IEEE Access
Multidisciplinary ⋮ Rapid Review ⋮ Open Access Journal

Received December 30, 2021, accepted January 25, 2022, date of publication February 21, 2022, date of current version March 1, 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3153075

A Graph Convolution Network-Based Bug Triage System to Learn Heterogeneous Graph Representation of Bug Reports

SYED FARHAN ALAM ZAIDI[1,2], HONGUK WOO[3], AND CHAN-GUN LEE[2]

[1]CAU Institute of Innovative Talent of Big Data, Department of Computer Science and Engineering, Chung-Ang University, Seoul 06974, South Korea
[2]Department of Computer Science and Engineering, Chung-Ang University, Seoul 06974, South Korea
[3]Department of Software, Sungkyunkwan University, Suwon 16419, South Korea

Corresponding author: Chan-Gun Lee (cglee@cau.ac.kr)

This work was supported by the National Research Foundation of Korea (NRF) under Grant NRF-2021R1F1A1059492.

ABSTRACT Many bugs and defects occur during software testing and maintenance. These bugs should be resolved as soon as possible, to improve software quality. However, bug triage aims to solve these bugs by assigning the reported bugs to an appropriate developer or list of developers. It is an arduous task for a human triager to assign an appropriate developer to a bug report, when there are several developers with different skills, and several automated and semi-automated triage systems have been proposed in the last decade. Some recent techniques have suggested possibilities for the development of an effective triage system. However, these techniques require improvement. In previous work, we proposed a heterogeneous graph representation for bug triage, using word–word edges and word-bug document co-occurrences to build a heterogeneous graph of bug data. Cosine similarity is used to weight the word–word edges. Then, a graph convolution network is used to learn a heterogeneous graph representation. This paper extends our previous work by adopting different similarity metrics and correlation metrics for weighting word–word edges. The method was validated using different small and large datasets obtained from large-scale open-source projects. The top-k accuracy metric was used to evaluate the performance of the bug triage system. The experimental results showed that the point-wise mutual information of the proposed model was better than that of other word–word weighting methods, and our method had better accuracy for large datasets than other recent state-of-the-art methods. The proposed method with point-wise mutual information showed 3% to 6% higher top-1 accuracy than state-of-the-art methods for large datasets.

INDEX TERMS Bug triage, bug report, software maintenance, defect triage, bug assignment, bug report, bug fixer recommendation.

I. INTRODUCTION

Bug triage is a difficult task in software maintenance. It requires the allocation of a suitable developer to a bug report. Bugs are faults, mistakes, or gaps in software that should be addressed with a specific priority, to improve software quality. Testing engineers or quality assurance engineers detect flaws and gaps during testing and maintenance of the software. Developers and engineers use open-bug repositories (JIRA or Bugzilla) for assistance in fixing issues. Mozilla, Eclipse, and Net-Beans are examples of well-known large-scale open-source projects that use open bug repositories to submit issues. A developer is assigned to a bug reports—a document that is used to report a problem—by a manual triager or a triage manager, a process which is time consuming. It is stressful for a triage manager to evoke the developer's expertise and allocate a bug to the most suitable developer.

Many automated bug triage approaches have been developed to overcome the manual triage problem in the last decade. However, these approaches are still producing unsatisfactory outcomes. Many researchers have used mining repositories, social network analysis, topic modeling, statistical approaches, and classic machine learning methods to solve bug triage problems. However, these techniques have yielded good results only for small datasets.

Recently, many deep learning methods, along with Natural Language Processing (NLP) methods, have shown promising results for bug triage. Word representation and word embedding are NLP techniques, used for converting text into

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

vectors. Then, these vectors are fed to deep neural networks, convolution neural networks (CNNs), or recurrent neural network (RNNs). Lee *et al.* [1] were the first to use CNN-based dense neural networks along with Word2Vec word embedding. Guo *et al.* [2] proposed a word2vec and CNN-based architecture for bug triage. However, their technique assigned the developer to a task based on their activities.

Mani *et al.* [3] proposed an attention-based bidirectional recurrent neural network that also used word2vec embedding. Zaidi *et al.* [4] used different context-aware and context-insensitive techniques for word-representation with a CNN model. They produced promising results compared to previous methods. However, the accuracy of these models is unsatisfactory and requires improvement.

Graph neural networks (GNNs) and graph embeddings are new research directions that have been used for various classification and categorization tasks. A GNN effectively learns a deep relational structure, and can maintain a graph's global structure information in graph embedding. Kipf *et al.* [5] proposed a graph convolution network (GCN), which understands the neighborhood information. Their model learns hidden layer representations, which encode local structures of graphs and the features of nodes.

