn; zhengz@buaa.edu.cn; zhouzenghui@buaa.edu.cn).

Guanping Xiao is with the College of Computer Science and Technology, Nanjing University of Aeronautics and Astronautics, Nanjing 211106, China, and also with the State Key Laboratory of Novel Software Technology, Nanjing University, Nanjing 210023, China (e-mail: gpxiao@nuaa.edu.cn).

Kishor S. Trivedi is with the Department of Electrical and Computer Engineering, Duke University, Durham, NC 27708 USA (e-mail: ktrivedi@duke.edu).

Color versions of one or more figures in this article are available at https://doi.org/10.1109/TR.2021.3110096.

Digital Object Identifier 10.1109/TR.2021.3110096

0018-9529 © 2021 IEEE. Personal use is permitted, but republication/redistribution requires IEEE permission.
See https://www.ieee.org/publications/rights/index.html for more information.

Authorized licensed use limited to: BEIHANG UNIVERSITY. Downloaded on November 24,2021 at 06:03:13 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

rules [19]. Nevertheless, it is difficult for developers to accurately classify the bug type without fully understanding the user's description in the bug report. For example, consider the following summary of a bug report: "Framebuffer driver doesn't load on systems with >1 G memor (radeon, nvidia)." This report contains the keyword "memor," which can mislead developers to classifying the bug as a memory-related bug. However, further analysis of the semantics of the entire sentence reveals that the report actually indicates that the failure is attributable to an external condition. Therefore, the contextual semantics contained in each bug report is crucial for the automatic classification of Mandelbugs.

A. Limitations and Insights

In summary, it is essential to learn the semantic information in bug reports before classifying a Mandelbug. However, existing automatic bug report classification methods in this field ignore the semantic information in bug reports [20], [21]. The main idea of these methods is to use the bag-of-words model [22] to represent a bug report and to classify bug reports by machine learning classifiers. However, the most significant features of the bag-of-words model are that it ignores the contextual semantic information contained in bug reports and uses the frequency of words in bug reports as the classification feature. As a result, existing methods may have limitations for the automatic classification of Mandelbugs. In our previous work [23], we considered semantic information, and several traditional machine learning classifiers were used to classify bug reports. However, a common limitation of traditional machine learning classifiers is that they rely heavily on inputs and cannot learn features by themselves.

B. Our Solution

To address the aforementioned issues, we propose a D*eep* Semantic Information-based Mandelbug classification method (DEEPSIM) based on a skip-gram semantic model [24], [25]. Unlike the bag-of-words model, this model utilizes the semantic features contained in bug reports by converting words and phrases into word embeddings. After training the bug report semantic model, a word embedding representation of each word in each bug report is obtained. These word embeddings are then used to represent each bug report as a two-dimensional matrix. However, there are still correlated semantic relations among different word embeddings in the matrix. In our method, we embed the state-of-the-art convolutional neural network (CNN) deep learning model as the classifier. Different from traditional machine learning classifiers, a strong advantage of deep learning models is their feature learning ability, i.e., deep learning models can capture the higher level features of bug reports from word embeddings and improve the classification accuracy. Finally, the effectiveness of DEEPSIM is evaluated on 6557 bug reports from four different open-source software projects [19], [26].

In summary, this article makes the following main contributions.

1) We present DEEPSIM, a new Mandelbug classification method that combines a semantic model and a deep learning classifier to improve the Mandelbug classification performance.
2) We extract the text information within bug reports and train a bug report semantic model by learning the semantic information from 2 038 675 bug reports.
3) We design a CNN model to classify Mandelbugs from a multigranular perspective and compare it with existing work. The results show that our method performs much better than the methods developed in previous work.
4) We further explore the factors that affect the automatic classification results, including the semantic model size and domain, and the parameters that influence the quality of word embeddings.

C. Organization

The remainder of this article is organized as follows. Section II describes our DEEPSIM method. Section III presents the experimental setup. Section IV discusses and analyzes the experimental results, and Section V identifies the main threats to validity. Section VI introduces related work. Finally, Section VII concludes this article.

II. Our Approach

In this section, we propose our DEEPSIM method. Fig. 1 shows the framework of DEEPSIM, which contains two parts: data preparation and classification model training. In the data preparation part, we first train the bug report semantic model and then represent each bug report as a two-dimensional matrix based on the word embeddings obtained by training the bug report semantic model. Then, considering the impacts of the class imbalance between the different bug types, a class imbalance mitigation approach is designed and embedded into the method. In the classification model training part, we construct a bug report classifier based on a CNN model to predict the type of bugs. Finally, multigranular classification of bug reports is performed. In the rest part of this section, we introduce our method in detail.

A. Bug Report Semantic Model Training

A word embedding is a real-valued vector representation of words by embedding both semantic and syntactic meanings obtained from large unlabeled corpus [27]. The word embedding technique is widely used for natural language processing tasks, such as text classification [28], question answering [29], and machine translation [30]. In recent years, multiple models for training word embeddings have been proposed [24], [31]–[33]. Word2vec is the most popular model [24], [25] and has achieved state-of-the-art results on a range of linguistic tasks [34]. The basis of word2vec is that words with similar meanings in a given context exhibit close distances in the vector space. By means of word2vec, words and sentences can be converted into distributed vector representations (i.e., word embeddings). In this study, semantic model training is carried out by using the skip-gram model architecture of word2vec that has been used in several software engineering tasks and has achieved impressive

Authorized licensed use limited to: BEIHANG UNIVERSITY. Downloaded on November 24,2021 at 06:03:13 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

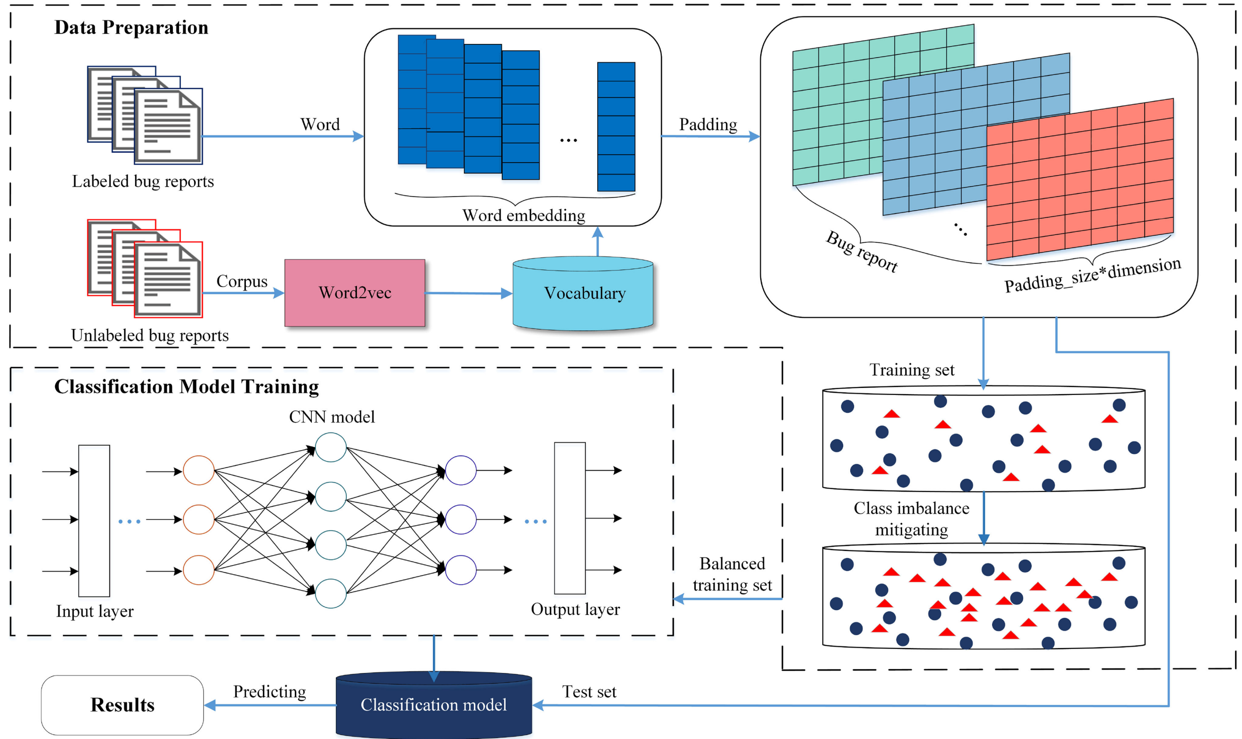

Fig. 1. Detailed structure of DEEPSIM.

performance [35], [36]. This model uses deep neural networks to learn semantic information from the context of a corpus, thereby producing a low-dimensional vector representation of each word.

After collecting bug reports and extracting the text information contained in these bug reports, we obtain a corpus of texts. Then, we apply word2vec [37] with the skip-gram model on the collected corpus to train the bug report semantic model. The training objective of the skip-gram model is to predict the word embedding representation of each word in its given context based on the word embedding of the current word. More precisely, the current word is used to predict the words in a range before and after the current word. Using the skip-gram model, we can convert words into distributed vectors, and the distances among these vectors can be used to represent the semantic similarity of words. For example, consider the following two phrases: "...resource leak in file linux..." and "...memory leak in linux...." In these two phrases, "resource leak" and "memory leak" represent similar bug manifestations. Thus, "resource" and "memory" are similar words. When mapped into the vector space, the positions of the word embeddings for these two words will be close to each other.

