IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received 11 April 2023, accepted 2 May 2023, date of publication 8 May 2023, date of current version 24 May 2023.

Digital Object Identifier 10.1109/ACCESS.2023.3273595

RESEARCH ARTICLE

On Fusing Artificial and Convolutional Neural Network Features for Automatic Bug Assignments

ATISH KUMAR DIPONGKOR[1], MD. SAIFUL ISLAM[2], (Senior Member, IEEE), ISHTIAQUE HUSSAIN[3], SIRA YONGCHAREON[4], (Senior Member, IEEE), AND SAJIB MISTRY[5], (Member, IEEE)

[1]Department Of Computer Science and Engineering, Jashore University of Science and Technology, Jessore 7408, Bangladesh
[2]School of Information and Physical Sciences, The University of Newcastle, Callaghan, NSW 2308, Australia
[3]Bloomberg LP, Princeton, NJ 08540, USA
[4]School of Engineering, Computer and Mathematical Sciences, Auckland University of Technology, Auckland 1010, New Zealand
[5]School of Electrical Engineering, Computing and Mathematical Sciences (EECMS), Curtin University, Bentley, WA 6102, Australia

Corresponding author: Md. Saiful Islam (saiful.islam@newcastle.edu.au)

The work of Saiful Islam was supported by the School of Information and Physical Sciences at The University of Newcastle, Australia. The work of Ishtiaque Hussain was supported by the Chancellor's Grant 2021 from Penn State Abington.

ABSTRACT Automated bug report assignment is critical for large-scale software projects where reported bugs are frequent and expert developers are required to fix them on time. Finding an appropriate developer with the necessary skill sets and prior experience in fixing similar bugs is difficult and can be an expensive process, depending on the severity of the reported bug. To address this issue, researchers have proposed several machine learning and deep learning-based automated bug report assignment techniques that make use of historical data on reported bugs as well as fixer information. However, there is still room for improvement in the performance of these techniques. In this paper, we propose a novel deep learning-based approach that utilizes two sets of features from the reported bugs' textual data, namely contextual information and the occurrence of repeating keywords. We develop convolutional neural network and artificial neural network modules to mine these features. We then fuse these two sets of extracted features to assign a bug to an appropriate developer. We conduct extensive experiments on eight benchmark datasets of open-source, real-world software projects to assess the effectiveness of our approach. The experimental results demonstrate that our information fusion-based approach outperforms previous models and improves automated bug report assignment. Furthermore, we debug the errors of our proposed model and publish all source code so that future researchers can contribute to this problem.

INDEX TERMS Artificial neural network, bug report assignment, convolutional neural network, deep learning, dimensionality reduction.

I. INTRODUCTION

Bugs in software are commonly caused by development time constraints, a lack of skills or domain knowledge, and insufficient testing. When these bugs are reported and eventually fixed, they are frequently regarded as client feedback that can help a software gradually improve to perfection. When a bug occurs in any industrial or open source project, it is routinely textually documented in a 'bug repository' or 'bug tracking system'. During bug triage, duplicate bug reports are detected, reproduction steps are validated, severity or priority is determined, and the report's validity is assessed. The bug report assignment process then finds and assigns an appropriate developer to fix the bug. The bug report assignment process, however, is not simple for large-scale software projects because it requires bug assignments based on the developers' domain knowledge and expertise. Furthermore, large-scale software projects produce more bugs than smaller projects. For example, Anvik et al. [1] reported that nearly 300 bugs are discovered in various Mozilla projects

The associate editor coordinating the review of this manuscript and approving it for publication was Shuihua Wang.

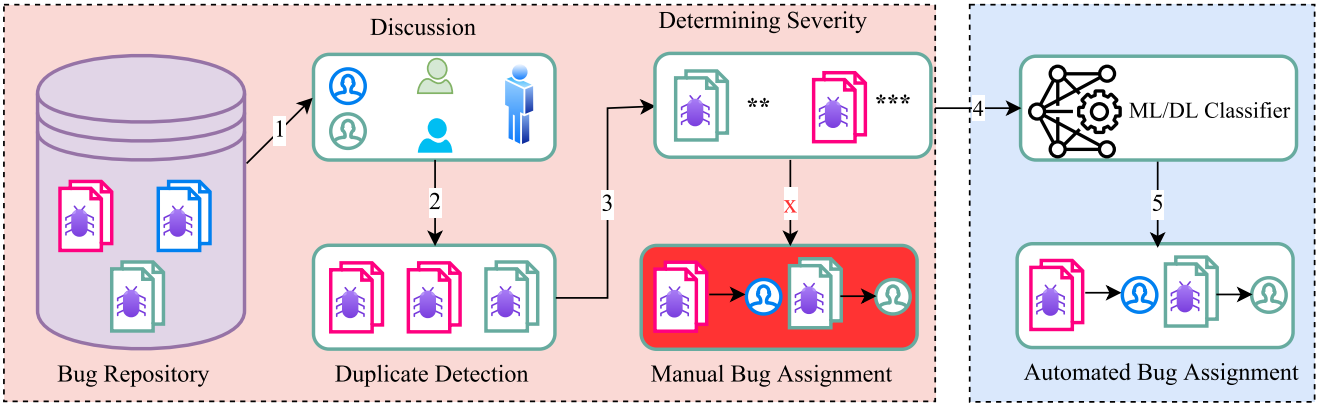


FIGURE 1. Automated bug assignment based on machine learning (ML) and deep learning (DL) techniques.

every day. Within Mozilla projects, the Firefox project alone receives an average of eight bug reports per day that require triage. It is an error-prone task that necessitates the use of a specialised quality assurance team. Similar to Mozilla, the Eclipse project receives a large number of daily bug reports, forcing them to implement decentralized triaging mechanisms in which teams are responsible for bug triaging for their components [2]. Manual bug assignment is costly, time-consuming, and unscalable because it consumes valuable development time and resources [3], [4], [5], [6], [7]. As a result, automated bug assignment is crucial, and improving the performance of existing techniques can be vital.

To report a bug in a 'bug tracking system' such as JIRA [8] or Bugzilla [9], among other optional things (e.g., stack trace, screenshots, etc.), the 'Title' and 'Description' of that bug are required. When a developer fixes a bug, his/her information is added later. This repetitive process generates historical data that grows over time. Machine Learning (ML) [10], [11] and Deep Learning (DL) [12], [13] techniques can learn and assign future bugs to an appropriate developer based on this historical information. Bug report assignment, according to those techniques, is a multi-class classification problem in which the 'Title' and 'Description' of a bug is treated as textual data and the respective developer who fixed that bug is considered the label/class. The idea of ML and DL-based automated bug assignment is illustrated in Fig. 1.

In this paper, we propose a novel DL-based model for automatically assigning bugs among developers. Our model's architecture is made up of two major modules that extract features: Convolutional Neural Network (CNN) and Artificial Neural Network (ANN). These modules operate in parallel, and their output is later fused to achieve the best performance. The main goal of the CNN module is to use convolutional filters to extract contextual relationships between adjacent words from the textual information of bug reports. The ANN module, on the other hand, utilizes *n*-grams to extract repeating keywords from bug reports. Convolutional filters have been used in previous studies for automated bug assignment [12], [14], [15]. To the best of our knowledge, we are the first to use ANN module to extract repeating keywords. To validate our research ideas, we conduct extensive experiments that demonstrate that our model outperforms existing DL-based techniques.

To be specific, the main contributions of this paper are as follows.

1) We introduce novel data preprocessing steps to clean the bug assignment dataset.
2) We develop a novel DL-based model to assign bugs among developers that makes use of two types of features: repeating keywords and contextual relationship between consecutive words.
3) We conduct extensive experiments on eight benchmark, open-source large projects and compare results to validate the robustness of our model against existing techniques.
4) We perform novel debugging of our model's errors and present the insights for future research that could improve overall performance.
5) The code for all of our experiments is freely available on the companion website [16].

The remainder of the paper is structured as follows: In Section II, we highlight notable existing works. Then, in Section III, we discuss our proposed approach. We design the necessary experiments to validate our research idea in Section IV. Then, in Section V, we discuss our findings. Finally, this study is concluded in Section VI.

II. RELATED WORK

Automatic bug assignment is an evolving field of study that employs custom ML algorithms and, more recently, DL techniques. The vast majority of these works are based on open bug repositories and open-source projects (e.g.,Bugzilla, JIRA, GNATS, Eclipse, Firefox, etc.). Anvik et al. [1] provide a detailed description of bug reports on their anatomy, life-cycle, and interactions with various development roles (e.g., reported, fixer, triager, etc.). In the following subsections, we discuss the background study in a categorized fashion. In addition, we discuss how we combined contextual

information retrieval with convolutional neural networks in other application domains.

A. INSIGHTS INTO BUG REPORTS, SOURCE CODE AND DEVELOPERS

Though not directly related to automatic bug assignment, there have been some early works that investigate the interrelationship between bug reports, source code, and developers. These investigations can help determine future problem spots for maintenance [17], ownership architecture that identifies expert developers for different parts of the system [18], predict files that change together [19] and clustering of related bug reports due to common errors [20]. Matter et al. [21] used source code and vocabulary-based expertise models for automatic bug assignment. Shokripour et al. [22] extended that and used a location-based approach in indexing only noun terms from four different sources (e.g., source code identifiers for names of variables, classes, and methods; source code comments, commit messages and previously fixed bug reports) that predicted bug location into source code files and then utilized term weighing method to provide a bug report assignment recommendation. Almhana and Kessentini [23] used multi-objective search to rank bug reports for developers according to their priorities and dependency between them. In our work, we develop an automatic deep bug assignment model based on features extracted from reported bugs and the history of bug assignments. Our model is able to assign a bug to a developer based on the description of bugs fixed by them before.

B. AUTOMATIC BUG ASSIGNMENT USING MACHINE LEARNING TECHNIQUES

Bug assignment is regarded as a supervised classification problem where the training data and input are the textual information from the bug reports and other related sources and the output classes are the developers' names. Researchers have applied different ML algorithms in this context, e.g., Naïve Bayes, Bayesian Networks, C4.5, Support Vector Machines (SVM), and k-Nearest Neighbors (kNN) and other feature selection, extraction techniques [24], [25]. Murphy and Cubranic [2] are regarded as the first to apply ML algorithms for bug assignment and reported 30% classification accuracy on more than 15 thousand bug reports from the Eclipse project. In follow-up works, Anvik et al. [1], [5] improved the accuracy by filtering out noisy data based on bug status and also compared different ML algorithms and showed that the SVM algorithm provides the best results [1]. Neysiani et al. [26] proposed a feature extraction model to identify duplicate bug reports. According to them, 1-gram and 2-gram provide better validation performance. In this study, we use the same n-grams to extract features from bug reports to distinguish developers since it provided promising results in earlier studies.

Ahsan et al. [27] used dimensionality reduction techniques (e.g., feature extraction and selection), and Nasim et al. [28] used *alphabet frequency matrix* with different ML algorithms and both confirmed that SVM or variant of SVM algorithms perform the best in automatic bug assignment. Wu et al. [29] and Xia et al. [30] used kNN algorithms combined with different similarity metrics, e.g., previous bug terms, term weighting, and developer ranking for classifying bug reports. Jonsson et al. [31] showed that for proprietary software projects, the ensemble-based Stacked Generalization (SG) technique that combines several classifiers scales well and outperforms other techniques that use single individual classifiers. They also recommended that old bug reports should be ignored and at least 2000 bug reports to be used for training data. To improve bug assignment performance, Ge et al. [32] proposed a high-dimensional hybrid data reduction technique by removing noninformative bug reports and words. In this work, we train three traditional ML models such as SVM, RF, and NB. During ML model training, we remove words such as error logs and hyperlinks as we find that these words affect the model performance negatively. In addition, we apply Principle Component Analysis (PCA) to select significant features from our training data, and experimental results show that our ML models can provide promising results than existing studies due to proper data cleaning and feature selection.