Recently, Wu *et al.* proposed a spatial-temporal dynamic graph neural network (ST-DGNN)-based automated bug triage method that considered the activity of developers when assigning bug reports. They considered the bug report summary, developers' activity, and their comments to triage the bug. They used joint random walk (JRWalk) for topological sampling and a graph recurrent convolutional neural network (GRCNN) to learn the spatial-temporal features of dynamic developer collaboration networks (DCN). [6].

Previous work [7] reported a heterogeneous graph representation of bug reports that builds heterogeneous graphs from summaries and description of bug reports. The heterogeneous graph has word-to-word co-occurrences and word-to-bug document co-occurrences. The technique uses term frequency-inverse document frequency (TF-IDF) for weighting words to bug document edges, and cosine similarity calculated for weighting word–word edges. Then, a simple two-layer GCN was trained on a heterogeneous graph.

In this study, we extended our previous work using different similarity and co-occurrence measures for weighting word–word edges. We adopted Jaccard similarity, Euclidean similarity, Pearson correlation, dice similarity, Hellinger similarity, and point-wise mutual information instead of the cosine similarity used in our previous work. The proposed method does not consider the developers' comments or activity, and so the proposed method does not rely on social graphs. It considers the summary and description, and builds a heterogeneous graph representation of the bug reports. ST-DGNN uses JRWalk, which aims to embed nodes or vertices in a homogeneous graph. In contrast, we use a heterogeneous graph with graph convolution network for bug triage.

Specifically, we raise the following research questions in the context of bug triage:

- Which method is effective for weighting the word–word edges?
- Is the graph embedding better than context-insensitive word-embeddings?
- Is the proposed triage technique faster than the word representation-based approaches?
- Is the graph representation memory efficient for bug reports?

The main contributions of the research are as follows:

- To the best of our knowledge, the proposed bug triage method is the first that solves bug triage problems using a heterogeneous graph with a graph convolution network. The previous graph-based methods used social network analysis techniques and dynamic graph techniques, and did not use heterogeneous graphs. Moreover, previous methods created relational graphs based on developers' activities using summaries, descriptions, and comments. The proposed method only uses summaries and descriptions to build a heterogeneous graph.
- The performance of the proposed method was validated on several large datasets from open-source projects.
- The proposed method was compared with some recent deep learning-based triage methods such as Deep triage [3], DA-CNN [2], Glove-CNN [4], ELMo-CNN [4], and ST-DGNN [6], which have used the publicly available datasets or have published their datasets.

II. RELATED WORK

Many recent studies have addressed the issue of bug triage. Researchers initially used non-machine learning methods for bug triage. They used entropy-based, ranking-based, and statistical methods. Between 2013 and 2017, many machine learning-based methods were proposed for bug triage. These methods used conventional machine learning methods. A deep learning-based method for bug triage was proposed for the first time in 2017.

Non-machine learning strategies included mining software repositories (MSR), social network analysis, and activity models. Historical information on system development and maintenance can be found in software repositories. Researchers used information retrieval techniques to extract the important information as features. Researchers have mined this historical information, including source code and version control repositories, to identify suitable developers to address bug reports.

Kagdi *et al.* [8] used source files to create a dataset, and used latent semantic indexing (LSI) to retrieve the information. They computed the similarity of the bug reports, to predict the relative source file using the indexed corpus. Then, their algorithm assigns developers based on their activity for the related source file. Shokripour *et al.* [9], [10] proposed two different methods. Firstly, they applied the phrase composition technique on commit and description to extract the information from repositories. This method suggests a developer based on their activity with the file and most similar phrase composition score. Secondly, they extracted nouns

from the commit message, source code, and description that determined the bug's location. Then, the term-weighting scheme was used to identify the files belonging to the new bug report, and assign developers based on their expertise with the predicted files.

Banitaan *et al.* [11], Zhang *et al.* [12], and Hu *et al.* [13] proposed social network analysis-based approaches for triaging bugs. They built social networks of developers' collaborations using the comments on bugs, and then calculated the fixing probability and associations scores for assigning the developers.

A few approaches have used the developers' knowledge to triage problems by modeling their commenting, reporting, and fixing activities. Some researchers assigned suitable developers based on association scores and correlation scores determined by the developer activity and related topics, using a topic modeling technique [14], [15]. Zhang *et al.* [16] enhanced work by combining the topic model with developer and reporter relations, based on their history.

Wang *et al.* [17] proposed an unsupervised method that groups the developers based on their component-level activities. The approach calculates the score of activity for specific periods in the group, and then assigns an appropriate developer to a bug report using the activity score. Xia *et al.* [18] and Zhang *et al.* [19] enhanced the latent Dirichlet allocation (LDA) model using a multi-feature approach and entropy-based optimization, respectively. Then, the method assigned developers based on their affinity scores.