More specifically, given a word w_t, the context of w_t is represented by Context_{w_t}. Consider a Bohrbug as an example, where the summary text of the bug report is "serio driver inconsistent use of names." Fig. 2 shows the process of training word embeddings using the skip-gram model when the current word is w_t = inconsistent. When w_t = inconsistent is mapped into the vector space, its corresponding word embedding is denoted by v_{w_t}. Thus, v_{w_t} can be used to predict the word

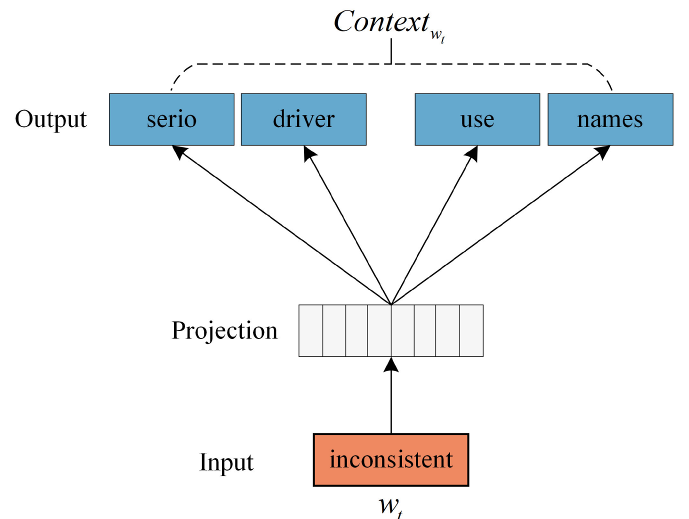

Fig. 2. Sketch of the skip-gram model architecture.

embeddings for the C words to the left and the C words to the right within its context. For example, let w_t = inconsistent, and suppose that $C = 2$. Then, the context of w_t = inconsistent is $\text{Context}_{w_t} = \{\text{serio, driver, use, names}\}$. During the training process, the objective function f is maximized by optimizing the word embedding v_{w_t} and the parameters in the neural network model. The objective function f is

$$f = \frac{1}{T}\sum_{t=1}^{T} \sum_{w_c \epsilon \text{Context}_{w_t}} \log p(w_c|w_t) \quad (1)$$

Authorized licensed use limited to: BEIHANG UNIVERSITY. Downloaded on November 24,2021 at 06:03:13 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

where w_c denotes a word in the context of the word w_t and T represents the total length of the word sequence (for example, $T = 5$). In this equation, $p(w_c|w_t)$ is the expression for a multiclass classifier, i.e., *softmax*, that is used to predict the word embedding representations. The calculation of $p(w_c|w_t)$ is defined as

$$p(w_c \epsilon \text{Context}_{w_t}|w_t) = \frac{\exp\{v_{w_c}^T \cdot v_{w_t}\}}{\sum_{j=1}^{W} \exp\{v_{w_j}^T \cdot v_{w_t}\}} \tag{2}$$

where v_{w_t} is the word embedding of the current word w_t, v_{w_c} is the word embedding of the context word w_c, and W is the length of the vocabulary consisting of all words. The objective function is designed based on the hypothesis that words that appear in similar contexts have similar meanings [38]. During the training of the semantic model, if w_t and w_c form a good word-context pair, the objective function attempts to increase the value of $v_{w_c}^T \cdot v_{w_t}$, whereas if w_t and w_c form a poor word-context pair, the objective function attempts to decrease the value of $v_{w_c}^T \cdot v_{w_t}$. This also means that words sharing many contexts will be more similar to each other.

B. Bug Report Representation

By training the bug report semantic model, a vocabulary is obtained that consists of each word in the corpus and its corresponding word embedding representation. By referencing this vocabulary, we can obtain the word embedding of each word in bug reports. Suppose there is a bug report consisting of m words $\text{word}_1, \text{word}_2, \ldots, \text{word}_m$, and assume each word is replaced by a corresponding n-dimensional pretrained word embedding. Then, the bug report can be represented as a two-dimensional matrix denoted $A \in \mathbb{R}^{m \times n}$. Since the input of the deep learning model requires a fixed shape and the length of the bug report (i.e., the number of words contained in a bug report) is different, we adjust the dimension of the matrix by padding [39] and set the padding size p to the maximum number of words contained in the bug report. For a report where the word length is not equal to p, zero padding is used to increase its length. After this step, we obtain the representation of each bug report $A' \in \mathbb{R}^{p \times n}$.

C. Class Imbalance Mitigation

A strong problem of class imbalance among the different types of bugs is present in bug report datasets. For example, the number of Mandelbugs in software systems is usually much fewer than that of Bohrbugs. One reason for this is that during the development of software systems, a large number of Bohrbugs are introduced. Another reason is that the activation and/or error propagation conditions of Mandelbugs are more complex than those of Bohrbugs, making the former more difficult for users to discover. According to previous studies [26], the classes of Bohrbugs and Mandelbugs are not balanced. For example, the proportions of Bohrbugs and Mandelbugs in the Linux kernel are 55.82% and 36.34%, respectively. In addition, for ARBs and NAMs, i.e., the subtypes of Mandelbugs, the class imbalance problem may be worse. The proportion of NAMs is approximately seven times larger than that of ARBs in the Linux kernel. To mitigate the impact of imbalanced data on the classification accuracy, the synthetic minority oversampling technique (SMOTE) is used in this article [40], [41].

The main idea of SMOTE is to oversample the instances in the minority class by creating synthetic examples rather than by replacement. Suppose that there are several instances from the minority class; in this case, SMOTE synthesizes new minority class instances by interpolating between these instances. The interpolation method selects instances close to each other, draws a line between the instances, and then inserts a new instance along the line. In this article, the SMOTE algorithm is used to guarantee that the number of the minority class samples is equal to that of the majority class samples. And the implementation of SMOTE is provided by the imbalanced-learn Python library [42]. Taking the class imbalance problem between Bohrbugs and Mandelbugs as an example. Suppose there are p Bohrbugs denoted as $\{b_1, b_2, \ldots, b_p\}$ and q Mandelbugs denoted as $\{m_1, m_2, \ldots, m_q\}$. Mandelbug is the minority class of these two bug types (i.e., $p > q$). Then, the total amount of oversampling $p - q$ is set up to obtain an approximately 1:1 class distribution. First, a random instance m_i is selected from the Mandelbugs, and K of its nearest neighbors are found $\{y_1, y_2, \ldots, y_k\}$. Then, synthetic Mandelbugs are created by interpolating between m_i and y_j, where j is $1, 2, \ldots, k$. For this purpose, we first calculate the difference between m_i and y_j, multiply the difference by a random number between 0 and 1, and finally add the result to m_i. The procedure is formulated as

$$m_{\text{new}} = m_i + (y_j - m_i) \times \delta \tag{3}$$

where m_i represents the matrix representation of the Mandelbug under consideration, y_j represents one of the K-nearest neighbors for m_i, m_{new} represents the new synthetic Mandelbug, and δ is a random number in [0,1].

D. CNN Classifier

One of the main advantages of deep learning is feature learning, which exploits the property that many natural signals are compositional hierarchies, i.e., the higher level features of the hierarchy are composed of lower level features [43]. After obtaining the word embedding representation of each word by training the bug report semantic model (see Section II-A), each bug report is represented as a two-dimensional matrix (see Section II-B). However, highly correlated semantic relations remain among the different vectors in the matrix. To capture the higher level features of bug reports from these vectors and obtain a more accurate classification, a deep learning model (i.e., CNN) is designed in this section. Fig. 3 presents the overall workflow of our CNN model. The first two layers of the model are convolutional layers that are used to extract the features. Each of these layers is followed by a tan$h$ activation function that is used to add nonlinear factors. Then, a pooling layer and a dropout layer are sequentially added. The last layer is a flatten layer that combines all of the features and outputs the value to a softmax layer for classification.

The function of each convolutional layer is to extract higher level features from the input matrix obtained in the data preparation part of DEEPSIM. Suppose there is a filter f, the length of

Authorized licensed use limited to: BEIHANG UNIVERSITY. Downloaded on November 24,2021 at 06:03:13 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

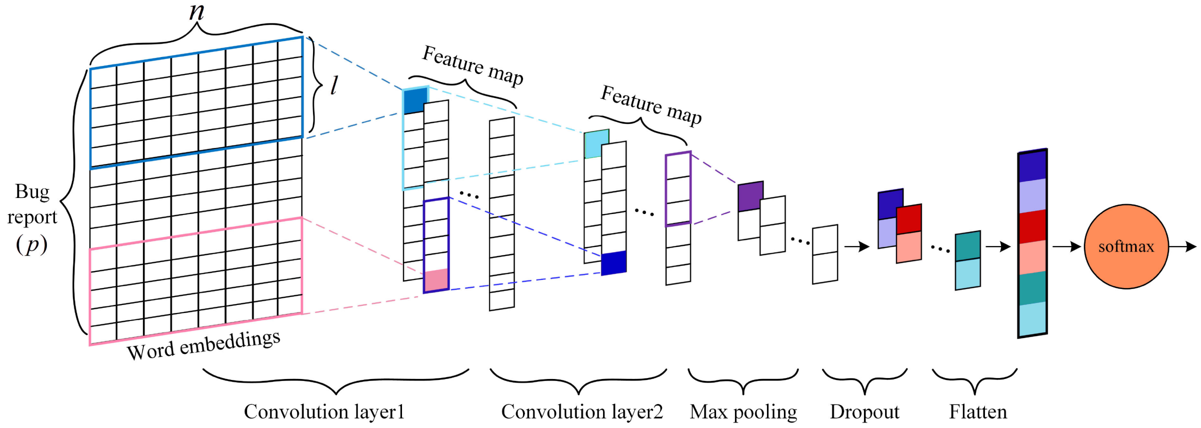

Fig. 3. Workflow of our CNN model.

f is l, and f is parameterized with a weight matrix $\mathbf{w}$. Since each row of the input matrix represents a word, in the first convolutional layer, the width of the filter is equal to n, which denotes the dimension of the word embedding. Given a bug report $A' \in \mathbb{R}^{p\times n}$, a submatrix $A'[j : j + l - 1]$ is applied to all possible windows of words in A'. By repeatedly applying the filter to A', we can obtain the output $o \in \mathbb{R}^{p-l+1}$, and the formula is given as follows:

$$o_j = \mathbf{w} \cdot A' [j : j + l - 1] \tag{4}$$

where j is $1, 2, \ldots, p - l + 1$. To extract more nonlinear features, a bias term $b \in \mathbb{R}$ and an activation function tan$h$ are added. As a result, the feature map $m \in \mathbb{R}^{p-l+1}$ for this filter is obtained

$$m_j = \tanh(o_j + b). \tag{5}$$

To further extract the association features in bug reports, we add a second convolutional layer. Then, the function of the pooling layer is to further extract the features generated from the convolutional layers by aggregating the scores for each filter. In our CNN model, we apply a max-over-time pooling operation to each feature map. The idea is to choose the highest value in each dimension of the vector to capture the most important feature. Then, dropout is applied, which is a regularizer that is used to prevent overfitting. Dropout is a regularization method that stochastically sets the activation of the hidden units for each training case to zero at the training time. Finally, a flatten layer is added to form a vector representation as the input to the softmax layer for classification.