C. AUTOMATIC BUG ASSIGNMENT USING DEEP LEARNING TECHNIQUES

Zhang [33] proposed a solution using a Deep Neural Network (DNN) for assigning bugs to components instead of developers. However, it is required to assign bugs to developers because a bug can be assigned to multiple components. Lee et al. [12] applied CNN and Word2Vec [34] word embedding for feature extraction and showed that DL-based techniques are better than other ML techniques in bug assignment. Mani et al. [13] used a deep bidirectional recurrent neural network with attention (DBRNN-A) that learns features from long word sequences and trained their model with unfixed bug reports. Guo et al. [14] applied developer-activity-based CNN techniques (CNN-DA) where they also used Word2Vec for word embedding and applied word segmentation, stop word removal and stemming techniques in their pre-processing step. Zaidi et al. [15] applied CNN techniques for the bug assignment problem but used three existing word embedding, namely, Word2Vec [34], GloVe [35] and ELMo [36] to train their dataset and compared their performance by calculating the top-k accuracy. They concluded that context-sensitive word embedding, ELMo outperforms the other two. The major difference between our model and these models is that we use two types of features (repeating keywords and the contextual relationship between adjacent words) to train our model. Moreover, we find that many of these generalized word representations or embedding do not work well for automated bug assignment since these representations are not fine-tuned locally with domain-specific bug information. As a result, we train our model without pretrained generalized word embedding and find that our DL model is able to outperform these models.

D. RECENT ADVANCEMENTS

Alazzam et al. [37] introduced a graph-based feature augmentation approach to classify bug reports into different priorities. Mohsin et al. [6] proposed a two-step self-paced unified bug classifier and subsequent bug assignment model. The authors considered ''component'', ''specific'', and ''general'' categories for the reported bugs and claimed 98% classification accuracy for their model, which exploits the connections between nodes in the Software Bug Report Network to identify the target features. Jahanshahi and Cevik [7] presented an innovative bug triage approach that addresses schedule and dependency considerations. By leveraging textual data, bug fixing costs, and bug dependencies, this method aims to reduce bug fixing time and avoid infeasible assignment of blocked bugs. The approach also takes into account the developers' schedule to determine the precise assignment date, resulting in a more efficient and effective bug resolution process. Sepahvand et al. [38] proposed a novel approach for predicting the need to assign a bug to a designer. They developed a convolutional neural network (CNN)-based model that learned the unique characteristics of bug reports that contribute to the creation of bad smells in the code. The model extracts features from the textual information, such as summary and description, of each bug and achieved an accuracy of 75% or higher on datasets with sufficient samples for deep learning-based model training. Kukkar et al. [39] proposed an ant colony optimization-based programmer recommendation model to manage software bugs. The authors utilized the Ant colony optimization (ACO) method to determine the appropriate subsets of features in the feature selection stage. In the programmer recommendation stages, the authors exploited three programmer metrics, namely, functionality ranking, bug occurrence, and mean Bug fixing time. In their recent paper, Dai et al. [40] introduced a bug triaging framework based on graph collaborative filtering. The authors represent the correlations between bugs and developers using a bipartite graph, and initialize the bug nodes by pre-training them with natural language processing (NLP) based techniques. To learn the representation of developer nodes, they employ a spatial-temporal graph convolution strategy. Finally, they propose an information retrieval-based classifier that matches bugs and developers.

The above approaches have significant potential in assisting developers with effective bug management and optimizing the software development process. Our work presented in this paper investigated the effectiveness of NLP-based bug-feature engineering and provided an in depth investigation of domain specific word embedding w.r.t. automatic bug assignment to developers. We also provided a comparative study on both ML and DL-based approaches such as [1], [11], and [27], SVM [41], RF [42], NB [43], ELMo-CNN [15], ELM [42], CNN-DA [14], and DBRNN-A [13] that are in line with this. We found that our information fusion-based approach outperforms previous ML and DL models.

III. OUR APPROACH

This section formally states the bug assignment problem, datasets used, data preprocessing, and the proposed automatic bug assignment model.

A. FORMAL PROBLEM STATEMENT

In this study, we present bug assignment as a single-label classification problem. Let, bug reports $\mathcal{B} = \{B_{11}, B_{12} \ldots B_{ij}\}$ were previously assigned to the developers $\mathcal{D} = \{D_1, D_2 \ldots D_i\}$. Here, bug B_{ij} was assigned to the developer D_i, that is, B_{ij} represents the jth bug report of ith developer D_i. In terms of classification, it can be said that D_i is the label of B_{ij}. On that note, a multi-class classifier should be able to assign a developer $D_{i'} \in \mathcal{D}$ for a new bug report $B_{ij'}$ automatically if it is trained using the historical data $\mathcal{B}$ and $\mathcal{D}$. Here, the classifier conjectures that $B_{ij'}$ has textual similarities with the prior bug reports of $D_{i'}$. To validate this idea, a subset $\mathcal{B}_{train}$ of $\mathcal{B}$ can be used to train the classifier, and the rest of the bug reports $\mathcal{B}_{test}$ can be used to measure the performance of the classifier. Here, $\mathcal{B} = \mathcal{B}_{train} \cup \mathcal{B}_{test}$. The goal of the classifier is to identify the appropriate developers for the bugs in $\mathcal{B}_{test}$.

B. DATASETS

To evaluate our proposed model, we conduct experiments on eight benchmark datasets collected from open source projects. Moreover, these datasets are experimented with in the earlier studies [13], [15]. The Sun Firefox, JDT, Netbeans, GUO Firefox, and GCC datasets are publicly available here [44]. The rest of the datasets are available here [45]. Table 1 presents the summarized statistics of these datasets, e.g., the total number of developers, bug reports, and Min-Max bug reports per developers in a particular dataset. These statistics are calculated after the preprocessing step (Section III-C). Few bug reports are found empty after preprocessing, i.e., no textual description or containing only screenshots/hyperlinks of the bugs. Since we have not used these bug reports to train our model, they are discarded from Table 1. To compare our results with the existing studies [13], [15], we consider only those developers who solved at least ten bugs. The last column of Table 1 represents the Measure of Balance (MB) in a particular dataset. In the real world, datasets of bug assignment are imbalanced as bugs are assigned based on the availability or experience of developers. In such scenarios, the trade-off between imbalanced datasets and the model's performance might be analyzed using this MB. We utilize Shannon Entropy (SE) [46] to calculate MB as given as follows.

$$MB = \frac{SE}{\log k} = \frac{-\sum_{i=1}^{k} \frac{c_i}{n} \log \frac{c_i}{n}}{\log k} \tag{1}$$

Here, n is the total number of bug reports, k is the total number of unique developers, and c_i is the total number of bugs assigned to the ith developer. MB ranges between 0 and 1; 0 for an imbalanced dataset and 1 for a

TABLE 1. Bug assignment datasets.

Dataset	Description	#Developers	#Bug Reports	#Min-Max Reports	MB
Sun Firefox [15]	This dataset was compiled from bugs in Mozilla Firefox's sun theme	104	6065	10-800	0.8542
JDT [15]	This dataset was compiled from Eclipse's JDT components' bug reports	19	949	10-183	0.8758
Netbeans [15]	This dataset was gathered from the Netbeans editor's bug report	108	9716	10-1294	0.8400
GUO Firefox [15]	This dataset was compiled from bugs in the GUO theme of the Mozilla Firefox web browser	159	6981	10-474	0.8957
GCC [15]	This dataset was gathered from the GCC compiler's bug	33	1417	10-198	0.9024
Google Chromium [13]	This dataset was collected from Google Chromium's bugs	624	27700	10-490	0.9356
Mozilla Core [13]	This dataset was collected from the Mozilla's Core component's bug	610	64397	10-2241	0.8821
Mozilla Firefox [13]	This dataset was collected from the bugs of Mozilla Firefox web browser	100	136312	10-117489	0.1438
Mozilla Firefox++	A reduced version of Mozilla Firefox [13] dataset	99	18823	10-11098	0.4109

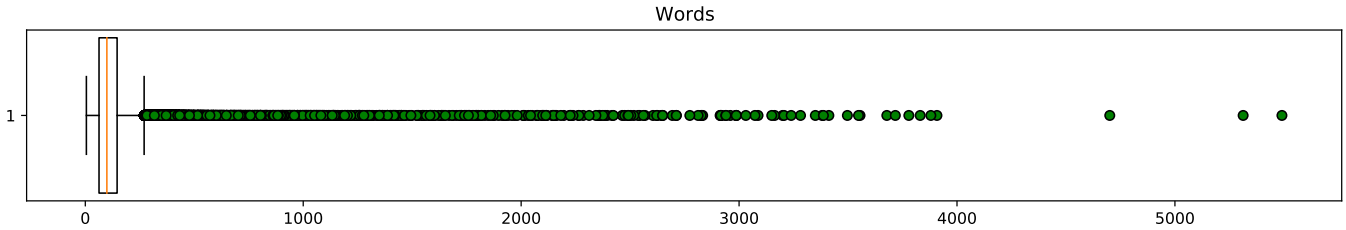


FIGURE 2. Distribution of words in the bug reports of Mozilla Firefox dataset.

balanced dataset. From Table 1, Google Chromium and GCC datasets are almost balanced as the values of MB are 0.9356 and 0.9024, respectively. In contrast, Mozilla Firefox is near to almost imbalance (MB is 0.1438) since a developer alone solved 117489 bug reports. To avoid bias in the experiments, we have created another dataset called Mozilla Firefox++ by removing the developer that has solved 117489 bugs and the associated bug reports from the original Mozilla Firefox dataset.

C. DATA PREPROCESSING

Since we use Natural Language Processing (NLP) techniques to solve the bug assignment problem, it is required to preprocess the data to achieve better performance. Besides, the datasets contain several types of noises (e.g., hyperlinks, newlines, and special characters) as we collect them from open-source projects. The overall steps of our data preprocessing are described below.

1) BASIC PREPROCESSING

A bug report consists primarily of a title and a description. It is worth noting that when the bug title is considered alongside the bug description, the accuracy of automated bug assignment improves [13]. As a result, we consider both in this study. Before training the model, we clean the title and description by converting them to lower case, removing special characters, URLs, hex codes, new lines and extra white spaces, removing stack traces, and truncating them. Except for the last, these preprocessing steps are common in existing works. The size of the bug reports in terms of total words is not uniformly distributed, as shown in Fig. 2. In comparison to the majority of reports, we find very few bug reports that contain too many words. Among the 136312 bug reports in the Mozilla Firefox dataset, only 339 have more than 1000 words. To that end, we truncate bug reports that are longer than 1000 words. Our hypothesis is that the main bug report is described in the first 1000 words, with the remaining words containing useless information such as error logs or the path to reproduce the bug.

2) TOKENIZATION

We tokenize the bug reports after performing the basic preprocessing in order to fit them into our model. As shown in Fig. 3, we use word-level tokenization for the deep embedding layer of our bug assignment model, which converts bug reports into space-separated sequences of words or tokens.