Recently, Yadav *et al.* [20] proposed a technique that ranks developers according to their expertise in triaging bugs. They decreased the bug tossing length. They constructed developers' profiles dependent on their commitment and collaboration. The developer expertise scores are produced by utilizing fixing time, priority weighted fixed issues, and indexed metrics. Then the component-based, cosine, and Jaccard similarity are determined to calculate the expertise score. Based on the expertise score, the method recommends a ranked list of suitable developers. Kumari *et al.* [21] tackled the bug triage problem with a bug dependency-based mathematical model. The bug dependency exists due to coding mistakes, deficiencies in design, and misconceptions among users and developers. The entropy was calculated from the summary, description, and comments from the bug reports. The developer's assignment was dependent on the entropy.

Many studies were proposed for bug triage using well-known machine learning algorithms between 2010 and 2017. These methods used term frequency-inverse document frequency (TF-IDF) for feature extraction from bug reports, such as summary description, comments, and source code. The studies [22]–[26], and [27] used TF-IDF for feature extraction and vectorization. These studies used machine learning algorithm such as support vector machines (SVMs), naïve Bayes, decision trees, k-nearest neighbors (KNN), and logistic regression. Logistic regression showed better performance than the other machine learning algorithms for the bug triage problem.

Alenezai *et al.* [28] built a model using a naïve Bayes classifier to assign a fixer to a newly reported bug. They used five term-selection methods: chi-square, log odds ratio, term frequency relevance frequency, mutual information, and distinguishing feature selector, to choose the discriminatory terms to engineer features for learning the prediction model. Alenezai *et al.* [29] considered categorical features and metadata from bug reports with textual attributes for feature extraction. They used the gain ratio to find the essential features that provided the normalized measure of each feature which contributed to the classification. The use of categorical data with text data produced slightly better results than only using test data. Only the use of categorical features produced a deficiency in the triage performance.

Zhao *et al.* [30] combined a topic model and a vector space model for triage data. The TF-IDF vectorizer was used to build a vector space model, and LDA was used to create a topic model. Two different machine learning algorithms, an SVM and a neural network, were used for classification tasks, and the SVM was found to perform better than the neural network.

Since 2017, deep learning techniques using NLP word-embedding or word representation techniques have been proposed to advance bug triage research. The word embedding techniques convert the text data into vectors which are fed into a deep learning model. The word2vec embedding is the most used technique for text classification tasks. Lee *et al.* [1] were the first to propose a deep learning-based solution for bug triage. They used the word2vec embedding model for vectorizing text, and a CNN model to predict the appropriate fixer. They calculated the top-1 to top-5 accuracy to evaluate the performance of 5 suitable developers against one bug report.

Mani *et al.* [3] proposed a bi-directional recurrent neural network-based technique for automatic bug triage. They used word2vec embedding for text vectorization and an attention mechanism that learns the syntactic and semantic features of a long word sequence. The approach was shown to be superior to traditional machine learning methods. Guo *et al.* [2] proposed a CNN-based method and also considered developer activities. They used time-split validation to make a real scenario, used word2vec embedding for vectorization, and validated their method with large datasets from open-source projects.

Zaidi *et al.* [4] also proposed a CNN-based bug triage system that recommends a list of ten developers. Three different word-embedding techniques (word2vec, GloVe, and ELMo) were used for vectorization. The ELMo-based CNN model performed better than the others. Mian *et al.* [31] proposed a bi-LSTM-DA based triage method with GloVe word embedding for efficient word representation. The method was compared with [2] and showed comparable results.

Recently, Aung *et al.* [32] proposed a multi-triage model that assigns developers and issue types simultaneously. They used two different deep learning models for feature extraction. The text encoder module was based on a CNN model,

and an abstract syntax tree encoder module was based on biLSTM. The researchers then concatenated the features of both encoders and trained the two different classifiers for the developer assignment and bug issue type tasks. Their model produced good accuracy and performed both tasks simultaneously, but required more training time than other methods, due to the need to train two different encoders and models.

Very few studies have been performed for bug triage using graph-based neural networks. Wu *et al.* [6] proposed a spatial-temporal graph-based dynamic graph neural network in which they used joint random walk (JRWalk) and a graph recurrent convolutional neural network (GRCNN). Thr JR walk was used for topological sampling, and the GRCNN was used to learn the spatio-temporal features of dynamic developer collaboration networks. The researchers used descriptions and comments to build the developer collaboration networks. Alazzam *et al.* [33] proposed a feature augmentation approach based on relationships in a social graph. They used frequency, correlation, and neighborhood overlap techniques to build an augmentation approach. They used the term ''bug triage'' in the context of correctly assign priority to new bugs.