E. Multigranular Classification of Bug Reports

To understand the features and characteristics of bugs in software systems, researchers have attempted to analyze the factors that trigger a fault and/or propagate a fault into a failure. Generally, fault triggers are complex and not only include the timing of inputs and operations but also involve the interactions with other systems, leading to failures that are very difficult to reproduce and requiring the use of specific strategies, such as fault tolerance strategies that mask faults for their handling [44]. Based on the complexity of fault activation and/or error propagation conditions, Grottke and Trivedi [16] divided bugs into two categories: Bohrbugs and Mandelbugs. The definitions of Bohrbugs and Mandelbugs are given as follows.

Borhbug: A bug that can be consistently manifested under well-defined conditions. The activation and/or error propagation of Bohrbugs is simple.

Mandelbug: A bug that cannot always be manifested even under exact conditions. In contrast to Bohrbugs, the activation and/or error propagation of Mandelbugs is complex.

The complexity of the triggering conditions may be caused by the following: there is a time lag between the activation of the fault and the occurrence of failure; some indirect factors may play a role, such as the interaction of the software application with the internal environment of the system; the timing of the inputs and operations; and the relative order of inputs and operations. Furthermore, according to whether a Mandelbug will cause a software aging phenomenon (i.e., an increase in the failure rate and/or performance degradation over time), Mandelbugs are further classified into ARBs and NAMs [16]. Based on the various causes that give rise to the complexity of fault triggering conditions [19], NAMs can be further classified into TIMs, ENVs, LAGs, and SEQs. Furthermore, according to the underlying causes of the software aging phenomenon, ARBs can be classified into LOGs, MEMs, NUMs, STOs, and TOTs. The definitions of these ARB and NAM subtypes are shown in Table I.

Based on the aforementioned classification criterion, to confirm the category of each submitted bug report, in this article, we automatically classify bug reports at four granularities. As depicted in Fig. 4, prior to classifying the bugs based on fault triggers, we first classify the collected bug reports into actual bugs and nonbugs (i.e., bug reports that do not contain actual bug descriptions). As reported in [6], not all bug reports contain actual bugs, and therefore, bug reports that do not contain an actual bug should be filtered out, i.e., requests for new features or enhancements, documentation issues (e.g., missing information, outdated documentation, or harmless warning outputs), compile-time issues (e.g., cmake errors or linking errors), operator errors, and duplicate bug reports. Once these nonbug

Authorized licensed use limited to: BEIHANG UNIVERSITY. Downloaded on November 24,2021 at 06:03:13 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

TABLE I
DEFINITIONS OF THE SUBTYPES OF ARBS AND NAMS

Subtypes of ARBs [19]	
LOG	A type of ARB that causes other logical resources to leak.
MEM	A type of ARB that causes errors to accumulate due to improper memory management.
NUM	A type of ARB that causes the accumulation of numerical errors.
STO	A type of ARB that causes errors to accumulate due to improper management of storage space.
TOT	A type of ARB for which the failure activation or error propagation rate increases as the system runs but is not caused by the accumulation of the internal error states.
Subtypes of NAMs [19]	
TIM	A type of NAM for which fault activation and/or error propagation is influenced by the timing of the inputs and operations.
ENV	A type of NAM for which fault activation and/or error propagation is related to the interaction between the software and the internal environment of the system.
LAG	A type of NAM for which there is a time lag between fault activation and the occurrence of failure.
SEQ	A type of NAM for which fault activation and/or error propagation is influenced by the order of the inputs and operations.

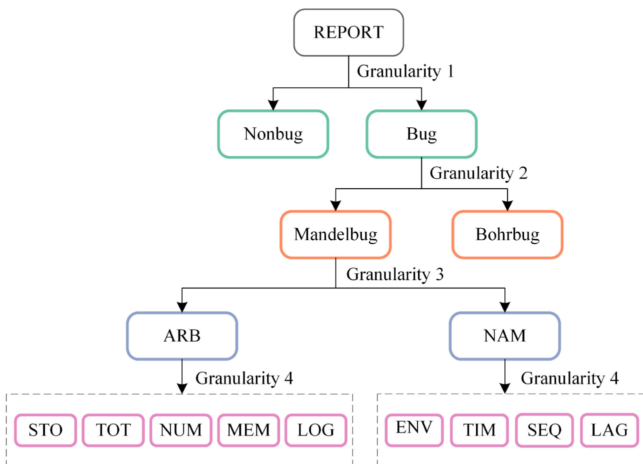

Fig. 4. Multigranular classification of bug reports.

reports are excluded, all other reports are considered to contain actual bugs. Then, considering the fault triggers, we classify the actual bugs into Bohrbugs and Mandelbugs in the second granularity. For the third granularity, Mandelbugs are further divided into two subtypes, i.e., NAMs and ARBs. Finally, we further divide ARBs and NAMs into their respective subtypes. This multigranular classification of bug reports requires our method to perform well in both the two-category (e.g., classification of Bohrbugs/Mandelbugs) and the multicategory (e.g., classification of ARB/NAM subtypes) classification tasks.

TABLE II
CORPUS USED FOR TRAINING THE SEMANTIC MODEL

Project	Time Frame	# of Reports
Eclipse	10/10/01 – 09/30/18	528,862
Freedesktop	01/09/03 – 09/30/18	106,065
GCC	08/03/99 – 09/30/18	81,463
GNOME	02/05/99 – 09/30/18	673,301
KDE	01/21/99 – 09/30/18	388,711
LibreOffice	08/03/10 – 09/30/18	62,029
Linux	11/06/02 – 09/30/18	32,340
LLVM	10/07/03 – 09/30/18	38,107
OpenOffice	10/16/00 – 09/30/18	127,797
	Total	2,038,675

TABLE III
DATASETS FOR CNN MODEL TRAINING AND TESTING

Project	Time Frame	# of Reports
Linux Data1	11/02 – 11/16	5,741
Linux Data2	07/03 – 05/11	267
MySQL	08/06 – 02/11	209
HTTPD	03/02 – 10/07	141
AXIS	07/01 – 11/05	199
	Total	6,557

III. EXPERIMENTAL SETUP

To verify the effectiveness of the proposed DEEPSIM method, we perform the automatic classification on datasets collected from four open-source projects. The performance of DEEPSIM is evaluated through two of the most frequently used measures in the studies [45]–[47], i.e., accuracy and F-measure. Finally, comparative experiments are conducted by comparing DEEPSIM with the existing methods [20], [23]. Detailed introductions are presented in the following.

A. Data Collection

Two parts of datasets are used in DEEPSIM. The first consists of unlabeled bug reports used for training the bug report semantic model, whereas the second consists of labeled bug reports that are used to train and test the CNN model.

1) Data for Training the Bug Report Semantic Model: We collect 2 038 675 bug reports from a bug tracking system (i.e., Bugzilla) to train the bug report semantic model and the summary, description, and comments of each bug report are extracted. The bug reports are distributed among nine projects, and detailed information on the collected bug reports is provided in Table II. The process of semantic model training is unsupervised and bug reports in this part are unlabeled.

2) Data for Training and Testing the CNN Model: Table III shows the datasets used for training and testing the CNN model. Among them, Linux data1 is a dataset obtained from our previous work [26] that contains 5741 bug reports for the Linux kernel. In [26], the classification of these bug reports is done manually by examining the detailed information contained in

Authorized licensed use limited to: BEIHANG UNIVERSITY. Downloaded on November 24,2021 at 06:03:13 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

bug reports. The information in bug reports includes the descriptions of bugs submitted by reporters, comments, additional files (for example, patches for correcting bugs) and external links provided for further information, such as Git commit IDs. The authors carefully cross-checked the results to reduce possible misclassification and inspection mistakes. The other four datasets are taken from Cotroneo *et al.* [19] and include 267 bug reports of Linux system, 209 bug reports of MySQL, 141 bug reports of HTTPD, and 199 bug reports of AXIS.

We publicly provide these two dataset parts at https://github.com/xiaotingdu/DeepSIM.

B. Data Preprocessing

For each bug report, we preprocess its texts (i.e., summary, description, and comments) in three steps: word tokenization, stop-word removal, and lemmatization.

Word tokenization is the most basic step of text preprocessing. The text is first divided into a stream of words or other meaningful elements. We then remove all numbers and punctuation marks that appear in the text and replace other nonalphabetic characters, such as "#," "*," and "&," with spaces. Then, the remaining words are extracted from the bug report text. For example, given the report "Kernel hangs when APIC/ACPI enabled," after word tokenization, we obtain ["kernel," "hangs," "when," "apic," "acpi," "enabled"].

Stop-word removal accounts for the fact that many of the most frequently used words in English are useless for information retrieval and text mining. Such words are called stop words, i.e., words that are frequently used but contain no information regardless of the bug type, such as "and," "the," and "to." For example, for the description "Changing the CPU frequency using CPUFREQ hangs the kernel," the word "the" is removed in this step. The removal of stop words is an important step in text preprocessing for eliminating unimportant information [48].

Lemmatization is the process of finding the canonical form of a word. The results obtained after lemmatization are meaningful and valid words in the vocabulary. For example, the word "aliases" in the bug description "broken module aliases in ieee1394 drivers" is transformed into its standard form "alias" after lemmatization. This operation is also used in text mining and many other linguistic fields [49]. All of these operations are conducted by employing the natural language toolkit [50].

C. Evaluation Metrics

In our experiments, we use the train_test_split function in sklearn to randomly split the labeled bug reports used for evaluating our CNN model: 80% of the bug reports are used as the training set and 20% are used as the test set. This whole process is repeated 1000 times, and the average results across all 1000 iterations are recorded. We use accuracy and F-measure to evaluate the performance of DEEPSIM. Among them, accuracy is calculated as the proportion of correctly classified bug reports relative to the total number of bug reports [51]. It can be used to access the predictive power of an algorithm. In addition to accuracy, F-measure is another popular measure [52]. It gives a good overall picture of predictive performance. The higher the F-measure is, the better the prediction performance represents.

In the following, we briefly introduce their definitions. We note that TP denotes true positive, FP denotes false positive, FN denotes false negative, and TN denotes true negative.

Accuracy: Accuracy is calculated as the proportion of the instances correctly predicted relative to all instances and is defined as

$$\text{Accuracy} = \frac{\text{TN} + \text{TP}}{\text{TP} + \text{TN} + \text{FP} + \text{FN}}. \tag{6}$$

F-measure: The F-measure is a composite metric that combines both precision and recall. It can be used to evaluate whether an increase in precision outweighs a reduction in recall. This metric is defined as

$$\text{F-measure} = \frac{\text{TP}}{\text{TP} + 0.5 * (\text{FP} + \text{FN})}. \tag{7}$$

D. Comparison Methods

In this section, we introduce the methods used for the comparison with the proposed method. In our previous work [23], seven bug report classification methods were proposed based on traditional machine learning classifiers to classify bug reports at four granularities. In Xia's work [20], a method named USES^B was proposed to classify bugs into Bohrbugs and Mandelbugs. We briefly describe each of these methods below.