3) *n*-GRAM GENERATION

In this paper, we develop an automated bug assignment model based on the contextual information extracted from the

linguistics patterns of a bug report ('Title'+'Description'). In the fields of natural language processing and language understanding research, n-grams are a popular method for capturing linguistic patterns in a document [47]. We assume that once a bug is reported and resolved by a developer, the linguistic patterns in the bug reports will be captured by the n-grams. Furthermore, we assume that a developer will fix similar bugs. For example, developer X fixes security bugs while developer Y fixes user interface (UI) bugs. Thus, it is discovered that security-related keywords appear frequently in X's bug reports, whereas UI-related keywords appear frequently in Y's bug reports. For this reason, we generate overlapping n-grams from bug reports to distinguish developers based on keywords. We use 1-gram and 2-gram in this study since we observe them frequently. Furthermore, higher n-grams, such as 3 or 4-grams, may not be repeated among a developer's bug reports.

4) TF-IDF VECTORIZATION

After generating n-grams from a bug report, we perform Term Frequency-Inverse Document Frequency (TF-IDF) vectorization. It is seen that some n-grams are very frequent in a particular developer's bug reports while on the contrary, few n-grams are very rare. To this end, TF-IDF vectorization is performed in order to provide importance on the significant n-grams.

5) DOCUMENT-TERM MATRIX

To train a model, all training data or bug reports must be brought into the same dimension. To accomplish this, we generate a document-term matrix, $M_{br} \in^{r \times t}$. Here, the number of training documents is represented by row r, and the number of unique terms/n-grams across all training documents is represented by column t. If the r-th bug report contains the t-th n-gram, then $M_{br}[r, t] = S_{tf-idf}$, where S_{tf-idf} is the TF-IDF score of t-th n-gram.

6) DIMENSIONALITY REDUCTION

Since t equals the number of unique n-grams across all training documents, $M_{br} \in^{r \times t}$ becomes a high dimensional sparse matrix. In Mozilla Firefox, for example, the dimension of M_{br} is $136,312 \times 1.1M$. The problem here is that these $1.1M$ unique overlapping n-grams are not distributed evenly across the $136,312$ bug reports. As a result, a significant number of S_{tf-idf} in M_{br} are zero, resulting in M_{br} eventually becoming a sparse matrix. To avoid these minor S_{tf-idf} or n-grams, we intend to reduce the dimension of M_{br} using a well-known dimensionality reduction technique known as Principle Component Analysis (PCA) [48]. When PCA is applied to $M_{br} \in^{r \times t}$, it yields the lower-dimensional matrix $M'_{br} \in^{r \times t'}$. Here, t' is less than t, and it explains the majority of the variance in the original matrix $M_{br} \in^{r \times t}$. Another idea is to train our model with the significant features rather than all n-grams as features. Significant features in reduced space $^{t'}$ are referred to as components in the following Sections.

D. BUG ASSIGNMENT MODEL

As depicted in Fig. 3, our model takes the textual information of a bug report as input and performs rigorous data cleaning which is describes in Section III-C. Then, it captures repeating keywords and contextual relationship among words using Convolutional Neural Network (CNN) and Artificial Neural Network (ANN), respectively. After that, these two types of features are combined in Information Fusion (IF) layer. The IF layer is then followed by a dense and softmax layer to assign a developer to the given bug report. The components CNN and ANN modules, IF Layer and bug assignment are described in detail below.

Our proposed model for bug assignment (Fig. 3) is based on deep learning techniques. Unlike existing deep learning-based bug assignment techniques that focus on a single feature (contextual information), our technique assigns bugs based on two types of features: repeating keywords and contextual relationships between words. Although a developer is expected to solve various types of bugs, some keywords appear repeatedly in the textual descriptions of his bug reports. Based on this insight, we intend to use repeating keywords as features in our model.

1) CNN MODULE

This module aims to capture the contextual relationship between the words in a specific bug report, which plays an important role in uniquely identifying a developer. This module is consisted of the following layers: Embedding Layer, Convolutional Filter Layer, Feature Map Layer, Pooling Layer, and Flatten Layer.

- **Embedding Layer.** To find the contextual relationship among words, it is required to embed the words at first. After performing tokenization (Section III-C2) on the input B_{ij}, this layer is used to transform its text into an embedding matrix. Let, $v_e \in^d$ represents the eth word of B_{ij} where v_e is d dimensional vector. Hence, B_{ij} can be represented as an embedding matrix $M_{eb} \in^{n \times d}$ by vertically arranging all the transposes of vectors that represent the words of B_{ij}. Here, n denotes the number of words in a bug report, and d denotes the embedding dimension. It is unlikely that all bug reports will contain the same number of words. In those scenarios, padding should be applied if a bug report does not contain n number of words. The formal notation of $M_{eb} \in^{n \times d}$ is as follows.

$$M_{eb} = v_1^T \perp v_2^T \perp \cdots \perp v_n^T \tag{2}$$

  Unlike existing approaches [12], [15], weights of the above vectors are non-static, i.e., no pretrained word embedding such as Word2Vec [49] or GloVe [35] is used. These weights are learned from the domain-specific bug assignment dataset during training using backpropagation [50].
- **Convolutional Filter Layer.** To capture the relationship between consecutive words, convolutional filters

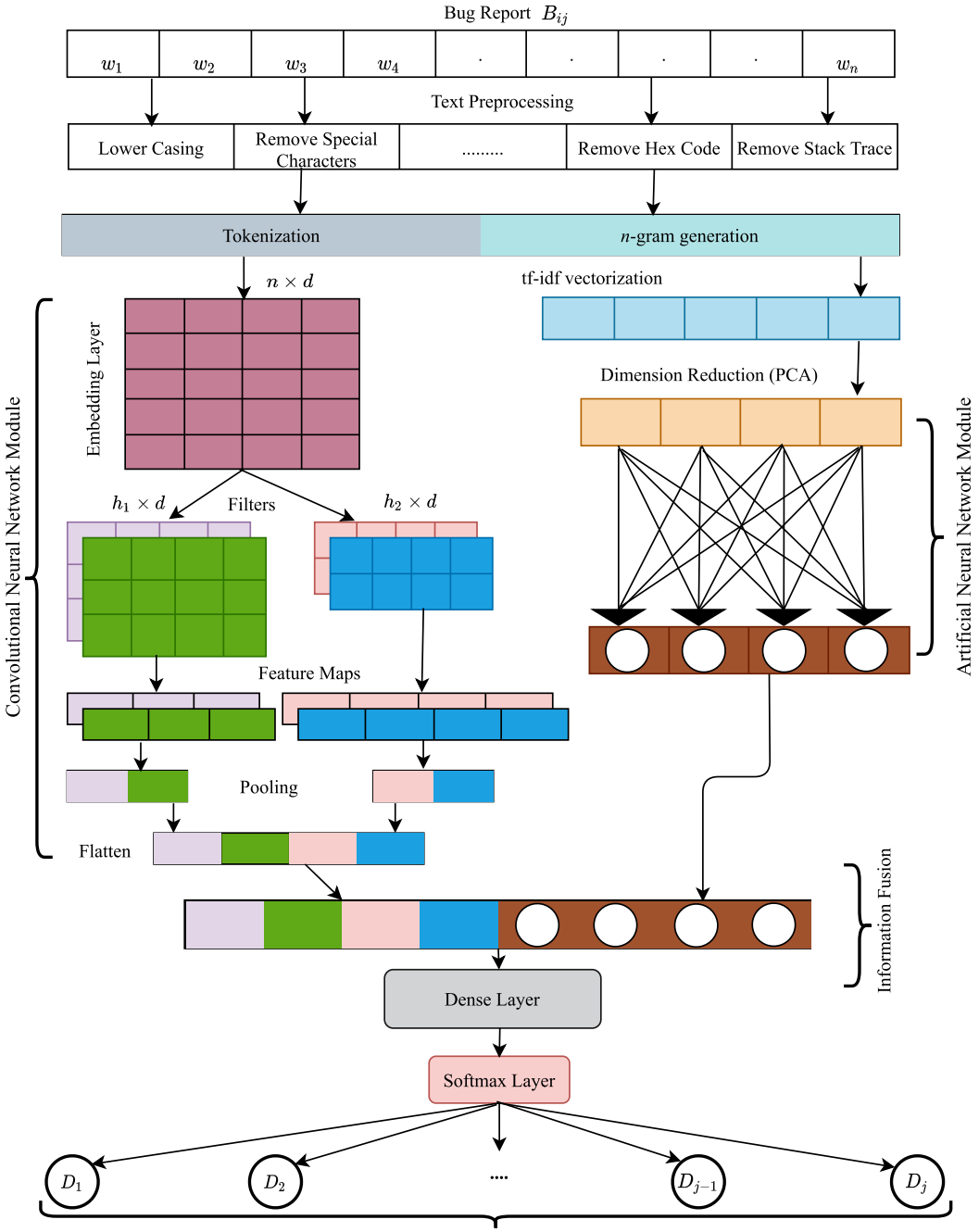


FIGURE 3. Architecture of the proposed information fusion based deep learning model for automatic bug assignment.

are used. These filters $W \in^{h \times d}$ are essentially applied on the embedding matrix such as M_{eb}. Here, h is the number of consecutive words to be used for capturing the relationship. It is often known as kernel or window size. For instance, if $h = 2$ is used, the filter $W \in^{2 \times d}$ will capture the contextual relationship between two adjacent words. When a filter $W \in^{h \times d}$ is applied in a particular window, it generates a new feature s_j as mentioned in equation 3. Here, $b_c \in$ is a bias term, g is a non-linear function and j is the index of the window.

$$s_j = g(W \cdot [v_e : v_{e+h-1}] + b_c) \tag{3}$$

- **Feature Map Layer.** If the same filters are applied to all possible windows of a particular bug report, a feature map $F \in^{n-h+1}$ that holds the contextual relationship between all words can be obtained. This feature map can be written in the format shown below. Here, f_j is the yield

of jth window s_j.

$$F = [f_1, f_2, \ldots, f_{n-h+1}] \tag{4}$$

- **Pooling Layer.** Depending on the value of n, the size of F will vary a lot. Hence, this layer is used to pool important features from F. For instance, the max-pooling operation can be used. Other pooling operations are min-pooling and average pooling. If a max-pooling operation is used, it will select maximum value $\hat{f}_j$ from each f_j, which can be written as follows.

$$\hat{F} = [max(f_1), max(f_2), \ldots, max(f_{n-h+1})] \tag{5}$$

- **Flatten Layer.** To increase feature coverage, different filters such as $W' \in^{h' \times d}$ and $W'' \in^{h'' \times d}$ can be used. In such cases, there will be multiple feature maps after the pooling operation, such as $\hat{F}'$ and $\hat{F}''$. The main function of this layer is to aggregate these $\hat{F}'$ and $\hat{F}''$. Moreover, this layer is the ending of contextual feature capturing. The outcome $V_{cnn} \in^{p}$ of this layer can be shown as follows, where p is the number of filters.

$$V_{cnn} = aggregate(\hat{F}', \hat{F}'', \ldots \hat{F}^{p\prime}) \tag{6}$$

2) ANN MODULE

This module is responsible for capturing repeating keywords from the bug report. It consists of two layers such as input and hidden layers. An overview of them is given below.