Most recently, researchers have proposed triage methods based on dependencies. Almhana *et al.* [34] proposed an automated bug triage method that considers dependencies between bug reports, and then localizes the files to be inspected for each open bug report. Multi-objective search is used to rank the bug reports for programmers, based on dependencies for other reports and priorities. Their approach produced a significant time reduction, of over 30%, in localizing bugs, compared to traditional bug prioritizing techniques.

Jahanshahi *et al.* [35] proposed a dependency-aware bug triage method unlike previous dependency-based methods. They used NLP and integer programming to assign bugs appropriately. Their method incorporated textual information, dependency between bugs, and the cost associated with each bug. The technique reduced the number of overdue bugs, and improved the bug-fixing time. However, they limited their work by assuming that each developer can work on only a single report at a time, which is not a realistic scenario in practice.

Software defect prediction is a similar problem to bug triage. Khurma *et al.* [36] proposed an island binary moth-flame optimization (IsBMFO) base model that divides the solution in the population into subpopulations called islands. Then, each island is treated independently. They used IsBMFo for feature selection and three different classifiers, SVM, KNN, and naïve Bayes, for classification. The SVM with IsBMFO performed better than KNN and naïve Bayes.

III. MOTIVATION AND PREVIOUS WORK

A significant number of bugs are reported daily, and these bugs should be fixed as soon as [possible, to improve the quality of the software. It is very difficult to triage bugs using a manual triage manager. To overcome these issues, many triage systems have been proposed in the last decades, which are discussed in Section II. However, the existing methods have some limitations.

Early research into the bug triage problem involved mining repositories, social network analysis, and activity modeling/topic modeling. These methods showed good results at that time. However, these methods were not scalable, and were tested only on small datasets with limited fixer information. In reality, open-source projects have significant numbers of developers.

The field evolved when researchers used machine learning techniques and treated bug triage problem as a classification problem. Machine learning has been used in various software engineering problems. Researchers have used summaries, descriptions, and comments for feature extraction. Then, they used well-known classifiers to assign a fixer to each reported bug. These triage methods showed good performance for small datasets, and in situations in which the number of developer classes is limited. In reality, open-source projects have many developers. The performance of these methods decreased with increasing dataset size and developer classes. Nevertheless, these methods achieved higher accuracy than previous mining and social network analysis-based triage techniques.

Recently, most researchers have used NLP techniques for word representation and have applied deep learning models such as CNNs and RNNs for training and assigning appropriate fixers. Deep learning requires a large amount of data for efficient training. The researchers used large datasets from large-scale open projects to train their CNN and RNN models. These triage techniques produced higher accuracy than previous machine learning methods. However, the top-k accuracies are still very far from satisfactory. Deep learning methods require a significant time for training models using parallel-processing graphical processing unit (GPUs).

Since graph convolution networks have produced good performance in classification tasks, heterogeneous graphs have also attracted more attention recently. Yao *et al.* [37] proposed a GCN-based method for classification. The 20 Newsgroup (20NG), Reuters-8 (R8), Reuters-52 (R52), and Movie Review (MR) datasets were used. The R52 dataset has 52 classification classes. Their GCN technique showed good accuracy on benchmark datasets compared to CNN, LSTM, and Bi-LSTM.

We found the work of Yao *et al.* to be highly relevant to our research. Consequently, we proposed a heterogeneous graph-based bug triage method that used GCN for learning a graph to predict the allocation of appropriate developers to bug reports [7]. A heterogeneous graph was built using the summaries and descriptions of the bug reports. Each bug report and the unique words (vocabulary) from the summary and description were used as nodes. The TF-IDF score was used to compute the co-occurrences between bug document (i) and word (j). The cosine similarity (CS) was used to weight the edges between words. Equation 1 shows the mathematical notation for calculating the cosine

similarity between two words i and j. A heterogeneous graph was represented by an adjacency matrix A, as shown in Equation 2. A two-layer GCN was used to train the graph representation using a softmax classifier for the prediction task.

The proposed method is different from other proposed graph-based triage techniques. The studies [6] and [33] use social graphs for feature extraction and feature augmentation, respectively. Both studies used comments and summaries from the bug reports, and make relations between developer and bug reports according to their activities. In our work, we only used summaries and descriptions to obtain the unique words and terms to make a model in the graph. Then, different edges were created according to the correlation and similarity scores.

$$CS(i,j) = \frac{i.j}{|i||j|} \tag{1}$$

$$A_{ij} = \begin{cases} 1, & \text{if } i = j. \\ TF - IDF(i,j), & \text{if } i \text{ is a document,} \\ & \text{and } j \text{ is word.} \\ CS(i,j), & \text{if } CS(i,j) > 0.9 \\ & \text{and } i,j \text{ are words.} \\ 0, & \text{otherwise.} \end{cases} \tag{2}$$

IV. METHODOLOGY

A relatively large amount of data is included in bug reports. Each report was treated as a separate document. The bug report includes text and categorical attributes, as well as other details regarding the bug. The summary and description are the text attributes utilized from bug reports for training the model. The text attributes needed to be cleaned before feeding into the recommendation system. Our bug triage process has three main phases: preprocessing, a graph representation of bug reports, and a graph convolution network. Figure 1 shows the schematic diagram of the proposed bug triage method.