1) Logistic Regression (LR) Based Method: This method classifies bug reports based on an LR classifier. An LR classifier uses linear models to perform the classification and has been used for text classification for many years. Compared with other linear classification methods, this method exhibits attractive classification performance [53].

2) Stochastic Gradient Descent (SGD) Based Method: This method classifies bug reports based on the SGD classifier. SGD is a very simple and effective method for fitting linear models [53], such as the support vector machine (SVM) and LR models. The model fitted here is an SVM model. Such models usually have good performance when dealing with large data volumes and large quantities of features.

3) Gaussian Naive Bayes (GNB) Based Method: This method classifies bug reports based on the GNB classifier. Naive Bayes algorithms are supervised learning algorithms [53]. Based on Bayes' theorem and the assumption of strong independence between each pair of features, the probability that a given instance belongs to a certain category is calculated.

4) Decision Tree Classifier (DTC) Based Method: This method classifies bug reports based on the DTC. The goal of a decision tree is to establish a model for predicting the values of the target variables by learning simple decision rules derived from the data characteristics [53].

5) Linear Discriminant Analysis (LDA) Based Method: This method classifies bug reports based on the LDA classifier. LDA offers closed-form solutions that can be easily calculated [53]. This method maximizes the ratio of the between-class variance to the within-class variance in any given dataset to ensure maximum separability and reasonable classification accuracy.

6) Random Forest Classifier (RFC) Based Method: This method classifies bug reports based on the RFC. The random forest approach is an integrated learning method based on random decision trees [53]. This method integrates multiple trees

Authorized licensed use limited to: BEIHANG UNIVERSITY. Downloaded on November 24,2021 at 06:03:13 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

TABLE IV
RESULTS OF BUG/NONBUG CLASSIFICATION ON LINUX DATA1

Method	Accuracy	F-measure
SGD	0.773	0.774
LR	0.832	0.842
GNB	0.740	0.729
GBC	0.829	0.848
DTC	0.718	0.715
RFC	0.791	0.802
LDA	**0.834**	**0.848**
DEEPSIM	**0.897**	**0.897**
Impro.	**7.55%**	**5.79%**

TABLE V
RESULTS OF BOHRBUG/MANDELBUG CLASSIFICATION ON LINUX DATA1

Method	Accuracy	F-measure
SGD	0.621	0.622
LR	0.682	0.688
GNB	0.663	0.661
GBC	**0.691**	**0.703**
DTC	0.590	0.589
RFC	0.629	0.629
LDA	0.677	0.684
DEEPSIM	**0.751**	**0.752**
Impro.	**8.68%**	**6.97%**

TABLE VI
COMPARISON WITH THE CLASSIFICATION RESULTS OF USES^B [20]

Metric	Method	HTTPD	Linux2	MySQL	AXIS	Avg.
F-measure Mandelbug	USES^B	0.375	0.524	0.615	0.298	0.453
	DEEPSIM	0.953	0.618	0.729	0.979	0.820
	Impro.	**154.13%**	**17.94%**	**18.54%**	**228.52%**	**81.02%**
F-measure Bohrbug	USES^B	0.872	0.587	0.758	0.906	0.781
	DEEPSIM	0.951	0.651	0.707	0.979	0.822
	Impro.	**9.06%**	**10.90%**	-6.73%	**8.06%**	**5.25%**
Accuracy	USES^B	0.787	0.558	0.703	0.834	0.721
	DEEPSIM	0.953	0.638	0.720	0.979	0.823
	Impro.	**21.09%**	**14.34%**	**2.42%**	**17.39%**	**14.15%**

based on ensemble learning and provides information about the importance of each variable for the classification task.

7) Gradient Boosting Classifier (GBC) Based Method: The method classifies bug reports based on the GBC. Gradient tree boosting is an ensemble algorithm motivated by the possibility of combining several weak models to produce a more powerful ensemble [53]. This method constructs models in a phased manner and generalizes them by permitting the optimization of arbitrary differential loss functions.

8) USES^B: This method was proposed by Xia *et al.* [20] to automatically classify bug reports into Bohrbugs and Mandelbugs, i.e., the second granular classification in this article. USES^B takes the appearance of word tokens as the feature of bug reports. If a word token appears in a bug report, it is labeled 1; otherwise, it is labeled 0. To reduce the number of features, the authors proposed a fuzzy set-based feature selection algorithm named USES. Then, they used the naive Bayes multinomial approach as the classifier after they leveraged USES to select the words and denoted the combination of the multinomial naive Bayes approach and USES as USES^B. To compare DEEPSIM with USES^B, we use the same datasets as [20], i.e., Linux data2, MySQL, HTTPD, and AXIS, as shown in Table III.

IV. EVALUATION AND ANALYSIS

A. RQ1: How Does DEEPSIM Perform in Mandelbug Classification?

This section presents the bug report classification results at four granularities and evaluates our method through a comparison with the eight methods described in Section III-D.

1) Granularity 1—Bugs and Nonbugs: In this section, we analyze the experimental results at the first granularity, i.e., classifying all bug reports into actual bugs and nonbugs. Table IV shows the automatic bug/nonbug classification results obtained by DEEPSIM and seven comparison methods that we have used in our previous work [23]. From Table IV, we observe that compared with the traditional machine learning classifiers, the performance of DEEPSIM on both accuracy and the F-measure is improved. Among seven traditional machine learning classifiers, LDA performs best in the task of classifying bug reports into bugs and nonbugs. The classification accuracy obtained by DEEPSIM is 0.897, which constitutes an improvement of 7.55% over the accuracy obtained by LDA classifier (i.e., 0.834). In addition, the F-measure obtained by DEEPSIM is 0.897, 5.79% higher than the F-measure obtained by the LDA classifier (i.e., 0.848).

2) Granularity 2—Bohrbugs and Mandelbugs: In this section, we analyze the experimental results at the second granularity, i.e., the classification of actual bugs into Bohrbugs and Mandelbugs. The results are presented in Table V. According to the results, the accuracy and F-measure obtained by DEEPSIM are 0.751 and 0.752, respectively. Additionally, the GBC classifier performs best among all the seven traditional machine learning classifiers. The accuracy and F-measure obtained by GBC are 0.691 and 0.703, respectively. As a result, the improvements in accuracy and the F-measure by DEEPSIM are 8.68% and 6.97%, respectively.

To further evaluate our DEEPSIM method, we use the same datasets (i.e., Linux data2, MySQL, HTTPD, and AXIS) utilized for USES^B [20] described in Section III-D. Table VI presents the comparison results. On average, DEEPSIM improves the Mandelbug F-measure, Bohrbug F-measure, and accuracy by 81.02%, 5.25%, and 14.15%, respectively, compared to USES^B. In particular, the classification results on projects HTTPD and AXIS are significantly improved. Specifically, the F-measure values achieved by DEEPSIM for Mandelbugs in HTTPD and AXIS are improved by 154.13% and 228.52%, respectively, compared with USES^B. Similarly, the F-measure values of our method for Bohrbugs are increased by 9.06% and 8.06% for HTTPD and AXIS, respectively. Moreover, the accuracy values for the HTTPD and AXIS projects are improved by 21.09% and

Authorized licensed use limited to: BEIHANG UNIVERSITY. Downloaded on November 24,2021 at 06:03:13 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

TABLE VII
RESULTS OF THE ARB/NAM CLASSIFICATION ON LINUX DATA1

Method	Accuracy	F-measure
SGD	0.856	0.865
LR	0.856	0.861
GNB	0.753	0.721
GBC	**0.882**	**0.909**
DTC	0.797	0.792
RFC	0.874	0.912
LDA	0.753	0.721
DEEPSIM	**0.945**	**0.945**
Impro.	**7.14%**	**3.96%**

TABLE VIII
RESULTS OF THE ARB SUBTYPE CLASSIFICATION ON LINUX DATA1

Method	Accuracy	F-measure
SGD	0.652	0.661
LR	0.675	0.678
GNB	**0.781**	**0.744**
GBC	0.740	0.696
DTC	0.650	0.655
RFC	0.776	0.720
LDA	0.661	0.666
DEEPSIM	**0.954**	**0.948**
Impro.	**22.15%**	**27.42%**

17.39%, respectively. In addition, the results of the Wilcoxon signed-rank test at a significance level of $\alpha = 0.05$ indicate that the classification results obtained by DEEPSIM are significantly better than those obtained by USES^B.

3) Granularity 3—ARBs and NAMs: In this section, we analyze the experimental results at the third granularity, i.e., classifying Mandelbugs into ARBs and NAMs. An examination of the results in Table VII shows that DEEPSIM achieves better performance than the traditional machine learning classifiers. The accuracy obtained by DEEPSIM is 0.945, which is 7.14% higher than the accuracy obtained by the best machine learning classifier (i.e., 0.882). At the same time, the F-measure obtained by DEEPSIM (i.e., 0.945) is 3.96% higher than the best F-measure obtained by traditional machine learning classifiers (i.e., 0.909).

4) Granularity 4—Subtypes of ARBs and NAMs: In this section, we analyze the experimental classification results at the fourth level of granularity for ARB subtypes and NAM subtypes.

a) Automatic classification of ARB subtypes: As stated in Section II-E, ARBs can be further classified into MEMs, STOs, LOGs, NUMs, and TOTs according to the root causes of the software aging phenomenon. The automatic classification results are presented in Table VIII. DEEPSIM significantly outperforms the traditional machine learning classifiers for both accuracy and the F-measure. The accuracy obtained by DEEPSIM is 0.954, which is 22.15% higher than the best accuracy obtained by traditional machine learning classifiers (i.e., 0.781 obtained by the GNB classifier). Likewise, the F-measure obtained by DEEPSIM is 0.948, 27.42% higher than the best F-measure obtained by traditional machine learning classifiers (i.e., 0.744 obtained by the GNB classifier).