- **Input Layer.** This layer depends on the n-grams of B_{ij} to extract meaningful information from repeating keywords. An input vector $v_j \in^{t}$ is constructed using TF-IDF scores of the n-grams. Here, t is the total number of unique n-grams. In real-world bug reports, there exists a large number of unique n-grams. They might not repeat equally among all bug reports. This is why, we propose to reduce the dimension of $v_j \in^{t}$ and select significant repeating features only. Let, $v'_j \in^{t'}$ denotes the reduced version of $v_j \in^{t}$. Here, v'_j is the jth row of $M'_{br} \in^{r \times t'}$.
- **Hidden Layer.** The main objective of this layer is to assign different weights to each element of $v_j \in^{t'}$. We propose to use different weights as all the elements of v_j may not play an equal role in uniquely identifying a developer. When such weight vector $\mathcal{W} \in t'$ is assigned to v_j, it generates a new vector V_{ann} which can be expressed as follows. Here, $b_a \in$ is the bias, and $\mathcal{F}$ is a non-linear function.

$$V_{ann} = \mathcal{F}(\mathcal{W} \cdot v_j + b_a) \tag{7}$$

3) INFORMATION FUSION LAYER AND BUG ASSIGNMENT

After obtaining contextual relationship (V_{cnn}) and repeating keywords (V_{ann}) from CNN and ANN modules, this layer fuses these features to perform better bug assignment. Since we have used two different types of features here, we call this layer the Information Fusion (IF) layer. The outcome of this layer $V_{if} \in^{p+t'}$ can be denoted as follows. Here, $\oplus$ is the concatenation operator.

$$V_{if} = V_{cnn} \oplus V_{ann} \tag{8}$$

After fusing these features, $V_{if} \in^{p+t'}$ is sent to a non-linear dense layer which assigns different weights to each element of V_{if}. Since V_{if} contains different types of features, all features may not contribute equally to assigning the bug reports. Thus, we assign different weights to each element of V_{if}. Finally, we use an output layer to transform the hidden layer's outcomes into probabilities. This layer assigns different probabilities to each developer. However, the original developer gets the highest score. Based on this probability score, a bug report is assigned to a developer. For instance, a bug report B_{ij} is passed to our model, which is the jth bug report of ith developer D_i. The model assigns the highest probability score to developer D_i compared to other developers.

IV. EXPERIMENTS

In this section, we discuss our experiments in detail, such as benchmark models and their training and evaluation. Moreover, we answer the following key research questions (RQ#) to evaluate the performance of the proposed automatic bug assignment model.

RQ1: Can TF-IDF vectorization of a bug report and an artificial neural network (ANN) model effectively automate the bug assignment?

RQ2: Can a convolutional neural network (CNN) model with non-static embedding effectively automate the bug assignment?

RQ3: Can we develop an effective bug assignment model by fusing both artificial and convolutional neural networks' features?

A. BENCHMARK MODELS

As stated earlier that our proposed model consists of two modules: Artificial Neural Network (ANN) and Convolutional Neural Network (CNN). However, we perform an ablation [51] study to justify whether it is possible to achieve the same performance by removing any of these modules. For this reason, we create the following combination of benchmark models from our proposed bug assignment model (Fig. 3).

- **ANN + CNN2 + CNN3:** This model consists of a single ANN layer and two CNN layers having different filters. For instance, CNN2 and CNN3 capture the contextual relationship between 2 and 3 words. Then, we use max-pooling to extract the significant and meaningful features from CNN layers. The rest of behaviors are the same as described in Section III-D. The target of this model is to observe the performance of Information Fusion (IF).
- **ANN + CNN2:** This model aims to observe the performance of IF. However, it has fewer filters than the above benchmark model. It is developed by removing CNN3 from the above ANN + CNN2 + CNN3 model.

- **ANN:** In this model, there is no information fusion, i.e., it only learns from repeating keywords. This benchmark model is developed by removing CNN2 and CNN3 from ANN + CNN2 + CNN3. This model aims to assign bugs without IF. This model takes repeating keywords (TF-IDF vectors) as input and assigns weights to the input in its dense layer. Finally, a softmax layer assigns the bug reports.
- **CNN2 + CNN3:** This model learns from the contextual relationship of consecutive words. Again, for instance, CNN2 and CNN3 capture contextual relationships between 2 and 3-words, respectively. Then, the most important features are selected using the max-pooling operation. The outcome of the pooling operation is processed via a dense and softmax layer to assign bug reports. This model is developed by removing ANN from ANN + CNN2 + CNN3.
- **CNN2 and CNN3, individually:** As CNN2 + CNN3 uses the contextual relationship between both 2 and 3-words, we develop further two individual models such as CNN2 and CNN3. These models aim to find whether it is 2 or 3 consecutive words that should be the choice for CNN layers.

Our benchmark models can be grouped into two categories such as independent models (ANN, CNN2, CNN3, and CNN2 + CNN3) and fusion-based models (ANN + CNN2 + CNN3 and ANN + CNN2). Apart from these benchmark models, we train traditional ML models such as SVM [41], RF [42] and NB [43] to compare our DL models' results against them. To train these models, we use tf-idf vectors (reduced using PCA) as features. In other words, our ANN and ML models are trained using the same set of features. We consider these models because we find that they were widely used in existing studies for solving the bug assignment problem [1], [11], [27].

Lastly, we compare our results against some existing benchmark models such as ELMo-CNN [15], ELM [42], CNN-DA [14], and DBRNN-A [13]. ELMo-CNN is a DL-based technique that uses a convolutional neural network with a pre-trained word embedding named ELMo to assign bugs. It achieved the best Top-1 accuracy in Sun Firefox, JDT, and GUO Firefox datasets. In the GCC dataset, the extreme learning machine (ELM) achieved the best Top-1 accuracy. In other datasets, DBRNN-A, an attention-based deep bidirectional recurrent neural network achieved the best Top-10 accuracy.

B. MODEL TRAINING AND EVALUATION

To train and evaluate our benchmark models, we apply 10-Fold cross-validation followed in the existing studies [13], [15]. In each fold, the performance of a particular model is measured using Top-1 to Top-10 accuracies so that we can compare our results with earlier works of bug assignment [13], [15]. Top-1 to Top-10 accuracy is a well-known metric in recommender systems. Automated bug assignment and recommender systems are pretty comparable. Using Top-1 to Top-10 accuracies for bug assignment makes sense in the following ways. Assume that our model uses Top-1 accuracy, in which case it recommends a specific developer for a given bug report. The developer in question, though, might be preoccupied with other projects. In this case, top-k recommended developers may be a good option. To train all of our benchmark models, we use a variety of settings, which are summarized below.

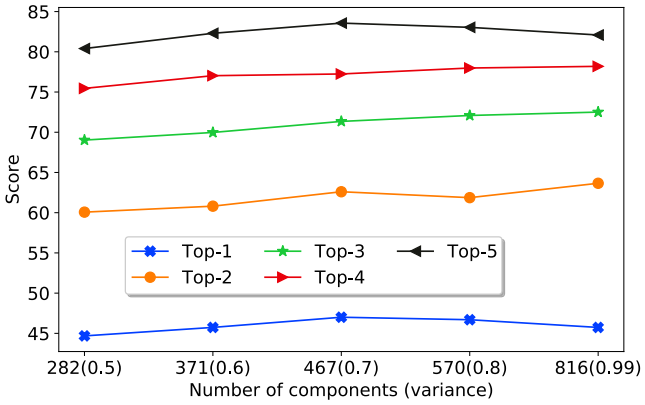


FIGURE 4. JDT Dataset: Effect of the number of components (variance %) on ANN based model's performance such as Top-1, Top-2, Top-3, Top-4 & Top-5 accuracy.

1) SETTINGS OF INDEPENDENT MODELS

To train the ANN model, training documents/bug reports are initially transformed into $M'_{br} \in^{r \times t'}$. Here, r is the total number of training documents. The value of number of components t' needs to be chosen carefully because it has major impact on the model's performance. During the training of this model, we try different values of t' to find optimum performance. Fig. 4 depicts different values of t' and their corresponding Top-1 to Top-5 accuracies in JDT dataset. It can be observed from Fig. 4 that same value of t does not provide optimum scores for all accuracies (Top-1 to Top-5). For instance, the optimum score of Top-1 accuracy can be obtained using 467 components whereas it is required to use 816 components for Top-2 accuracy. Here, 467 & 816 components explains 70% & 99% variance of the original data. Thus, it is also difficult to choose the value of t' based on the variance explained. For these reasons, we tune the value of t' to train our ANN model until we obtain optimum scores for all accuracies (Top-1 to Top-10).

In CNN2 & CNN3 training, 300 embedding dimension is used, i.e., each word is represented using a 300-dimensional vector. Moreover, we use 256 filters of kernel sizes 2×300 and 3×300, respectively. We train our CNN2 + CNN3 model by combining the settings of CNN2 and CNN3. We also try more than 256 filters, however, it does not improve the performance of these models. In an existing study [15], Zaidi et al. also used the same number of filters to train their CNN-based bug assignment model.

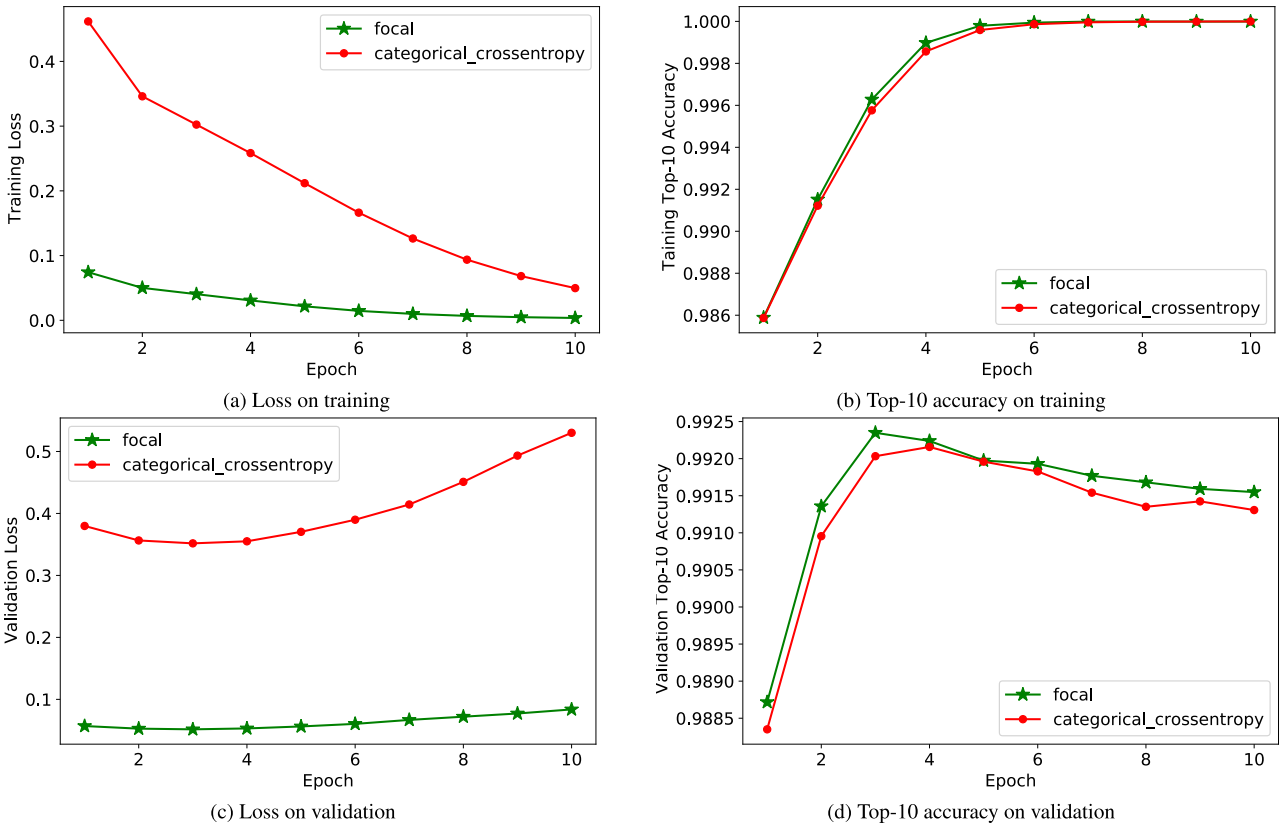


FIGURE 5. History of ANN + CNN2 + CNN3 model training over Mozilla Firefox dataset using focal & categorical_crossentropy loss (a) Comparison of total losses during training (b) Comparison of Top-10 accuracies during training (c) Comparison of total losses during validation (d) Comparison of Top-10 accuracies during validation.