A. PREPROCESSING

We use the summary and description from the bug reports as input information. The information about the fixer is utilized as a label or a class attribute. The textual attributes, such as summary and description, are unstructured data in the triage system. Therefore, whitespace, stack traces, URLs, special characters, hexadecimal codes, punctuation marks, code snippets, and directory paths from the description are removed in the preprocessing step. Stopwords are removed from the summary and description using Stanford's NLTK library. Finally the vocabulary (unique words) is created from the clean data.

B. GRAPH REPRESENTATION OF BUG REPORTS

A graph G has vertices V and edges E. The cleaned data is used to create the graph. As mentioned in previous work, we generated a heterogeneous graph for training the triage system. The graph's vertices are based on the vocabulary size and the number of bug reports. The adjacency matrix describes the heterogeneous graph, which is used as a feature matrix. The adjacency matrix is estimated to identity matrix I initially, representing a self-loop of vertices. The heterogeneous graph has two types of edges: the word-to-bug document edge, which uses the TF-IDF score for weighting the edges, and the word-to-word co-occurrence, weighted by calculating a similarity measure between two words.

We adopted different methods for weighting word-to-word edges. We used Jaccard similarity (JS), Euclidean similarity, Pearson correlation, dice similarity, Hellinger similarity, and point-wise mutual information instead of cosine similarity. The $Similarity(i,j)$ in equation 4 shows the generalization that can be replaced by any of the above word–word weighting techniques with the relative threshold value.

$$A_{ij} = \begin{cases} 1, & \text{if } i = j. \\ TF - IDF, & \text{if } i \text{ is a document, and } j \text{ is word.} \\ Similarity(i,j), & \text{if } Similarity(i,j) > Th \\ & \text{and } i,j \text{ are words.} \\ 0, & \text{otherwise.} \end{cases} \tag{3}$$

1) JACCARD SIMILARITY (JS)

JS is a statistical way in which to determine the similarity and diversity between two finite sample sets. We used JS to find the similarity between two words (i and j) and make an edge between words if the similarity is >= 0.5. The JS is calculated by Equation 4, which was also used by [38] for keywords similarity.

$$JS(i,j) = \frac{|i \cap j|}{|i \cup j|} = \frac{|i \cap j|}{|i| + |j| - |i \cap j|} \tag{4}$$

2) EUCLIDEAN SIMILARITY (ES)

We calculate Euclidean similarity by subtracting the Euclidean distance (ED) from 1. The same threshold value as JS is used to make edges between the words. The ES is calculated using Equation 5.

$$\begin{aligned} ES(i,j) &= 1 - ED(i,j) \\ ES(i,j) &= 1 - \sqrt{|(i-j)|^2} \end{aligned} \tag{5}$$

3) PEARSON CORRELATION (PC)

We carried out experiments using correlation instead of similarity. We used Pearson correlation to determine the linear relationship between two words. An edge was only made if the words were highly correlated (where the correlation value was greater than 0.5 and close to 1). We used Scipy's "Pearsonr" function to calculate the Pearson correlations between words.

4) DICE SIMILARITY (DS)

Dice similarity (DS) was used to calculate the similarity between two words. Edges were established if the similarity