TABLE IX
RESULTS OF THE NAM SUBTYPE CLASSIFICATION ON LINUX DATA1

Method	Accuracy	F-measure
SGD	0.548	0.548
LR	0.549	0.550
GNB	0.576	0.586
GBC	**0.587**	**0.592**
DTC	0.478	0.480
RFC	0.548	0.550
LDA	0.478	0.486
DEEPSIM	**0.670**	**0.667**
Impro.	**14.14%**	**12.67%**

b) Automatic classification of NAM subtypes: Based on the different types of the complexity of their fault triggering conditions, NAMs can be further divided into four categories of ENVs, LAGs, TIMs, and SEQs. The corresponding automatic classification results are shown in Table IX. Compared with the best results obtained by the seven traditional machine learning classifiers, DEEPSIM improves the accuracy from 0.587 to 0.670, corresponding to an increase of 14.14% and improves the F-measure from 0.592 to 0.667, obtaining an increase of 12.67%.

To summarize, our DEEPSIM method performs better than all of the seven comparison methods we used in previous work at all four granularities. In addition, DEEPSIM outperforms USES^B, which further confirms the effectiveness of DEEPSIM in classifying Mandelbugs.

B. RQ2: How Do the Size and the Domain of the Semantic Model Influence DEEPSIM?

In this section, we study the impacts of the semantic model on the classification results. First, we study the effect of the semantic model size, i.e., the effect of training semantic models by using corpora with different numbers of bug reports. Second, we study the influence of the semantic model domain, i.e., the influence of using corpora originating from different domains to train the semantic model.

1) Impact of the Model Size: In our work, a total of 2 038 675 bug reports are collected for semantic model training, as described in Table II. We randomly select a certain number of these bug reports to construct a corpus for semantic model training and continuously increase the number of bug reports included in this corpus. After obtaining semantic models of different sizes in this manner, we analyze the impact of the model size on the classification results.

The experimental results are illustrated in Fig. 5. For all eight methods (i.e., the proposed DEEPSIM method and the seven traditional machine learning methods), as the size of the semantic model increases, the accuracy and F-measure both increase. We further confirm these results by implementing Mann-Kendall

Authorized licensed use limited to: BEIHANG UNIVERSITY. Downloaded on November 24,2021 at 06:03:13 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

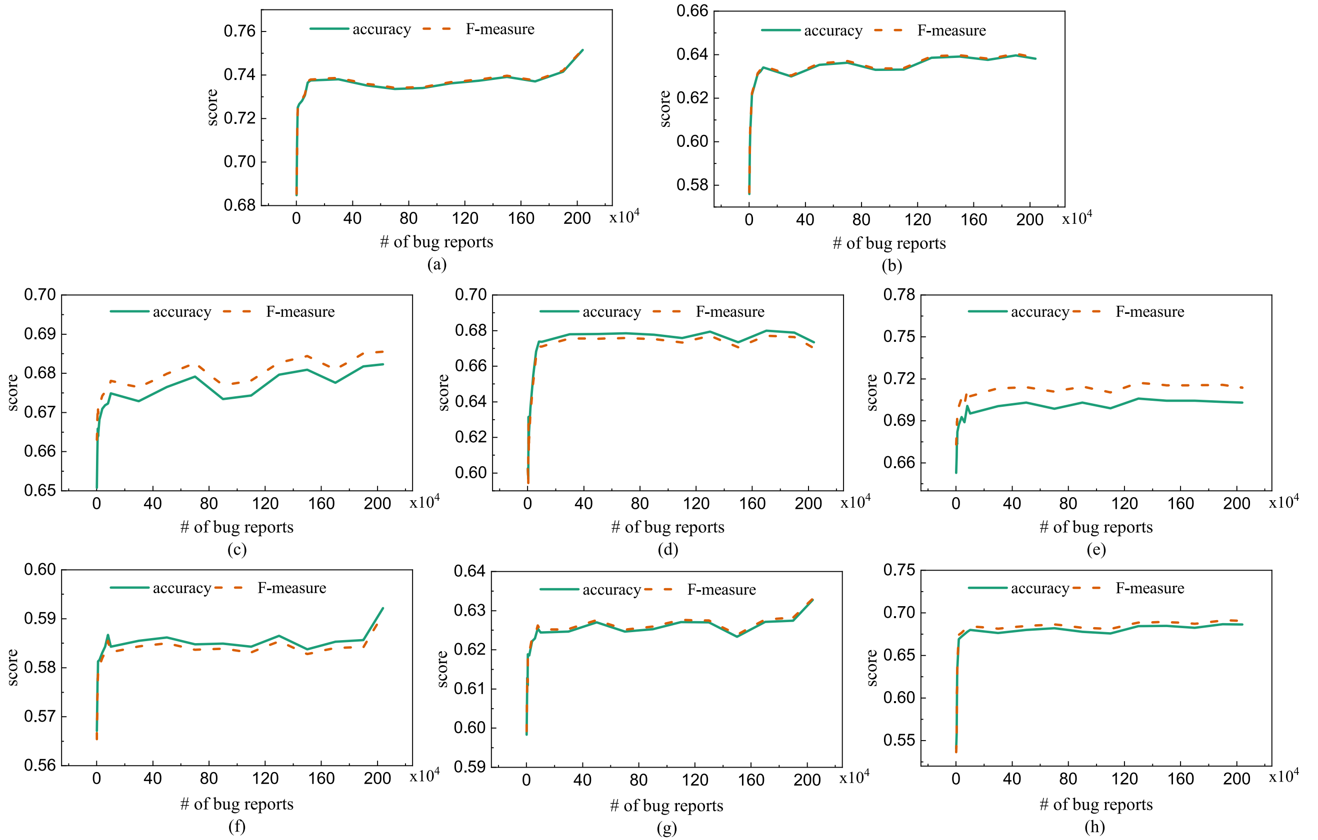

Fig. 5. Influence of the semantic model size on the Bohrbug/Mandelbug classification results obtained using the (a) DEEPSIM, (b) SGD, (c) LR, (d) GNB, (e) GBC, (f) DTC, (g) RFC, and (h) LDA methods.

TABLE X
RESULTS OF MANN–KENDALL TREND DETECTION FOR FIG. 5

Method	Metric	Z	p value	Trend
DEEPSIM	accuracy	4.83	<0.001	Increasing
	F-measure	4.83	<0.001	Increasing
SGD	accuracy	5.22	<0.001	Increasing
	F-measure	5.22	<0.001	Increasing
LR	accuracy	5.42	<0.001	Increasing
	F-measure	5.35	<0.001	Increasing
GNB	accuracy	4.06	<0.001	Increasing
	F-measure	3.99	<0.001	Increasing
GBC	accuracy	4.80	<0.001	Increasing
	F-measure	4.80	<0.001	Increasing
DTC	accuracy	3.93	<0.001	Increasing
	F-measure	3.99	<0.001	Increasing
RFC	accuracy	4.64	<0.001	Increasing
	F-measure	4.57	<0.001	Increasing
LDA	accuracy	5.16	<0.001	Increasing
	F-measure	5.22	<0.001	Increasing

trend detection, as shown in Table X. The Mann–Kendall results indicate that for a significance level of $\alpha = 0.05$, the increasing trends of the classification results with respect to the model size are all statistically significant. In addition, Fig. 5 demonstrates that the classification results initially rise rapidly, i.e., when the number of bug reports in the training corpus is smaller than 100 000. Then, as the size of the corpus continues to increase, the rate of increase gradually decreases. This is because as the size of the corpus increases, the available semantic information is almost completely learned, and the words needed for the classification process gradually become saturated. Thus, the rate of improvement in the classification results slows down. However, the overall trend continues to rise, implying that for the classification of Bohrbugs/Mandelbugs, increasing the size of the semantic model is an effective approach to improve the classification results.

TABLE XI
DETAILS OF OUR SEMANTIC MODEL AND OTHER PRETRAINED SEMANTIC MODELS

Model	# of Words	Size
Bug Reports	704,571	1.22 GB
Google News	3,000,000	1.53 GB
Wiki English	729,001	1.25 GB

2) Impact of the Model Domain: In this section, we download the pretrained Google News and Wiki English semantic models to execute a comparison with our trained bug report semantic model. The details of these three models are shown in Table XI. In this table, the column entitled "# of Words" reports the number of words contained in each model vocabulary, and

Authorized licensed use limited to: BEIHANG UNIVERSITY. Downloaded on November 24,2021 at 06:03:13 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

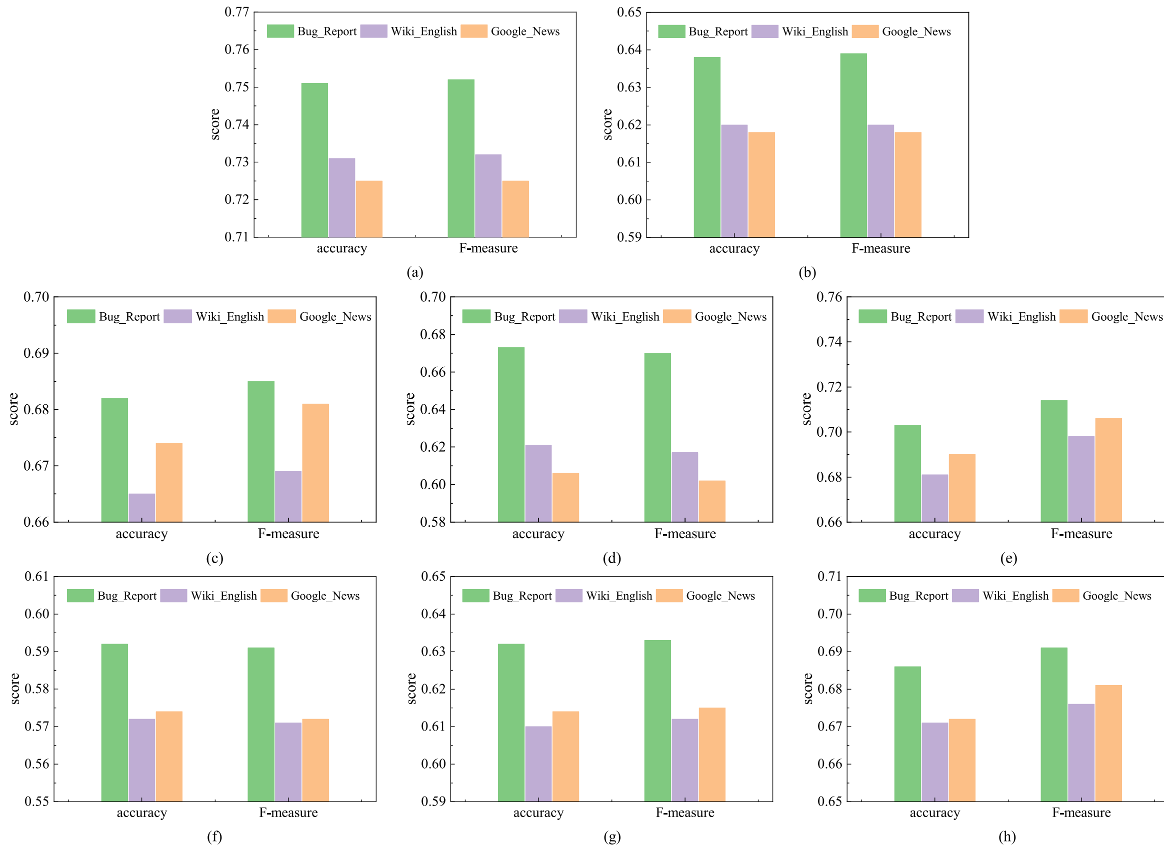

Fig. 6. Influence of the semantic model domain on the Bohrbug/Mandelbug classification results obtained using the (a) DEEPSIM, (b) SGD, (c) LR, (d) GNB, (e) GBC, (f) DTC, (g) RFC, and (h) LDA methods.

the column entitled "Size" indicates the size of each model. Bohrbug/Mandelbug classification experiments are performed in this section.