2) SETTINGS OF FUSION-BASED MODELS

To train ANN + CNN2, we borrow the setting of ANN and CNN2 since we achieved the optimum performance for those individual settings. Similarly, the parameter settings of ANN and CNN2 + CNN3 are used to train ANN + CNN2 + CNN3.

3) COMMON SETTINGS

Apart from these settings, we use the same batch size, number of epochs, loss function and optimizer to train all the benchmark models. We find optimum performance of these models using 32 batch size and setting the required epochs to 4. It is needed to mention that we also try higher batch size (64) and epochs (10). However, it did not help to achieve better performance, i.e., validation accuracy (Fig. 5d) starts decreasing if more than 4 epochs are used. To optimise the loss of any multi-class classifier, the use of categorical_crossentropy [52] is seen to be very common among deep learning-based classifier. However, we use state-of-the-art focal loss [53] during model trainings to gain performance over imbalance datasets. Fig. 5 shows comparison between categorical_crossentropy and focal loss. From this figure, it is evident that the optimization of focal loss is better than categorical_crossentropy. For instance, focal loss minimizes the total losses significantly during training and validation compared to categorical_crossentropy (Fig. 5a and Fig. 5c). On the other hand, focal loss maximizes the training and validation accuracies (Fig. 5b and Fig. 5d). Although Fig. 5 shows the training history of our ANN + CNN2 + CNN3 model over Mozilla Firefox only, we also observe same patterns for all benchmark models in the rest of the datasets. In Convolutional Filter and Hidden Layers, we use a commonly used activation function such as Rectified Linear Units (ReLU) [54]. Lastly, we use the *softmax* function in output layer as bug assignment is multi-class classification problem.

4) ENVIRONMENT

We conduct experiments in a Windows 10 operating system equipped with 12 core 3.80 GHz processor, 64 GB RAM, and 8 GB VRAM. We implement the Deep learning (DL) models using Keras (https://keras.io) framework. The traditional machine learning (ML) models are implemented using the scikit-learn (https://scikit-learn.org) library.

V. RESULT AND DISCUSSION

Table 2 displays the experimental results we obtain using our deep learning-based benchmark models. Table 3 shows the results obtained by applying traditional ML models. In addition to these results, we present a comparative study between

our benchmark models, existing deep learning-based models, and traditional ML models in Table 4 and Table 5, respectively. The last columns of Table 4 and Table 5 represent the average improvements our benchmark models achieve compared to existing deep learning and ML-based models. Lastly, we answer the following RQs by analyzing the above results.

A. ANSWERS TO THE RQs

1) RQ1

Can TF-IDF vectorization of a bug report and an artificial neural network (ANN) model effectively automate the bug assignment? Although previsous researchers used TF-IDF for feature extraction [31], [41], [42], [43], none of the existing works has explored ANN for bug assignment. To our best knowledge, this is the first study that uses significant repeating keywords (TF-IDF vectors after PCA) to assign bugs using an ANN.

Our experimental results demonstrate that ANN performs better than traditional ML models. Top-K accuracies of ANN and traditional ML models are displayed in Table 2 and Table 3, respectively. Most importantly, ANN outperforms existing deep learning-based benchmark models such as ELMo-CNN [15], CNN-DA [14], DBRNN-A [13] in JDT, GUO Firefox, Google Choromium, Mozilla Core and Mozilla Firefox datasets. The last column of Table 4 shows the average improvement of ANN-based model compared to these existing benchmark models. Moreover, ANN based model shows comparable performance with the fusion-based models such as ANN + CNN2 and ANN + CNN2 + CNN3. Table 2 displays the results of ANN, ANN + CNN2 and ANN + CNN2 + CNN3 in all datasets. From these observations, it is evident that the idea of TF-IDF vectorization and ANN to assign bugs is non-negligible. Therefore, we conclude that ANN-based model can be an effective way to automate bug assignment.

2) RQ2

Can a convolutional neural network (CNN) model with non-static embedding effectively automate the bug assignment? Based on the experimental results in Table 2, CNN-based models (CNN2, CNN3, CNN2+CNN3) that use contextual relationship among adjacent words does not perform well compared to our other benchmark models such as ANN, ANN + CNN2, and ANN + CNN2 + CNN3. However, our CNN-based models with non-static word embedding outperform existing CNN-based benchmark models that use pre-trained word embedding. For example, ELMo-CNN achieved the best Top-1 accuracy in Sun Firefox (31.01%), Netbeans (40.17%), and GUO Firefox (16.73%) according to Zaidi et al. [15]. On the contrary, our CNN-based models achieve 34.74% (CNN2 + CNN3), 54.17% (CNN2 + CNN3) and 30.23% (CNN2) Top-1 accuracy, respectively. These results are also better than the other CNN-based models of Zaidi et al. [15], such as Word2Vec-CNN and GloVe-CNN. Our major finding from this RQ is that non-static embedding layers perform better than other pre-trained embedding layers.

3) RQ3

Can we develop an effective bug assignment model by fusing both artificial and convolutional neural networks' features? To answer this RQ, we compare our fusion-based models (ANN + CNN2 and ANN + CNN2 + CNN3) with our other benchmark models (ANN, CNN2, CNN3, and CNN2 + CNN3) and existing deep learning-based benchmark models (ELMo-CNN, CNN-DA, and DBRNN-A). As shown in Table 4 and Table 5, our fusion-based models outperform our other benchmark models and traditional ML models in Sun Firefox, Netbeans, and GCC datasets. In other datasets, the performance of fusion-based models is comparable with our other benchmark models which are shown in Table 2. However, fusion-based models significantly outperform existing deep learning-based models almost in all datasets. According to Zaidi et al. [15], ELMo-CNN and ELM are the best performing models in Sun Firefox, JDT, Netbeans, GUO Firefox and GCC datasets which are displayed in the 3rd column of Table 4. From Table 2, it can be observed that the performance of ELMo-CNN and ELM is beatable by ANN + CNN2 and/or ANN + CNN2 + CNN3. According to Mani et al. [13], DBRNN-A is the best performing model in terms of Top-10 accuracy in Google Chromium (41.80%), Mozilla Core (36.10%) and Mozilla Firefox (44.80%) datasets. In contrast, our fusion-based models achieve better Top-10 accuracies in these datasets, i.e., 51.48% (ANN + CNN2), 58.89% (ANN + CNN2) and 98.53% (ANN + CNN2 + CNN3), respectively. Lastly, the main finding from this RQ is that the performance of fusion-based deep learning models is more promising than other models that do not fuse information to assign bug reports.

B. DEEP ERROR DEBUGGING AND FUTURE DIRECTION

As our fusion-based models (ANN + CNN2 and ANN + CNN2 + CNN3) performed better compared to the existing deep learning-based models (ELMo-CNN and DBRNN-A) over all datasets, we debug the errors of one of these models (ANN + CNN2) to improve their performance further. To this end, we observe Top-1 accuracy and note the incorrect bug assignments during 1 to 10 folds. Then, we analyze these data and find a coinciding behavior among these incorrect bug assignments. For instance, ANN + CNN2 predicted 12 times developer D_3's bugs as D_{11}'s and 18 times D_{11}'s bugs as D_3's. So, the total number of coinciding incorrect bug assignments between D_3 and D_{11} equals 30 times. Fig. 6d displays the total number of coinciding incorrect bug assignments among all developers in the JDT dataset. In this dataset, the top most coinciding developers in terms of our model's incorrect assignments are $\{D_8, D_{16}\} = 46$, $\{D_8, D_{11}\} = 31$, $\{D_3, D_{11}\} = 30$ and $\{D_8, D_{14}\} = 27$. To understand the causes of coinciding incorrect assignment, we measure the similarities among developers in terms of their historical bug

TABLE 2. Average Top-1 (%) to Top-10 (%) accuracies of our models in all datasets.

Dataset	Model	Top-1	Top-2	Top-3	Top-4	Top-5	Top-6	Top-7	Top-8	Top-9	Top-10
Sun Firefox	ANN+CNN2+CNN3	**36.56**	**48.29**	55.54	60.46	64.15	67.40	69.71	71.77	73.62	75.17
	ANN+CNN2	36.41	47.71	55.08	60.34	64.25	67.33	69.96	72.28	74.13	75.73
	ANN	34.92	48.19	**56.88**	**63.25**	**66.83**	**70.09**	**72.64**	**74.91**	**76.75**	**78.51**
	CNN2+CNN3	34.74	46.67	53.49	58.66	62.72	65.95	68.43	70.86	73.03	74.56
	CNN3	31.64	43.05	50.17	55.45	59.39	62.77	65.57	67.85	69.68	71.64
	CNN2	33.62	45.94	53.85	58.83	63.09	66.12	68.64	70.77	73.04	74.84
JDT	ANN+CNN2+CNN3	45.75	61.02	69.45	75.88	80.19	82.62	85.88	88.41	90.2	91.89
	ANN+CNN2	42.88	59.64	66.92	73.03	77.35	80.52	83.46	85.88	88.83	90.20
	ANN	**47.53**	**62.71**	**72.08**	**77.98**	**82.62**	**86.09**	**89.15**	**90.93**	**92.31**	**93.36**
	CNN2+CNN3	42.68	59.75	69.44	77.25	80.83	83.98	86.31	88.52	91.57	92.84
	CNN3	39.62	54.69	66.59	74.51	79.25	84.51	87.15	88.52	89.99	91.78
	CNN2	40.15	58.17	68.28	75.34	80.40	83.88	86.93	88.62	90.83	92.52
Netbeans	ANN+CNN2+CNN3	**57.77**	**70.16**	75.89	79.26	81.83	83.64	85.21	86.41	87.71	88.45
	ANN+CNN2	57.20	69.80	**76.30**	**79.69**	**82.24**	**84.16**	**85.63**	**86.72**	**87.89**	**88.69**
	ANN	51.59	64.56	71.71	75.88	78.73	81.17	82.96	84.29	85.62	86.61
	CNN2+CNN3	54.17	66.78	73.54	77.67	80.40	82.23	83.76	85.19	86.46	87.63
	CNN3	52.46	64.98	71.22	75.35	78.18	80.17	81.78	83.24	84.48	85.47
	CNN2	53.68	66.34	72.49	76.54	79.09	81.22	83.11	84.61	85.70	86.67
GUO Firefox	ANN+CNN2+CNN3	33.33	46.09	52.92	57.72	61.53	64.57	67.05	69.23	70.89	72.48
	ANN+CNN2	33.36	**46.43**	**54.19**	**58.95**	62.79	65.43	68.01	70.08	71.97	**73.52**
	ANN	**33.50**	46.38	53.81	58.91	**62.81**	**65.66**	**68.22**	**70.39**	**71.99**	73.14
	CNN2+CNN3	30.00	42.24	49.59	54.73	58.85	61.75	64.44	66.76	68.75	70.49
	CNN3	28.31	39.74	46.89	52.37	55.96	58.85	61.63	64.17	65.89	67.51
	CNN2	30.23	42.84	50.41	55.77	59.45	62.39	65.02	67.34	69.33	70.83
GCC	ANN+CNN2+CNN3	49.89	61.74	69.02	73.53	**77.77**	**80.73**	**82.71**	**84.54**	**85.95**	**87.43**
	ANN+CNN2	**53.63**	**63.87**	**70.22**	**74.59**	77.49	80.10	82.15	83.91	85.47	86.74
	ANN	44.03	51.72	57.02	60.55	63.16	66.05	68.31	70.14	72.61	75.01
	CNN2+CNN3	47.71	60.05	67.61	72.27	76.43	79.33	81.44	83.27	84.69	86.73
	CNN3	46.58	58.43	65.91	70.36	74.09	77.13	79.32	81.58	83.98	85.67
	CNN2	50.03	62.67	68.18	73.12	76.51	79.19	82.08	83.69	85.11	86.52
Google Choromium	ANN+CNN2+CNN3	16.75	25.08	30.7	35.06	38.56	41.43	43.89	46.13	48.05	49.71
	ANN+CNN2	20.07	28.74	34.29	38.81	41.92	44.45	46.49	48.38	49.99	51.48
	ANN	**21.33**	**31.36**	**37.89**	**42.62**	**46.36**	**49.38**	**52.03**	**54.24**	**56.18**	**57.80**
	CNN2+CNN3	14.88	22.88	28.35	32.22	35.58	38.28	40.61	42.62	44.46	46.15
	CNN3	14.38	22.03	27.19	31.29	34.49	37.19	39.42	41.38	43.19	44.87
	CNN2	15.85	23.72	29.03	33.20	36.58	39.21	41.66	43.69	45.53	47.05
Mozilla Core	ANN+CNN2+CNN3	20.68	30.19	36.44	41.20	44.9	47.98	50.60	52.68	54.51	56.24
	ANN+CNN2	22.48	32.59	39.05	43.76	47.55	50.54	53.06	55.26	57.16	58.89
	ANN	**25.73**	**36.41**	**43.27**	**48.29**	**52.17**	**55.29**	**57.82**	**60.07**	**62.13**	**63.84**
	CNN2+CNN3	18.79	28.12	34.25	38.83	42.42	45.28	47.72	49.89	51.83	53.49
	CNN3	18.48	27.85	33.93	38.54	42.02	45.05	47.65	49.81	51.75	53.44
	CNN2	19.69	29.14	35.37	39.93	43.72	46.61	49.19	51.28	53.18	54.83
Mozilla Firefox	ANN+CNN2+CNN3	89.03	94.96	97.27	97.75	97.98	98.16	98.29	98.39	98.47	98.53
	ANN+CNN2	88.82	94.89	97.18	97.71	98.00	98.21	98.35	98.43	98.48	98.51
	ANN	**90.13**	**95.96**	**97.88**	**98.45**	**98.74**	**98.93**	**99.01**	**99.12**	**99.18**	**99.24**
	CNN2+CNN3	88.56	94.71	97.07	97.65	97.94	98.15	98.28	98.41	98.49	98.54
	CNN3	87.51	94.56	97.17	97.65	97.94	98.16	98.31	98.39	98.47	98.54
	CNN2	88.80	95.03	97.39	97.78	98.09	98.22	98.39	98.49	98.53	98.55
Mozilla Firefox++	ANN+CNN2+CNN3	66.10	83.25	87.67	89.52	90.94	91.80	92.51	93.04	93.40	93.73
	ANN+CNN2	67.45	83.59	87.76	89.63	90.86	91.82	92.49	93.02	93.42	93.69
	ANN	69.21	84.54	88.73	90.71	92.06	92.76	93.36	93.93	94.35	94.72
	CNN2+CNN3	66.36	82.72	87.25	89.31	90.70	91.42	92.04	92.63	93.08	93.35
	CNN3	63.69	82.10	86.66	88.64	89.84	90.66	91.31	91.86	92.29	92.63
	CNN2	65.81	82.47	87.00	89.09	90.20	90.97	91.59	92.03	92.45	92.85