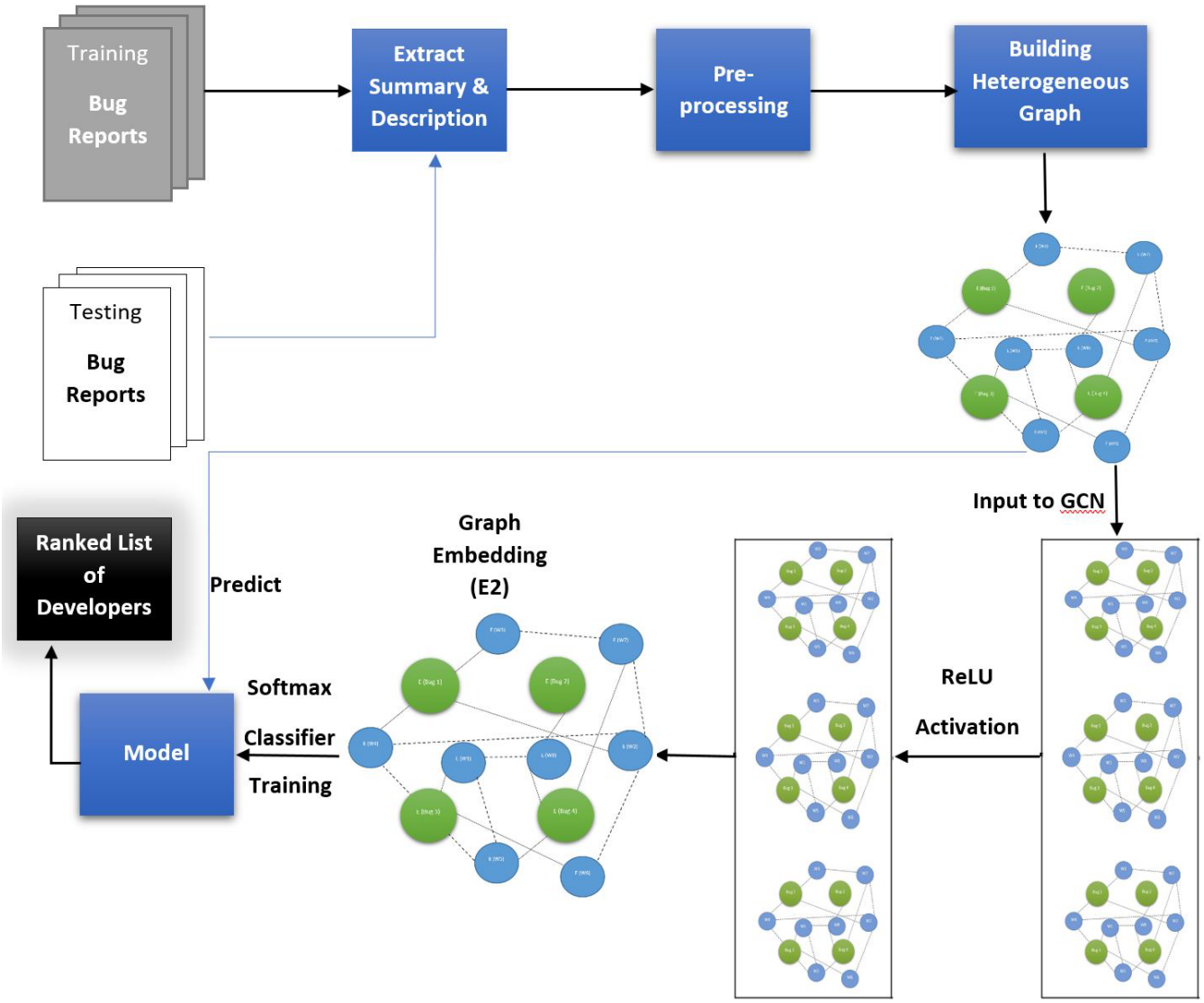

FIGURE 1. Proposed graph convolution based bug triage system.

threshold was >= 0.5. The DS was calculated according to Equation 6, which is a widely used metric.

$$DS(i,j) = \frac{2|i \cap j|}{|i| + |j|} \tag{6}$$

5) HELLINGER SIMILARITY (HS)

Hellinger Distance (HD) is the probabilistic analog of ED. Hellinger similarity is calculated by $1 - HD$. Hellinger similarity was calculated using Equation 7. Two words were linked if the Hellinger similarity was >= 0.5.

$$HS(i,j) = 1 - \frac{1}{\sqrt{2}}\sqrt{|(\sqrt{i} - \sqrt{j})|^2} \tag{7}$$

6) POINT-WISE MUTUAL INFORMATION (PMI)

PMI is a popular statistical measure for determining the association between two words. In a text corpus, co-occurrences and occurrences of words may be used to estimate the probabilities $p(i, j)$ and $p(i)$, respectively. Edges were established if the threshold was greater than zero. It is computed as follows:

$$PMI(i,j) = \log \frac{p(i,j)}{p(i)p(j)} \tag{8}$$

C. GRAPH CONVOLUTION NETWORK (GCN)

A GCN is a multi-layer neural network that directly acts on graphs and generates embedding vectors based on their neighborhood properties. We use a simple two-layer GCN for our work, as Yao *et al.* used for the text classification tasks. The heterogeneous graph was fed into a GCN, and a normalized symmetric adjacency matrix ($\hat{A}$) was used for better computation and results. $\hat{A}$ was computed as follows:

$$\hat{A} = D^{-1/2}AD^{-1/2} \tag{9}$$

where D is the degree of matrix A. So, The output of the first layer is the new feature matrix E_1 or the word embedding of, which is computed as follows:

$$E_1 = ReLU(\hat{A}XW_0) \tag{10}$$

where X is the input feature matrix, W_0 is the initial weights, and ReLU was used as the activation function. The second layer is fed to the softmax classifier. Therefore, we used the softmax activation function in layer two instead of ReLU. The number of nodes was the same as the number of labels in the second layer. So, the output (O) is calculated as follows:

$$E_2 = \hat{A}E_1W_1 \tag{11}$$

$$O = softmax(E_2) \tag{12}$$

E_2 is the embedding and new feature matrix for the second layer, and W_1 is the first layer's weight.