The classification results are presented in Fig. 6. Although our trained bug report model has a minimal size, it performs better than the Google News model and the Wiki English model. For all eight methods, the bug report model yields the best classification results in terms of both accuracy and the F-measure. Thus, the results presented in Fig. 6 indicate that the domain of the corpus used to train the semantic model influences the Bohrbug/Mandelbug classification results. This finding implies that using a semantic model whose domain is more closely related to the data to be classified can lead to superior results.

C. RQ3: What Impact Does the Parameter Setting of Word Embedding Have on DEEPSIM?

When training the skip-gram model, we consider various important parameters that can affect the quality of the word embeddings, including the *word frequency threshold*, the *vector dimensionality*, and the *context window*.

Among these parameters, the word frequency threshold specifies the lowest frequency at which a word may appear. If the number of occurrences of a word is below the minimum frequency setting, then that word is discarded. If an appropriate word frequency threshold is chosen, low-frequency words that will otherwise interfere with the classification will be removed while useful information will be retained. The context window refers to the window used to train the semantic model. For a context window of n, the positional relationships between the current word and both the n words to the left and the n words to the right within its context are considered. The vector dimensionality determines the number of dimensions of the vector representing each word that is obtained through semantic model training. If the vector dimensionality is too small, the quality of the model will not be sufficient. However, an excessively large value will make the training process highly time-consuming.

Based on the aforementioned considerations and guided by the parameter settings used in other studies [35], [54]–[56], we perform a series of comparative experiments to explore the variations in following three parameters.

1) *Word Frequency:* We vary the word frequency threshold setting from 1 to 6 and train six different semantic models with various word frequency threshold values. Then, we classify the bug reports based on these semantic models. Table XII shows that the classification accuracy achieves the highest score when the word frequency threshold is set to 5.
2) *Context Window:* To establish the context window size, six sets of experiments are performed, and the context window values are varied from 3 to 8. As shown in Table XII, when the context window is set to 5, the classification accuracy has the highest score.

Authorized licensed use limited to: BEIHANG UNIVERSITY. Downloaded on November 24,2021 at 06:03:13 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

TABLE XII
COMPARISON OF DIFFERENT PARAMETER SETTINGS ON THE BUG/NONBUG CLASSIFICATION

Frequency	1	2	3	4	**5**	6
Accuracy	0.874	0.875	0.873	0.876	**0.887**	0.886
Window	3	4	**5**	6	7	8
Accuracy	0.876	0.876	**0.887**	0.875	0.874	0.873
Dimension	50	100	200	300	400	**500**
Accuracy	0.829	0.861	0.884	0.887	0.888	**0.897**

TABLE XIII
EXAMPLES OF SIMILAR WORDS ACCORDING TO DIFFERENT SIMILARITY METRICS

Word: bug			Word: memory		
Manhattan	Cosine	Euclidean	Manhattan	Cosine	Euclidean
issue	issue	issue	diskspace	ram	ram
problem	problem	problem	ram	memeory	diskspace
defect	defect	defect	mempool	memmory	bandwith
probelm	bugs	rainerbielefeld	bandwith	heap	memmory
probem	ticket	probem	vram	diskspace	memeory
probs	issues	isssue	memeory	meory	vram

3) *Vector Dimensionality:* We limit the number of vector dimensions to between 50 and 500, and we perform six comparative experiments with the vector dimensionality set to 50, 100, 200, 300, 400, and 500. The results in Table XII show that vector dimensionality has a significant impact on the experimental results. As the dimensionality increases, the classification accuracy improves. However, the dimensionality cannot be increased indefinitely because this will inevitably downgrade the experimental efficiency.

Based on the experimental results, in the design of DEEPSIM, we set both the word frequency threshold and the value of the context window to 5 and set the vector dimension to 500.

By training the bug report semantic model, we can obtain the word embedding representation of each word, and the distance between vectors should be able to indicate the semantic similarities between words. To evaluate the quality of the word embeddings obtained by training the bug report semantic model, we use three distance metrics, namely, the Manhattan distance, the cosine distance, and the Euclidean distance, all of which are commonly used in machine learning [57].

According to these similarity metrics, we identify the words that are most similar to “bug” and “memory,” as shown in Table XIII. Note that the words in each column are arranged in order of decreasing similarity. An examination of the data presented in Table XIII shows that most similar words are synonyms. For example, the word “bug” resembles “issue,” “problem,” and “defect.” Similarly, “memory” is close to “ram” and “diskspace.” Even if a user or developer misspells a word, this will not affect the learning of the semantic relationships. For example, sometimes “problem” is mistakenly spelled as “probelm” or “probem,” and “memory” may be mistakenly spelled as “memeory” or “memmory.” Our trained semantic model can still recognize the semantic information contained in these misspelled words. These findings indicate that the generated word embeddings can effectively represent the semantic relationships contained in the text of bug reports.

V. THREATS TO VALIDITY

A. Threats to Construct Validity

The threats to construct validity depend on the datasets used in this study. Since bug reports are written by users and developers, the quality of the word embeddings may be affected by the texts within the bug reports. Moreover, the accuracy and completeness of the summary, description, and comment sections of the bug reports may influence the quality of the word embeddings. To mitigate the impact of this threat, we use more than two million bug reports to train the semantic model. Although our trained word embeddings can reliably represent the semantic relations among words, their quality could be further improved by using a corpus of larger size and higher quality.

B. Threats to Internal Validity

The threats to internal validity in this study involve factors that may affect our experimental results. To reduce these internal threats, we repeat each experiment 1000 times to reduce the effects of random errors. In addition, the quality and consistency of manual classification results of bug reports are threats to internal validity. The authors of our previous work [26] separately took manual classification of all bug reports to reduce the threat. During the process, cross-checks were performed and conflicting cases were eliminated through discussion to reach a consensus. Besides, several tools were utilized to help ensure the correctness and consistency of the results. However, no matter how serious the authors are, we admit that the possibility of classification mistakes cannot be completely avoided.

C. Threats to External Validity

Threats of this type are related to the generalizability of the obtained results. We validate the performance of DEEPSIM on 6557 bug reports collected from four popular open-source software systems that are frequently investigated in related studies, including the Linux, MySQL, AXIS, and HTTPD. However, although the four software systems have been widely used and each one is a representative of a category, the evaluation results of DEEPSIM on them may be different from those on some other systems. Besides, the interpretability of CNN is also one of the threats to external effectiveness, even if it has achieved human-level performance in various fields. Due to the black-box nature of deep learning models, it is difficult to explain their decision-making processes. The problem may undermine the confidence on the classification results of our proposed technique. In recent years, researchers have proposed many methods and tried to explain the deep learning process [58]–[60], which may alleviate the problem to some extent.

Authorized licensed use limited to: BEIHANG UNIVERSITY. Downloaded on November 24,2021 at 06:03:13 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

VI. Related Work

A. Empirical Study of Bug Types

Antoniol *et al.* [6] studied 1800 bug reports and classified them into bugs and nonbugs. Similarly, Herzig *et al.* [7] manually examined more than 7000 issues from five open-source projects and found that 33.8% of all bug reports were misclassified. Herbold *et al.* [8] studied the negative impact of mislabeling bugs by performing an extensive empirical analysis of the bug labels created with the SZZ algorithm. In addition to classify bug reports into actual bugs and nonbugs, the classification of specific bug types also has practical implications in the process of software development [61]–[63]. Previous studies have proposed various bug taxonomies from different perspectives [64]–[68]. The earliest and most popular bug classification taxonomy was the ODC scheme [69], which was introduced by IBM. This taxonomy includes 12 categories and classifies bugs in terms of the involved program structure. Another popular bug classification scheme was the HP scheme [70], which aims at deriving process improvement proposals. In [70], bugs were characterized by three attributes, including "origin," "mode," and "type." Tan *et al.* [71] studied 2060 bugs in Linux kernel, Mozilla and Apache from three dimensions, including root causes, impacts, and components.

The criterion used in this article is related to fault triggers and reproducibility, which was first discussed by Gray [70]. In this work, Gray classified bugs into Bohrbugs and Heisenbugs. Here, Bohrbugs are a type of deterministic or hard bugs that are easy to reproduce, whereas Heisenbugs are a type of elusive or soft bugs that behave uncertainly. Grottke and Trivedi [16] updated the nomenclature of Heisenbugs and classified bugs into Bohrbugs and Mandelbugs, where the term "Mandelbug" was defined as the antonym of "Bohrbug." According to this definition, a Mandelbug is a type of "complex" bug that is difficult to detect, isolate, and correct during the testing process.

Several studies have focused on understanding the characteristics of Mandelbugs, including the proportions of Mandelbugs in different software systems [72], the prediction of Mandelbug locations [73], and recovery from Mandelbugs [44]. There are two subcategories of Mandelbugs: ARBs and NAMs [16]. ARBs, which were first defined by Huang *et al.* [74], can lead to an increase in the failure rate and/or a decrease in the performance of a software system. For an ARB, the probability of failure increases with the system's run time, but proactive measures can be taken to clean up the system's internal conditions and, thus, reduce the chance of failure; such measures are referred to as "software rejuvenation" [17]. Based on the aforementioned classification, Cotroneo *et al.* [19] presented a more detailed classification of bug types and analyzed the proportions of bugs in four open-source software systems, i.e., Linux, MySQL, HTTPD, and AXIS.