reports. Let B_p be the profile of pth developer D_p that contains all historical bug reports of D_p. Here, we formulate B_p as follows, where b_{pj} denotes the jth bug report of pth developer.

$$B_p = b_{p1} \oplus b_{p2} \oplus \cdots \oplus b_{pj} \tag{9}$$

To find the similarities among developers, we transform the developers' profiles into vector spaces. Initially, we generate unique 1 and 2 grams from B^g where we express B^g formally as follows.

$$B^g = B_1 \oplus B_2 \oplus \cdots \oplus B_p \tag{10}$$

Concerning the unique n-grams of B^g, we generate a TF-IDF vector for each developer's profile. Then, we obtain a developer-term matrix $M_{DT} \in^{p \times t}$ by stacking all developers' TF-IDF vectors. Here, p and t denote the total developers and unique terms/n-grams, respectively. To keep in accordance with model training and validation, we apply PCA on $M_{DT} \in^{p \times t}$ and obtain $M'_{DT} \in^{p \times t'}$. The pth row of $M'_{DT} \in^{p \times t'}$ represents the pth developer's profile, denoted by D^v_p, in a vector space. Now, we can measure similarities among developers' profiles using Cosine Similarities. Formula 11 denotes

TABLE 3. Average Top-1 (%) to Top-10 (%) accuracies of traditional ML models in all datasets.

Dataset	Model	Top-1	Top-2	Top-3	Top-4	Top-5	Top-6	Top-7 (%)	Top-8	Top-9	Top-10
Sun Firefox	SVM	**34.26**	**48.36**	**57.32**	**64.12**	**68.37**	**72.02**	**74.62**	**76.75**	**78.43**	**79.78**
	RF	28.96	39.25	48.13	53.54	57.05	60.18	63.00	65.18	67.02	68.52
	NB	11.34	17.43	22.75	28.33	34.41	39.11	43.61	47.58	51.67	54.62
JDT	SVM	**45.83**	**62.60**	**73.24**	**78.92**	**83.35**	**86.73**	**89.36**	**90.94**	**92.52**	**93.57**
	RF	34.35	52.16	62.38	68.81	75.03	79.03	82.93	85.56	88.41	90.62
	NB	4.85	54.40	70.29	84.52	89.78	93.36	94.84	96.63	97.58	98.63
Netbeans	SVM	**48.96**	**62.36**	**69.42**	**73.71**	**76.80**	**79.29**	**81.40**	**83.18**	**84.73**	**85.99**
	RF	38.40	48.44	54.79	60.39	64.42	67.84	70.34	72.81	74.77	76.50
	NB	18.45	25.83	30.58	35.29	39.54	43.54	47.10	50.15	52.90	55.37
GUO Firefox	SVM	**32.27**	**44.69**	**52.15**	**57.26**	**61.31**	**64.33**	**66.72**	**69.11**	**71.39**	**72.86**
	RF	27.18	35.51	41.64	46.06	49.89	52.74	55.43	57.64	59.71	61.88
	NB	14.56	21.38	25.79	29.36	32.64	35.76	39.46	42.61	45.32	47.45
GCC	SVM	**53.42**	**64.71**	**71.98**	**76.29**	**79.88**	**82.85**	**85.74**	**87.79**	**89.06**	**90.61**
	RF	40.51	48.48	54.62	59.07	63.51	67.68	70.85	73.74	76.49	78.96
	NB	3.31	85.81	86.87	87.29	87.72	88.07	88.21	88.35	88.99	89.55
Google Choromium	SVM	**17.83**	**27.29**	**32.45**	**35.88**	**38.95**	**41.76**	**44.07**	**46.78**	**48.48**	**50.00**
	RF	16.24	21.98	26.17	29.21	32.16	33.97	35.77	37.51	38.88	40.51
	NB	14.87	24.80	30.54	35.27	38.48	40.79	42.74	44.69	46.67	48.19
Mozilla Core	SVM	**22.25**	**32.88**	**39.31**	**44.06**	**47.87**	**50.52**	**53.19**	**55.63**	**57.50**	**59.37**
	RF	14.08	21.45	26.45	30.41	33.35	35.76	38.15	40.01	41.97	43.63
	NB	13.11	22.34	27.84	31.75	35.06	37.29	39.75	41.42	43.24	44.79
Mozilla Firefox	SVM	**89.49**	**95.49**	**97.49**	**98.17**	**98.48**	**98.67**	**98.84**	**98.92**	**99.02**	**99.17**
	RF	87.78	94.75	97.23	97.93	98.34	98.55	98.72	98.91	99.06	99.23
	NB	48.45	69.17	80.56	85.97	88.78	90.44	91.47	92.19	92.79	93.33
Mozilla Firefox++	SVM	69.34	83.92	88.24	90.20	91.52	92.41	93.07	93.59	94.07	94.41
	RF	61.52	78.67	83.25	86.11	87.93	88.84	89.45	89.94	90.33	90.73
	NB	35.18	55.35	67.15	74.38	78.63	81.46	83.38	84.88	86.12	87.13

TABLE 4. Performance comparison between our DL/ML models and existing DL models in all datasets.

Dataset	Our Best DL/ML Model			Existing DL Model			Improvement In DL		
	Top-1	Top-5	Top-10	Top-1	Top-5	Top-10	Top-1	Top-5	Top-10
Sun Firefox	36.56 (ANN+CNN2+CNN3)	66.83 (ANN)	75.73 (ANN)	31.01 (ELMo-CNN)	57.96 (ELMo-CNN)	67.9 (ELMo-CNN)	17.900	15.30	11.530
JDT	47.53 (ANN)	82.62 (ANN)	93.36 (ANN)	49.75 (ELMo-CNN)	78.26 (ELMo-CNN)	87.23 (ELMo-CNN)	00.000	05.57	07.030
Netbeans	57.77 (ANN+CNN2+CNN3)	82.24 (ANN+CNN2)	88.69 (ANN+CNN2)	40.17 (ELMo-CNN)	66.07 (ELMo-CNN)	74.89 (CNN-DA)	43.810	24.47	18.430
GUO Firefox	33.50 (ANN)	62.81 (ANN)	73.52 (ANN+CNN2)	16.73 (ELMo-CNN)	36.47 (ELMo-CNN)	45.40 (ELMo-CNN)	100.24	72.22	61.940
GCC	53.63 (ANN+CNN2)	77.77 (ANN+CNN2+CNN3)	87.43 (ANN+CNN2+CNN3)	63.30 (ELM)	74.02 (ELMo-CNN)	83.20 (ELMo-CNN)	00.000	05.07	05.080
Google Choromium	21.33 (ANN)	46.36 (ANN)	57.8 (ANN)	…	…	41.80 (DBRNN-A)	…	…	38.280
Mozilla Core	25.73 (ANN)	52.17 (ANN)	63.84 (ANN)	…	…	36.10 (DBRNN-A)	…	…	76.840
Mozilla Firefox	90.13 (ANN)	98.74 (ANN)	99.24 (ANN)	…	…	44.80 (DBRNN-A)	…	…	121.51
Mozilla Firefox++	69.21 (ANN)	92.76 (ANN)	94.72 (ANN)	…	…	…	…	…	…

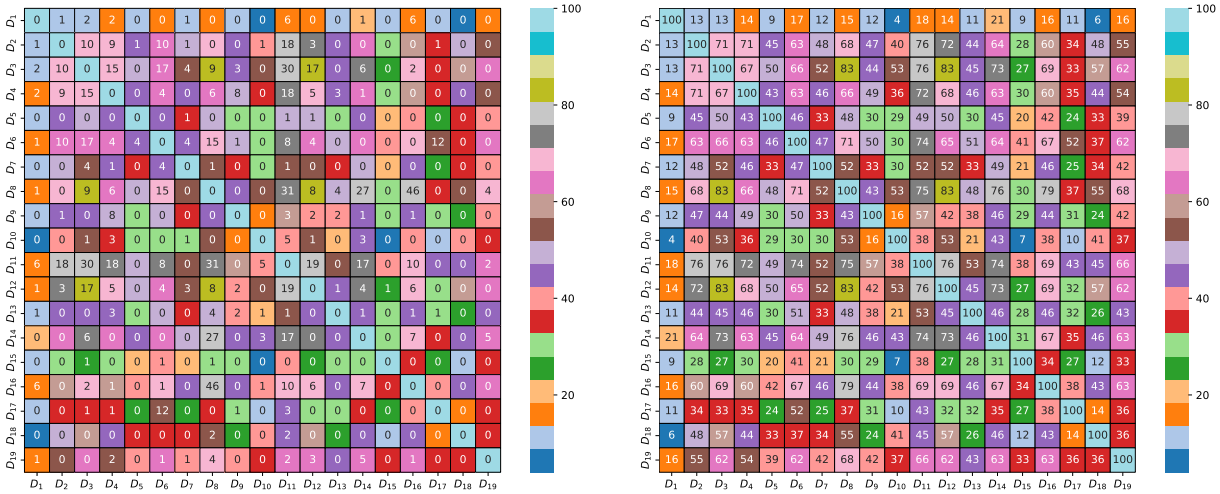
TABLE 5. Performance comparison between our DL/ML models and traditional ML models in all datasets.