V. EVALUATION AND RESULTS

This section evaluates the proposed bug triage system and addresses the research questions mentioned in the Introduction section.

A. DATA COLLECTION

Data from the large-scale open-source projects was used to evaluate the performance of the proposed bug triage system. Lee [1], and Zaidi [4] used two datasets, Eclipse Platform and Mozilla Firefox, to evaluate the performance of their triage system. Mani [3] and Zaidi [4] used another Mozilla Firefox dataset, including variants with minimum 0, 5, 10, and 20 numbers of bug reports per developer. We used the same datasets to evaluate and validate our proposed bug triage system. Guo *et al.* [2] and Zaidi *et al.* [4] used a massive Mozilla dataset for their experimentation. We used this dataset to validate the performance of our proposed approach. The datasets are publicly available on GitHub.[1]

Wu *et al.* [6] built an Eclipse dataset, which contains 200K solved bug reports from 81 components between October 2001 and November 2011. The dataset has 3,893 developers who participated in the bug fixing process. The dataset is publicly available at GitHub.[2]

B. EVALUATION MEASURE

The top-k accuracy is used to evaluate the proposed bug triage system. We calculated the top-1 to top-10 accuracy and compared it with state-of-the-art triage methods. Equation 13 was used to calculate the top-k accuracy.

$$Top-k\ accuracy = \frac{\sum_{i=1}^{N} I(rec_i@k, dev_i)}{|N|} \tag{13}$$

Ten-fold cross-validation and time-split validation were used to evaluate the method. Ten-fold cross-validation was used to evaluate the proposed triage system's performance on the Eclipse's JDT, Eclipse's Platform, and Mozilla Firefox datasets. Time-split validation was used to evaluate the model with the thresholded Mozilla Firefox dataset.

[1]https://github.com/farhan-93/bugtriage

[2]https://github.com/ssea-lab/BugTriage/tree/master/ (The GitHub link is taken from their own paper.)

C. EXPERIMENTAL RESULTS

We used the datasets described above for the experiments. Table 1 shows the experimental results of the Platform [1], Firefox-small [1], Firefox-thresholded [3], and Firefox [2] datasets. The reported results are the average of five trials. The experimental results show the superiority of the proposed method with the PMI method for weighting the word–word edges for most cases. The platform was a small dataset with a small number of developers compared to other Firefox datasets. Our GCN-PMI performs better for the Platform dataset than previous work, and ELMo-CNN for the top-1 to top-4 accuracy. However, ELMo-CNN shows better results for thr top-5 to top-10 accuracies. However, our proposed method produced better performance than all other methods for the Firefox [1] dataset, which is a larger dataset than the Platform.

The Firefox thresholded [3] dataset is large. The ELMo-CNN produced better results for the top-1 to top-6 accuracies on the 0-threshold dataset. However, PMI-GCN performed well for the top-7 to top-10 accuracies. In contrast, our PMI method performed more effectively on a threshold of 10, indicating that when a fixer has a good triage history, it can build a good prediction model.

Similarly, Guo *et al.* [2] cleaned the datasets with 10-threshold—that is, they selected only those developers who had fixed at least 10 bugs. Our previous work and our-PMI methods show a noticeable improvement in performance from top-1 to top-10 accuracy compared to DA-CNN, Word-2Vec-CNN, GloVe-CNN, and ELMo-CNN.

D. ADDRESSING THE RESEARCH QUESTIONS

RQ 1: Which method is effective for weighting the word–word edges?

As mentioned in Section IV-B, different methods were used to weight the word–word edges: JS, Euclidean similarity, Pearson correlation, dice similarity, Hellinger similarity, and point-wise mutual information. The cosine similarity was used in previous work [7] for word–word edges weighting. The experimental results demonstrate the superiority of PMI compared to the other methods on all five datasets. The detailed experimental results are shown in Table 2.

To check the significance of the results, we performed Friedman's test. The Friedman's test has a p-value < 0.05, that confirms the significance of the results. A post-hoc Nemenyi test was performed to check the significant difference between the different word–word weighting based GCN methods with a 95% confidence interval. Overall, the PMI based GCN method showed higher accuracy. The Nemenyi test confirmed the significant difference between the PMI based GCN method and CS-GCN, JS-GCN, and ES-GCN. The PC-GCN, HS-GCN, and Dice-GCN had negligible differences.

Figure 2 is the Demšar diagram that shows the average rank of the proposed methods with different word–word weighting schemes. The horizontal line shows the average rank.

TABLE 1. Average top-1 to top-10 accuracy obtained on platform, Firefox-small [1], Firefox-thresholded [3] and Firefox [2] datasets with different word-word weighing scheme based GCN.