B. Automatic Classification of Bug Reports

Amounts of approaches were proposed to automatically classify bug reports into actual bugs and nonbugs [9], [10], [12]–[14], [75]–[78]. In [9], [12], and [13], term frequency matrix was built for each bug report and used as input of classifiers. Limsettho *et al.* [75] and Pingclasai *et al.* [79] proposed to derive features via topic-modeling. Herbold *et al.* [14] trained different models for the title and description of the bug report to investigate whether the prediction results of bug types can be improved by considering the structural difference between them. Compared with these studies, except for the classification of bugs and nonbugs, we also classify bugs into more detailed types according to the fault triggering conditions.

Frattini *et al.* [21] automatically classified bug reports into workload-dependent and environment-dependent bugs in accordance with their reproducibility. Xia *et al.* [20] developed a fuzzy-set-based feature selection algorithm for automatically classifying Bohrbugs/Mandelbugs. The difference between these two studies and this work is that the models used for textual representation in these previous studies did not consider the semantic relationships contained in the bug report texts. In our previous work [23], an automatic classification framework based on word2vec was proposed to classify Mandelbugs. In that work, several traditional machine learning classifiers were used for classification. Traditional machine learning classifiers have a common limitation, i.e., their performance relies heavily on input features, which means that they cannot learn higher level features by themselves. However, highly correlated semantic relations still exist among different word embeddings. To compensate for this limitation and learn the higher level features of bug reports, we propose the DEEPSIM method in this article to combine the semantic model with a deep learning classifier. In addition, to mitigate the class imbalance problem in the data, SMOTE is also introduced.

VII. Conclusion

In this article, we proposed DEEPSIM, an automatic Mandelbug classification method that combines a semantic model with a deep learning classifier. In DEEPSIM, a bug report semantic model was trained based on millions of bug reports to learn semantic relations among words. With the trained model, each word in the bug reports was represented as a word embedding. In addition, a CNN model was designed to capture the higher level features in the bug reports from word embeddings. We evaluated DEEPSIM on a total of 6557 bug reports and compared the results with those of existing studies. The experimental results showed that our method was effective for the classification of Mandelbugs and performed better than the existing methods for this purpose. Moreover, we investigated the impacts of the semantic model and word embedding parameters on the classification results.

In the future, we plan to study methods for classifying bugs in machine learning systems, such as the TensorFlow framework. According to our research [80], the Mandelbugs in machine learning frameworks have different characteristics from those in traditional software systems. Thus, how to improve the Mandelbug classification accuracy for this kind of system is an important topic. In addition, handling all-in-one classification of bug reports is an important task, and specific strategies need

Authorized licensed use limited to: BEIHANG UNIVERSITY. Downloaded on November 24,2021 at 06:03:13 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

to be developed to deal with the extremely imbalanced multiclass problem that existed in bug data. This will be our work in the near future.

References

[1] B. Wang, L. Xu, M. Yan, C. Liu, and L. Liu, "Multi-dimension convolutional neural network for bug localization," *IEEE Trans. Services Comput.*, Jul. 2020, pp. 1–1.

[2] H. Zhong, X. Wang, and H. Mei, "Inferring bug signatures to detect real bugs," *IEEE Trans. Softw. Eng.*, May 2020, pp. 1–1.

[3] M. Yan, X. Xia, Y. Fan, A. E. Hassan, D. Lo, and S. Li, "Just-in-time defect identification and localization: A two-phase framework," *IEEE Trans. Softw. Eng.*, Mar. 2020, pp. 1–1.

[4] S. Chakraborty *et al.*, "EReinit: Scalable and efficient fault-tolerance for bulk-synchronous MPI applications," *Concurrency Comput.: Pract. Experience*, vol. 32, no. 3, pp. 1–21, 2020.

[5] H. Cruz, R. P. Duarte, and H. Neto, "Fault-tolerant architecture for on-board dual-core synthetic-aperture radar imaging," in *Proc. Int. Symp. Appl. Reconfigurable Comput.*, 2019, pp. 3–16.

[6] G. Antoniol, K. Ayari, M. Di Penta, F. Khomh, and Y.-G. Guéhéneuc, "Is it a bug or an enhancement?: A text-based approach to classify change requests," in *Proc. Conf. Center Adv. Stud. Collaborative Res.: Meeting Minds*, 2008, pp. 304–318.

[7] K. Herzig, S. Just, and A. Zeller, "It's not a bug, it's a feature: How misclassification impacts bug prediction," in *Proc. IEEE 35th Int. Conf. Softw. Eng.*, 2013, pp. 392–401.

[8] S. Herbold, A. Trautsch, F. Trautsch, and B. Ledel, "Issues with SZZ: An empirical assessment of the state of practice of defect prediction data collection," Nov. 2019, *arXiv:1911.08938*.

[9] I. Chawla and S. K. Singh, "An automated approach for bug categorization using fuzzy logic," in *Proc. 8th India Softw. Eng. Conf.*, 2015, pp. 90–99.

[10] Y. Zhou, Y. Tong, R. Gu, and H. Gall, "Combining text mining and data mining for bug report classification," *J. Softw.: Evol. Process*, vol. 28, no. 3, pp. 150–176, 2016.

[11] H. Qin and X. Sun, "Classifying bug reports into bugs and non-bugs using LSTM," in *Proc. 10th Asia-Pacific Symp. Internetware*, 2018, pp. 1–4.

[12] N. Pandey, A. Hudait, D. K. Sanyal, and A. Sen, "Automated classification of issue reports from a software issue tracker," in *Proc. Prog. Intell. Comput. Techn.: Theory, Pract., Appl.*, 2018, pp. 423–430.

[13] A. F. Otoom, S, Al jdaeh, and M. Hammad, "Automated classification of software bug reports," in *Proc. 9th Int. Conf. Inf. Commun. Manage.*, 2019, pp. 17–21.

[14] S. Herbold, A. Trautsch, and F. Trautsch, "On the feasibility of automated prediction of bug and non-bug issues," *Empirical Softw. Eng.*, vol. 25, no. 6, pp. 5333–5369, 2020.

[15] B. Boehm, and V. R. Basili, "Software defect reduction top 10 list," *Found. Empirical Softw. Eng.: Legacy Victor R. Basili*, vol. 426, no. 37, pp. 426–431, 2005.

[16] M. Grottke and K. S. Trivedi, "Software faults, software aging and software rejuvenation (special survey: New development of software reliability engineering)," *J. Rel. Eng. Assoc. Jpn.*, vol. 27, no. 7, pp. 425–438, 2005.

[17] R. Hanmer, "Software rejuvenation," in *Proc. 17th Conf. Pattern Lang. Programs*, 2010, pp. 1–13.

[18] R. Chillarege, "Comparing four case studies on Bohr-Mandel characteristics using ODC," in *Proc. IEEE Int. Symp. Softw. Rel. Eng. Workshops*, 2013, pp. 285–289.

[19] D. Cotroneo, M. Grottke, R. Natella, R. Pietrantuono, and K. S. Trivedi, "Fault triggers in open-source software: An experience report," in *Proc. IEEE 24th Int. Symp. Softw. Rel. Eng.*, 2013, pp. 178–187.

[20] X. Xia, D. Lo, X. Wang, and B. Zhou, "Automatic defect categorization based on fault triggering conditions," in *Proc. IEEE 19th Int. Conf. Eng. Complex Comput. Syst.*, 2014, pp. 39–48.

[21] F. Frattini, R. Pietrantuono, and S. Russo, "Reproducibility of software bugs," in *Principles of Performance and Reliability Modeling and Evaluation*. Berlin, Germany: Springer, 2016, pp. 551–565.

[22] Y. Zhang, R. Jin, and Z.-H. Zhou, "Understanding bag-of-words model: A statistical framework," *Int. J. Mach. Learn. Cybern.*, vol. 1, no. 1–4, pp. 43–52, 2010.

[23] X. Du, Z. Zheng, G. Xiao, and B. Yin, "The automatic classification of fault trigger based bug report," in *Proc. IEEE Int. Symp. Softw. Rel. Eng. Workshops*, 2017, pp. 259–265.

[24] T. Mikolov, I. Sutskever, K. Chen, G. S. Corrado, and J. Dean, "Distributed representations of words and phrases and their compositionality," in *Proc. Adv. Neural Inf. Process. Syst.*, 2013, pp. 3111–3119.

[25] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," Jan. 2013, *arXiv:1301.3781*.

[26] G. Xiao, Z. Zheng, B. Yin, K. S. Trivedi, X. Du, and K. Cai, "An empirical study of fault triggers in the Linux operating system: An evolutionary perspective," *IEEE Trans. Rel.*, vol. 68, no. 4, pp. 1356–1383, Dec. 2019.

[27] S. Ruder, I. Vulić, and A. Søgaard, "A survey of cross-lingual word embedding models," *J. Artif. Intell. Res.*, vol. 65, pp. 569–631, 2019.

[28] A. El Mahdaouy, S. O. El Alaoui, and E. Gaussier, "Word-embedding-based pseudo-relevance feedback for Arabic information retrieval," *J. Inf. Sci.*, vol. 45, no. 4, pp. 429–442, 2019.

[29] Y. Hao *et al.*, "An end-to-end model for question answering over knowledge base with cross-attention combining global knowledge," in *Proc. 55th Annu. Meeting Assoc. Comput. Linguistics*, 2017, pp. 221–231.

[30] K. Chen *et al.*, "A neural approach to source dependence based context model for statistical machine translation," *IEEE/ACM Trans. Audio, Speech, Lang. Process.*, vol. 26, no. 2, pp. 266–280, Feb. 2018.

[31] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *Trans. Assoc. Comput. Linguistics*, vol. 5, pp. 135–146, 2017.

[32] J. Pennington, R. Socher, and C. D. Manning, "Glove: Global vectors for word representation," in *Proc. Conf. Empirical Methods Natural Lang. Process.*, 2014, pp. 1532–1543.

[33] J. Devlin, M. W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," Oct. 2018, *arXiv:1810.04805*.

[34] R. Kurnia, Y. D. Tangkuman, and A. S. Girsang, "Classification of user comment using Word2vec and SVM classifier," *Int. J. Adv. Trends Comput. Sci. Eng.*, vol. 9, pp. 643–648, 2020.

[35] X. Yang, D. Lo, X. Xia, L. Bao, and J. Sun, "Combining word embedding with information retrieval to recommend similar bug reports," in *Proc. IEEE 27th Int. Symp. Softw. Rel. Eng.*, 2016, pp. 127–137.

[36] Y. Uneno, O. Mizuno, and E.-H. Choi, "Using a distributed representation of words in localizing relevant files for bug reports," in *Proc. IEEE Int. Conf. Softw. Qual., Rel. Secur.*, 2016, pp. 183–190.