Dataset	Our Best DL/ML Model			Traditional Best ML Model			Improvement In ML		
	Top-1	Top-5	Top-10	Top-1	Top-5	Top-10	Top-1	Top-5	Top-10
Sun Firefox	36.56 (ANN+CNN2+CNN3)	66.83 (ANN)	75.73 (ANN)	34.26 (SVM)	68.37 (SVM)	79.78 (SVM)	06.71	00.00	00.00
JDT	47.53 (ANN)	82.62 (ANN)	93.36 (ANN)	45.83 (SVM)	83.35 (SVM)	93.57 (SVM)	03.71	00.00	00.00
Netbeans	57.77 (ANN+CNN2+CNN3)	82.24 (ANN+CNN2)	88.69 (ANN+CNN2)	48.96 (SVM)	76.8 (SVM)	85.99 (SVM)	17.99	07.08	03.14
GUO Firefox	33.50 (ANN)	62.81 (ANN)	73.52 (ANN+CNN2)	32.27 (SVM)	61.31 (SVM)	72.86 (SVM)	03.81	02.45	00.91
GCC	53.63 (ANN+CNN2)	77.77 (ANN+CNN2+CNN3)	87.43 (ANN+CNN2+CNN3)	53.42 (SVM)	79.88 (SVM)	90.61 (SVM)	00.39	00.00	00.00
Google Choromium	21.33 (ANN)	46.36 (ANN)	57.8 (ANN)	17.83 (SVM)	38.95 (SVM)	50.00 (SVM)	19.63	19.02	15.60
Mozilla Core	25.73 (ANN)	52.17 (ANN)	63.84 (ANN)	22.25 (SVM)	47.87 (SVM)	59.37 (SVM)	15.64	08.98	07.53
Mozilla Firefox	90.13 (ANN)	98.74 (ANN)	99.24 (ANN)	89.49 (SVM)	98.48 (SVM)	99.77 (SVM)	00.71	00.26	00.00
Mozilla Firefox++	69.21 (ANN)	92.76 (ANN)	94.72 (ANN)	69.34 (SVM)	91.52 (SVM)	94.41 (SVM)	00.00	01.35	00.33

the similarities between developer D_p and $D_{p'}$.

$$cos\_sim(D_p, D_{p'}) = \frac{D_p^v \cdot D_{p'}^v}{\|D_p^v\| \|D_{p'}^v\|} \tag{11}$$

Following the above steps, we measure similarities among different developer profiles of the JDT dataset which is demonstrated in Fig. 6b. From the same cell of Fig. 6a and Fig. 6a, we can find that highly similar developer profiles tend to cause a higher number of coinciding incorrect bug assignments by our model. For instance, the similarities between top most coinciding developers ($\{D_8, D_{16}\} = 46$, $\{D_8, D_{11}\} = 31$, $\{D_3, D_{11}\} = 30$ and $\{D_8, D_{14}\} = 27$) are 79%, 75%, 76%, and 76%. To verify whether this pattern exists for all developers, we measure the correlation between their profile's similarity (Fig. 6b) and the total number of coinciding incorrect bug assignments (Fig. 6a). From Fig. 7 that depicts the correlation between Fig. 6a and Fig. 6b, we can observe that the similarity and the coinciding incorrect assignments exhibit a moderate positive correlation (0.56). This correlation is also statistically significant since the obtained P-value (2.1×10^{-13}) is very small even if we consider the lowest significance level, i.e., $\alpha = 0.001$. We also observe similar patterns while debugging the errors in other datasets. In the future, we aim to deal with the similarity issues among developers for making our model more accurate.



(a) Coinciding incorrect bug report assignment (b) Similarity (%) between developers' bug reports

FIGURE 6. Overview of deep error debugging over the JDT dataset. (a) Coinciding incorrect bug report assignments among developers. Here, each cell represents the total number of coinciding times between two developers; (b) Similarity between developers' in terms of their historical bug reports. Here, each cell represents the similarity between two developers.

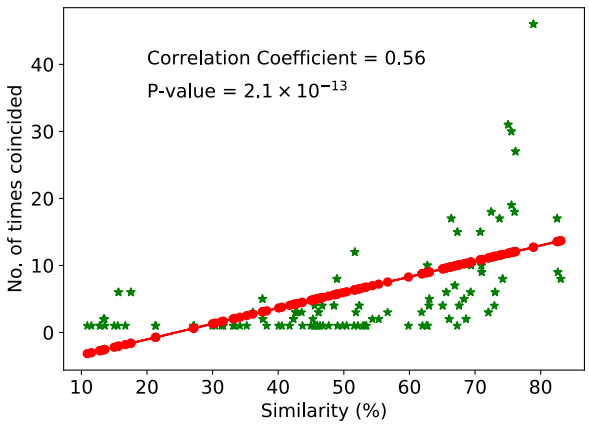


FIGURE 7. Correlation between bug report similarities and the coinciding incorrect assignments between two developers - the correlation coefficient score 0.56 indicates moderate positive correlation.

C. THREATS TO VALIDITY

1) EXTERNAL VALIDITY

We make our claims based on the findings from running experiments on the dataset of eight real-world open-sourced projects, which may not represent the characteristics of industrial, proprietary, and closed-sourced projects. In such cases, our approach may render different results. It is noteworthy that bug reports for proprietary industrial projects are not publicly available. Therefore, it is challenging to conduct experiments for any verification or validation purposes. However, since we take our benchmark dataset from a variety of open-source software, the severity of this external threat is relatively low.

Since our approach depends on the availability of bug assignment history, it cannot be readily applied to a new project or when a new developer joins a team. One possible remedy for the latter problem can be to match the skill sets of a new developer with an existing developer with a bug-fixing history [55].

2) INTERNAL VALIDITY

Our data preprocessing is somewhat different than other existing deep learning-based techniques. As a result, comparing results from our and existing technique(s) can be questioned. However, our data preprocessing is an integral part of the proposed approach and indicates better performance. Another internal threat is that if keywords do not repeat considerably among the bug reports of a particular developer, our ANN module may not perform well. To reduce this threat, our fusion-based models, such as ANN + CNN2 or ANN + CNN2 + CNN3 extract two types of features: repeating keywords and contextual relationships. If repeating keywords are missing, the model will rely on the contextual relationship or vice versa.

VI. CONCLUSION

Assigning reported bugs to appropriate developers is a critical and often expensive aspect of software development. Automated bug assignments with minimum or no user intervention could play a vital role in the bug resolution process. In this study, we propose a novel information fusion-based deep learning technique for automatically assigning the reported bugs. In our approach, we extract two types of features: (1) contextual relationship among consecutive words and (2) repeating keywords from the bug reports. We conduct an extensive ablation study of our proposed benchmark models on eight widely used, open-source datasets and present the results that show that our fusion-based approach performs better than other existing deep-learning-based techniques. The paper also contributes by debugging and reporting insights on bug assignment errors and publishes all the code used for running the experiments.

REFERENCES

[1] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?" in *Proc. 28th Int. Conf. Softw. Eng.*, May 2006, pp. 361–370.

[2] G. Murphy and D. Cubranic, "Automatic bug triage using text categorization," in *Proc. 16th Int. Conf. Softw. Eng. Knowl. Eng.*, 2004, pp. 1–6.

[3] K. Crowston, J. Howison, and H. Annabi, "Information systems success in free and open source software development: Theory and measures," *Softw. Process, Improvement Pract.*, vol. 11, no. 2, pp. 123–148, 2006.

[4] G. Jeong, S. Kim, and T. Zimmermann, "Improving bug triage with bug tossing graphs," in *Proc. 7th Joint Meeting Eur. Softw. Eng. Conf. ACM SIGSOFT Symp. Found. Softw. Eng.*, Aug. 2009, pp. 111–120.

[5] J. Anvik and G. C. Murphy, "Reducing the effort of bug report triage: Recommenders for development-oriented decisions," *ACM Trans. Softw. Eng. Methodol.*, vol. 20, no. 3, pp. 1–35, Aug. 2011.

[6] H. Mohsin, C. Shi, S. Hao, and H. Jiang, "SPAN: A self-paced association augmentation and node embedding-based model for software bug classification and assignment," *Knowl.-Based Syst.*, vol. 236, Jan. 2022, Art. no. 107711.

[7] H. Jahanshahi and M. Cevik, "S-DABT: Schedule and dependency-aware bug triage in open-source bug tracking systems," *Inf. Softw. Technol.*, vol. 151, Nov. 2022, Art. no. 107025.

[8] ATLASSIAN. *JIRA Software*. Accessed: Jul. 2, 2021. [Online]. Available: https://www.atlassian.com/software/jira

[9] *Bugzilla*. Accessed: Jul. 2, 2021. [Online]. Available: https://www.bugzilla.org/download/

[10] G. Yang, T. Zhang, and B. Lee, "Towards semi-automatic bug triage and severity prediction based on topic model and multi-feature of bug reports," in *Proc. IEEE 38th Annu. Comput. Softw. Appl. Conf.*, Jul. 2014, pp. 97–106.

[11] J. Xuan, H. Jiang, Z. Ren, J. Yan, and Z. Luo, "Automatic bug triage using semi-supervised text classification," in *Proc. 22nd Int. Conf. Softw. Eng. Knowl. Eng.*, 2010, pp. 209–214.

[12] S.-R. Lee, M.-J. Heo, C.-G. Lee, M. Kim, and G. Jeong, "Applying deep learning based automatic bug triager to industrial projects," in *Proc. 11th Joint Meeting Found. Softw. Eng.*, Aug. 2017, pp. 926–931.

[13] S. Mani, A. Sankaran, and R. Aralikatte, "DeepTriage: Exploring the effectiveness of deep learning for bug triaging," in *Proc. ACM India Joint Int. Conf. Data Sci. Manage. Data*, Jan. 2019, pp. 171–179.

[14] S. Guo, X. Zhang, X. Yang, R. Chen, C. Guo, H. Li, and T. Li, "Developer activity motivated bug triaging: Via convolutional neural network," *Neural Process. Lett.*, vol. 51, no. 3, pp. 2589–2606, Jun. 2020.

[15] S. F. A. Zaidi, F. M. Awan, M. Lee, H. Woo, and C. Lee, "Applying convolutional neural networks with different word representation techniques to recommend bug fixers," *IEEE Access*, vol. 8, pp. 213729–213747, 2020.

[16] A. K. Dipongkor. *Bug Triage Using Information Fusion*. [Online]. Available: https://github.com/dipongkor/bug-triage/

[17] M. Fischer, M. Pinzger, and H. Gall, "Analyzing and relating bug report data for feature tracking," in *Proc. 10th Work. Conf. Reverse Eng.*, Nov. 2003, pp. 90–99.