[37] K. W. Church, "Word2vec," *Natural Lang. Eng.*, vol. 23, no. 1, pp. 155–162, 2017.

[38] Y. Goldberg and O. Levy, "Word2vec explained: Deriving Mikolov et al.'s negative-sampling word-embedding method," Feb. 2014, *arXiv:1402.3722*.

[39] M. Dwarampudi and N. Reddy, "Effects of padding on LSTMs and CNNs," Mar. 2019, *arXiv:1903.07288*.

[40] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic minority over-sampling technique," *J. Artif. Intell. Res.*, vol. 16, pp. 321–357, 2002.

[41] W. Jia, H. Xia, L. Jia, Y. Deng, and X. Liu, "The selection of wart treatment method based on synthetic minority over-sampling technique and axiomatic fuzzy set theory," *Biocybern. Biomed. Eng.*, vol. 40, no. 1, pp. 517–526, 2020.

[42] G. Lemaître, F. Nogueira, and C. K. Aridas, "Imbalanced-learn: A Python toolbox to tackle the curse of imbalanced datasets in machine learning," *J. Mach. Learn. Res.*, vol. 18, no. 1, pp. 559–563, 2017.

[43] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, pp. 436–444, 2015.

[44] M. Grottke, D. S. Kim, R. Mansharamani, M. Nambiar, R. Natella, and K. S. Trivedi, "Recovery from software failures caused by Mandelbugs," *IEEE Trans. Rel.*, vol. 65, no. 1, pp. 70–87, Mar. 2016.

[45] T. Hall, S. Beecham, D. Bowes, D. Gray, and S. Counsell, "A systematic literature review on fault prediction performance in software engineering," *IEEE Trans. Softw. Eng.*, vol. 38, no. 6, pp. 1276–1304, Nov./Dec. 2012.

[46] A. Kamilaris and F. X. Prenafeta-Boldú, "Deep learning in agriculture: A survey," *Comput. Electron. Agriculture*, vol. 147, pp. 70–90, 2018.

[47] S. Hosseini, B. Turhan, and D. Gunarathna, "A systematic literature review and meta-analysis on cross project defect prediction," *IEEE Trans. Softw. Eng.*, vol. 45, no. 2, pp. 111–147, Feb. 2019.

[48] C. Silva and B. Ribeiro, "The importance of stop word removal on recall values in text categorization," in *Proc. IEEE Int. Joint Conf. Neural Netw.*, 2003, vol. 3, pp. 1661–1666.

[49] J. Plisson, N. Lavrac, and D. Mladenic, "A rule based approach to word lemmatization," in *Proc. IS04*, 2004, pp. 83–86.

[50] S. Bird and E. Loper, "NLTK: The natural language toolkit," in *Proc. ACL Interactive Poster Demonstration Sessions, Assoc. Comput. Linguistics*, 2004, pp. 31–38.

Authorized licensed use limited to: BEIHANG UNIVERSITY. Downloaded on November 24,2021 at 06:03:13 UTC from IEEE Xplore. Restrictions apply.

This article has been accepted for inclusion in a future issue of this journal. Content is final as presented, with the exception of pagination.

[51] D. D. Patil, V. Wadhai, and J. Gokhale, "Evaluation of decision tree pruning algorithms for complexity and classification accuracy," *Int. J. Comput. Appl.*, vol. 11, no. 2, pp. 23–30, 2010.

[52] D. Gray, D. Bowes, N. Davey, Y. Sun, and B. Christianson, "Further thoughts on precision," in *Proc. 15th Annu. Conf. Eval. Assessment Softw. Eng.*, 2011, pp. 129–133.

[53] F. Pedregosa *et al.*, "Scikit-learn: Machine learning in Python," *J. Mach. Learn. Res.*, vol. 12, pp. 2825–2830, 2011.

[54] Q. Le and T. Mikolov, "Distributed representations of sentences and documents," in *Proc. Int. Conf. Mach. Learn.*, 2014, pp. 1188–1196.

[55] M. J. Lin, C. Z. Yang, C. Y. Lee, and C. C. Chen, "Enhancements for duplication detection in bug reports with manifold correlation features," *J. Syst. Softw.*, vol. 121, pp. 223–233, 2016.

[56] R. Kato and H. Goto, "Categorization of web news documents using Word2Vec and deep learning," in *Proc. Int. Conf. Ind. Eng. Oper. Manage.*, 2016, pp. 476–481.

[57] S. Pandit *et al.*, "A comparative study on distance measuring approaches for clustering," *Int. J. Res. Comput. Sci.*, vol. 2, no. 1, pp. 29–31, 2011.

[58] K. Pei, Y. Cao, J. Yang, and S. Jana, "DeepXplore: Automated whitebox testing of deep learning systems," in *Proc. 26th Symp. Operating Syst. Princ.*, 2017, pp. 1–18.

[59] J. M. Zhang, M. Harman, L. Ma, and Y. Liu, "Machine learning testing: Survey, landscapes and horizons," *IEEE Trans. Softw. Eng.*, Feb. 2020, pp. 1–1.

[60] X. Xie *et al.*, "DeepHunter: A coverage-guided fuzz testing framework for deep neural networks," in *Proc. 28th ACM SIGSOFT Int. Symp. Softw. Testing Anal.*, 2019, pp. 146–157.

[61] I. Alazzam, A. Aleroud, Z. Al Latifah, and G. Karabatis, "Automatic bug triage in software systems using graph neighborhood relations for feature augmentation," *IEEE Trans. Comput. Social Syst.*, vol. 7, no. 5, pp. 1288–1303, Oct. 2020.

[62] S. S. Chouhan, S. S. Rathore, and R. Choudhary, "A study of aging-related bugs prediction in software system," in *Proc. Int. Conf. Paradigms Comput., Commun., Data Sci.*, 2021, pp. 49–61.

[63] A. Avritzer, M. Grottke, and D. S. Menasché, "Using software aging monitoring and rejuvenation for the assessment of high-availability systems," in *Handbook of Software Aging and Rejuvenation: Fundamentals, Methods, Applications, and Future Directions*. Singapore: World Scientific, 2020, p. 197.

[64] S. K. Sahoo, J. Criswell, and V. Adve, "An empirical study of reported bugs in server software with implications for automated bug diagnosis," in *Proc. ACM/IEEE 32nd Int. Conf. Softw. Eng.*, 2010, vol. 1, pp. 485–494.

[65] N. Humbatova, G. Jahangirova, G. Bavota, V. Riccio, A. Stocco, and P. Tonella, "Taxonomy of real faults in deep learning systems," in *Proc. ACM/IEEE 42nd Int. Conf. Softw. Eng.*, 2020, pp. 1110–1121.

[66] S. Lu, S. Park, E. Seo, and Y. Zhou, "Learning from mistakes: A comprehensive study on real world concurrency bug characteristics," in *Proc. 13th Int. Conf. Architectural Support Program. Lang. Oper. Syst.*, 2008, pp. 329–339.

[67] J. Garcia, Y. Feng, J. Shen, S. Almanee, Y. Xia, and Q. A. Chen, "A comprehensive study of autonomous vehicle bugs," in *Proc. ACM/IEEE 42nd Int. Conf. Softw. Eng.*, 2020, pp. 385–396.

[68] S. Li *et al.*, "An exploratory study of bugs in extended reality applications on the web," in *Proc. IEEE 31st Int. Symp. Softw. Rel. Eng.*, 2020, pp. 172–183.

[69] R. Chillarege *et al.*, "Orthogonal defect classification—A concept for in-process measurements," *IEEE Trans. Softw. Eng.*, vol. 18, no. 11, pp. 943–956, Nov. 1992.

[70] R. B. Grady, *Practical Software Metrics for Project Management and Process Improvement*. Englewood Cliffs, NJ, USA: Prentice-Hall, 1992.

[71] L. Tan, C. Liu, Z. Li, X. Wang, Y. Zhou, and C. Zhai, "Bug characteristics in open source software," *Empirical Softw. Eng.*, vol. 19, no. 6, pp. 1665–1705, 2014.

[72] R. Chillarege, "Understanding Bohr-Mandel bugs through ODC triggers and a case study with empirical estimations of their field proportion," in *Proc. IEEE 3rd Int. Workshop Softw. Aging Rejuvenation*, 2011, pp. 7–13.

[73] G. Carrozza, D. Cotroneo, R. Natella, R. Pietrantuono, and S. Russo, "Analysis and prediction of Mandelbugs in an industrial software system," in *Proc. IEEE 6th Int. Conf. Softw. Testing, Verification, Validation*, 2013, pp. 262–271.

[74] Y. Huang, C. Kintala, N. Kolettis, and N. D. Fulton, "Software rejuvenation: Analysis, module and applications," in *Proc. 25th Int. Symp. Fault Tolerant Comput.*, 1995, pp. 381–390.

[75] N. Limsettho, H. Hata, and K.-I. Matsumoto, "Comparing hierarchical Dirichlet process with latent Dirichlet allocation in bug report multiclass classification," in *Proc. 15th IEEE/ACIS Int. Conf. Softw. Eng., Artif. Intell., Netw. Parallel/Distrib. Comput.*, 2014, pp. 1–6.

[76] I. Chawla and S. K. Singh, "Automated labeling of issue reports using semi supervised approach," *J. Comput. Methods Sci. Eng.*, vol. 18, no. 1, pp. 177–191, 2018.

[77] M. S. Zolkeply and J. Shao, "Classifying software issue reports through association mining," in *Proc. 34th ACM/SIGAPP Symp. Appl. Comput.*, 2019, pp. 1860–1863.

[78] P. Terdchanakul, H. Hata, P. Phannachitta, and K. Matsumoto, "Bug or not? Bug report classification using N-gram IDF," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol.*, 2017, pp. 534–538.

[79] N. Pingclasai, H. Hata, and K.-I. Matsumoto, "Classifying bug reports to bugs and other requests using topic modeling," in *Proc. IEEE 20th Asia-Pacific Soft. Eng. Conf.*, 2013, vol. 2, pp. 13–18.

[80] X. Du, G. Xiao, and Y. Sui, "Fault triggers in the TensorFlow framework: An experience report," in *Proc. IEEE 31st Int. Symp. Softw. Rel. Eng.*, 2020, pp. 1–12.

Authorized licensed use limited to: BEIHANG UNIVERSITY. Downloaded on November 24,2021 at 06:03:13 UTC from IEEE Xplore. Restrictions apply.