[18] I. T. Bowman and R. C. Holt, "Reconstructing ownership architectures to help understand software systems," in *Proc. 7th Int. Workshop Program Comprehension*, 1999, pp. 28–37.

[19] T. Zimmermann, A. Zeller, P. Weissgerber, and S. Diehl, "Mining version histories to guide software changes," *IEEE Trans. Softw. Eng.*, vol. 31, no. 6, pp. 429–445, Jun. 2005.

[20] A. Podgurski, D. Leon, P. Francis, W. Masri, M. Minch, J. Sun, and B. Wang, "Automated support for classifying software failure reports," in *Proc. 25th Int. Conf. Softw. Eng.*, 2003, pp. 465–475.

[21] D. Matter, A. Kuhn, and O. Nierstrasz, "Assigning bug reports using a vocabulary-based expertise model of developers," in *Proc. 6th IEEE Int. Work. Conf. Mining Softw. Repositories*, May 2009, pp. 131–140.

[22] R. Shokripour, J. Anvik, Z. M. Kasirun, and S. Zamani, "Why so complicated? Simple term filtering and weighting for location-based bug report assignment recommendation," in *Proc. 10th Work. Conf. Mining Softw. Repositories (MSR)*, May 2013, pp. 2–11.

[23] R. Almhana and M. Kessentini, "Considering dependencies between bug reports to improve bugs triage," *Automated Softw. Eng.*, vol. 28, no. 1, pp. 1–26, May 2021.

[24] P. Bhattacharya, I. Neamtiu, and C. R. Shelton, "Automated, highly-accurate, bug assignment using machine learning and tossing graphs," *J. Syst. Softw.*, vol. 85, no. 10, pp. 2275–2292, Oct. 2012.

[25] A.-C. Florea, J. Anvik, and R. Andonie, "Spark-based cluster implementation of a bug report assignment recommender system," in *Proc. Int. Conf. Artif. Intell. soft Comput.*, 2017, pp. 31–42.

[26] B. Soleimani Neysiani, S. M. Babamir, and M. Aritsugi, "Efficient feature extraction model for validation performance improvement of duplicate bug report detection in software bug triage systems," *Inf. Softw. Technol.*, vol. 126, Oct. 2020, Art. no. 106344.

[27] S. N. Ahsan, J. Ferzund, and F. Wotawa, "Automatic software bug triage system (BTS) based on latent semantic indexing and support vector machine," in *Proc. 4th Int. Conf. Softw. Eng. Adv.*, Sep. 2009, pp. 216–221.

[28] S. Nasim, S. Razzaq, and J. Ferzund, "Automated change request triage using alpha frequency matrix," in *Frontiers Inf. Technol.*, vol. 10, pp. 298–302, Jan. 2011.

[29] W. Wu, W. Zhang, Y. Yang, and Q. Wang, "DREX: Developer recommendation with K-nearest-neighbor search and expertise ranking," in *Proc. 18th Asia–Pacific Softw. Eng. Conf.*, Dec. 2011, pp. 389–396.

[30] X. Xia, D. Lo, X. Wang, and B. Zhou, "Accurate developer recommendation for bug resolution," in *Proc. 20th Work. Conf. Reverse Eng. (WCRE)*, Oct. 2013, pp. 72–81.

[31] L. Jonsson, M. Borg, D. Broman, K. Sandahl, S. Eldh, and P. Runeson, "Automated bug assignment: Ensemble-based machine learning in large scale industrial contexts," *Empirical Softw. Eng.*, vol. 21, no. 4, pp. 1533–1578, Aug. 2016.

[32] X. Ge, S. Zheng, J. Wang, and H. Li, "High-dimensional hybrid data reduction for effective bug triage," *Math. Problems Eng.*, vol. 2020, pp. 1–20, May 2020.

[33] W. Zhang, "Efficient bug triage for industrial environments," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol. (ICSME)*, Sep. 2020, pp. 727–735.

[34] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," in *Proc. 1st Int. Conf. Learn. Represent.*, 2013, pp. 1–14.

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

[36] M. Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations," in *Proc. Conf. North Amer. Chapter Assoc. Comput. Linguistics, Hum. Lang. Technol.*, 2018, pp. 2227–2237.

[37] 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. Computat. Social Syst.*, vol. 7, no. 5, pp. 1288–1303, Oct. 2020.

[38] R. Sepahvand, R. Akbari, B. Jamasb, S. Hashemi, and O. Boushehrian, "Using word embedding and convolution neural network for bug triaging by considering design flaws," *Sci. Comput. Program.*, vol. 228, Jun. 2023, Art. no. 102945.

[39] A. Kukkar, U. K. Lilhore, J. Frnda, J. K. Sandhu, R. P. Das, N. Goyal, A. Kumar, K. Muduli, and F. Rezac, "ProRE: An ACO-based programmer recommendation model to precisely manage software bugs," *J. King Saud Univ.-Comput. Inf. Sci.*, vol. 35, no. 1, pp. 483–498, Jan. 2023.

[40] J. Dai, Q. Li, H. Xue, Z. Luo, Y. Wang, and S. Zhan, "Graph collaborative filtering-based bug triaging," *J. Syst. Softw.*, vol. 200, Jun. 2023, Art. no. 111667.

[41] V. Dedik and B. Rossi, "Automated bug triaging in an industrial context," in *Proc. 42th Euromicro Conf. Softw. Eng. Adv. Appl. (SEAA)*, Aug. 2016, pp. 363–367.

[42] Y. Yin, X. Dong, and T. Xu, "Rapid and efficient bug assignment using ELM for IoT software," *IEEE Access*, vol. 6, pp. 52713–52724, 2018.

[43] J. Xuan, H. Jiang, Y. Hu, Z. Ren, W. Zou, Z. Luo, and X. Wu, "Towards effective bug triage with software data reduction techniques," *IEEE Trans. Knowl. Data Eng.*, vol. 27, no. 1, pp. 264–280, Jan. 2015.

[44] S. F. A. Zaidi. *Bug Triage Datasets: Sun Firefox, JDT, Netbeans, GUO and GCC*. Accessed: Feb. 7, 2021. [Online]. Available: https://github.com/farhan-93/bugtriage

[45] S. Mani. *Bug Triage Datasets: Google Choromium, Mozilla Firefox and Mozilla Core*. Accessed: Feb. 7, 2021. [Online]. Available: http://bugtriage.mybluemix.net

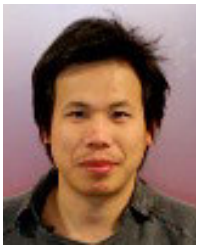
[46] R. A. Meyers, "Encyclopedia of physical science and technology," in *Data Mining and Knowledge Discovery*, 3rd ed. New York, NY, USA: Academic, 2003, pp. 229–246.

[47] A. K. Dipongkor, Md. S. Islam, H. Kayesh, M. S. Hossain, A. Anwar, K. A. Rahman, and I. Razzak, "DAAB: Deep authorship attribution in Bengali," in *Proc. Int. Joint Conf. Neural Netw. (IJCNN)*, Jul. 2021, pp. 1–9.
[48] M. Verleysen and D. François, "The curse of dimensionality in data mining and time series prediction," in *Proc. Int. Work-Conf. Artif. Neural Netw.*, 2005, pp. 758–770.
[49] K. W. Church, "Word2Vec," *Nat. Lang. Eng.*, vol. 23, no. 1, pp. 155–162, 2017.
[50] R. Hecht-Nielsen, "Theory of the backpropagation neural network," in *Neural Networks for Perception*. Washington, DC, USA: IEEE, 1992, pp. 65–93.
[51] R. Meyes, M. Lu, C. W. de Puiseau, and T. Meisen, "Ablation studies in artificial neural networks," 2019, *arXiv:1901.08644*.
[52] Z. Zhang and M. R. Sabuncu, "Generalized cross entropy loss for training deep neural networks with noisy labels," 2018, *arXiv:1805.07836*.
[53] T.-Y. Lin, P. Goyal, R. Girshick, K. He, and P. Dollar, "Focal loss for dense object detection," in *Proc. IEEE Int. Conf. Comput. Vis. (ICCV)*, Oct. 2017, pp. 2980–2988.
[54] A. Fred Agarap, "Deep learning using rectified linear units (ReLU)," 2018, *arXiv:1803.08375*.
[55] A. Sajedi Badashian, A. Hindle, and E. Stroulia, "Crowdsourced bug triaging," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol. (ICSME)*, Sep. 2015, pp. 506–510.

ATISH KUMAR DIPONGKOR received the B.Sc. and M.Sc. degrees from the University of Dhaka, Bangladesh. He is currently pursuing the Ph.D. degree in software engineering with George Mason University, USA. His research interests include the intersection between software engineering and machine learning, and natural language processing.

MD. SAIFUL ISLAM (Senior Member, IEEE) received the B.Sc. (Hons.) and M.S. degrees in computer science and engineering from the University of Dhaka, Bangladesh, in 2005 and 2007, respectively, and the Ph.D. degree in computer science and software engineering from the Swinburne University of Technology, Australia, in February 2014. He is currently a Senior Lecturer with the School of Information and Physical Sciences, The University of Newcastle, Australia. His current research interests include artificial intelligence, big data, and security analytics.

ISHTIAQUE HUSSAIN received the B.S. degree in computer science from the University of Dhaka, Bangladesh, and the Ph.D. degree from The University of Texas at Arlington. He is a former Assistant Professor of computer science with the Pennsylvania State University, Abington. He is currently a Senior Software Engineer with a company in New York.



SIRA YONGCHAREON (Senior Member, IEEE) received the Ph.D. and M.IT. degrees from the Swinburne University of Technology, Australia, in 2009 and 2012, respectively. He is currently an Associate Professor with the Computer Science and Software Engineering Department, Auckland University of Technology, New Zealand. He has published in several reputable journals and international conferences, including *ACM CSUR*, IEEE Internet of Things Journal, IEEE Transactions on Services Computing, *FGCS*, *ESWA*, *Knowledge-Based Systems*, IEEE Sensors, *Information Systems* (Elsevier), *WWWJ*, *ACM TMIS*, *Computers in Industry*, IEEE TrustCom, IEEE SCC, *BPM*, CoopIS, and *WISE*. His research interests include ubiquitous/pervasive computing, including AI/machine learning for the Internet of Things, ambient intelligence, human activity recognition, wireless sensing, and mobile/edge computing in intelligent environments. He is a Senior Member of ACM.

SAJIB MISTRY (Member, IEEE) received the B.S. and M.S. degrees in computer science from the University of Dhaka, Bangladesh, and the Ph.D. degree in computer science from the School of Science, RMIT University, Melbourne, Australia. He was a Postdoctoral Research Fellow with the School of Computer Science, The University of Sydney, Australia. He is currently a Senior Lecturer with the School of EECMS, Curtin University, Australia. He has teaching/research experience with The University of Sydney, RMIT University, Monash University, University of Dhaka, and the University of Liberal Arts. He has authored and edited several books and published in several top journals and conferences, such as *ACM TOIT*, *CACM*, IEEE Transactions on Services Computing, IEEE Transactions on Knowledge and Data Engineering, ICSOC, ICWS, and *SCC*. His primary research interests include service computing, cloud/edge computing, machine learning, augmented reality, and the Internet of Things (IoT). He is a member of the Prestigious Sydney IoT Hub. He was a PC Member in ICWS 2019–2020, ICSOC 2019, and WISE 2018–2019. He is a regular reviewer of top journals and conferences in the TCSVC field. He won the Best Research Paper Award from ICSOC 2016 and the RMIT Publication Award 2016. One of his journal articles was selected as the spotlight paper for IEEE Transactions on Services Computing. He has contributed significantly with his community services as the PC Chair of ASSRI 2018.