Dataset	Techniques	top-1 Acc. %	top-2 Acc. %	top-3 Acc. %	top-4 Acc. %	top-5 Acc. %	top-6 Acc. %	top-7 Acc. %	top-8 Acc. %	top-9 Acc. %	top-10 Acc. %
Platform [1]	CS	39.54	50.00	56.09	59.90	62.01	64.40	65.42	67.49	69.87	72.11
	JS	41.476	49.482	53.727	56.729	60.041	62.112	65.217	67.495	69.669	71.222
	ES	42.029	51.449	56.624	59.627	61.077	63.665	66.046	69.255	70.807	72.36
	PC	39.027	48.654	54.451	57.453	60.87	63.251	65.321	67.184	68.427	69.876
	HS	42.754	51.553	55.28	58.075	62.629	65.224	68.323	69.876	71.222	72.05
	DS	42.961	50.414	55.072	59.731	62.112	63.768	67.081	68.944	70.393	72.464
	PMI	44.410	52.588	57.35	60.559	63.458	64.7001	67.495	70.001	70.222	73.188
Firefox [1]	CS	29.37	41.81	47.84	53.89	57.03	59.52	61.91	63.82	68.16	66.5
	JS	32.177	44.158	51.526	56.077	58.373	61.627	64.402	66.699	69.038	70.378
	ES	29.378	40.67	47.177	52.249	55.502	58.756	61.244	62.775	64.402	66.22
	PC	32.345	45.933	51.962	55.885	60.383	63.254	65.742	67.751	68.889	70.239
	HS	30.048	44.115	51.196	56.651	61.053	63.158	64.21	66.029	68.134	69.569
	DS	30.335	41.914	50.813	56.364	60.67	64.115	65.55	67.273	68.995	69.761
	PMI	35.120	48.705	55.024	59.904	63.349	65.550	67.751	69.856	71.866	73.11
Firefox [3] 0-Threshold	CS	15.48	22.18	27.67	30.89	34.03	36.63	38.49	40.68	42.09	43.60
	JS	16.853	24.081	29.616	33.416	36.142	38.455	40.81	42.173	43.536	45.188
	ES	16.894	24.122	29.951	33.456	36.349	38.993	41.983	42.667	44.123	44.857
	PC	17.266	24.411	29.451	33.251	35.812	38.125	40.107	41.883	43.04	44.692
	HS	17.203	24.751	30.051	33.548	36.316	39.538	42.021	43.04	44.94	46.799
	DS	16.02	24.081	29.781	33.292	36.101	38.001	39.901	41.925	43.412	45.188
	PMI	19.849	28.414	34.041	37.544	40.784	43.51	**46.014**	**47.941**	**50.248**	**51.216**
Firefox [3] 10-Threshold	CS	25.661	34.55	36.458	40.004	44.078	48.415	52.845	56.474	60.78	63.14
	JS	21.982	28.735	34.765	38.056	41.733	44.917	47.317	52.178	56.716	59.746
	ES	26.806	33.451	36.82	40.327	43.281	47.311	51.296	54.154	58.879	62.27
	PC	29.984	40.177	46.288	50.006	52.735	56.735	59.507	62.048	64.041	65.055
	HS	26.811	33.456	36.825	40.332	43.286	47.316	51.266	54.024	58.749	62.14
	DS	26.914	34.578	36.914	40.128	42.845	46.214	52.266	55.014	58.965	61.746
	PMI	**32.343**	**44.639**	**51.390**	**56.318**	**61.374**	**63.197**	**65.559**	**68.304**	**69.998**	**71.008**
Firefox [2]	CS	18.126	27.035	32.719	36.866	40.015	42.704	45.161	46.237	47.619	48.201
	JS	17.626	26.535	32.219	36.366	39.515	42.204	44.661	45.737	47.119	47.701
	ES	19.906	28.501	33.965	38.016	42.885	44.295	46.475	48.295	49.797	53.055
	PC	19.811	28.401	33.890	37.990	42.770	44.120	46.390	47.970	49.540	53.010
	HS	20.001	28.601	34.040	38.041	43.000	44.470	46.560	48.620	50.053	53.100
	DS	20.110	29.010	34.010	38.140	43.010	44.610	46.890	48.990	50.001	52.987
	PMI	**21.352**	**30.031**	**35.714**	**39.555**	**43.395**	**45.852**	**47.542**	**50.693**	**52.845**	**54.379**

The connection between the word–word weighting methods shows an insignificant difference. The critical distance calculated for the data of five datasets was 3.209, with a 95% confidence interval. As in the Nemenyi test, the PMI-based GCN method was significantly different from the cosine, Jaccard, and Euclidean similarity-based GCN methods. The

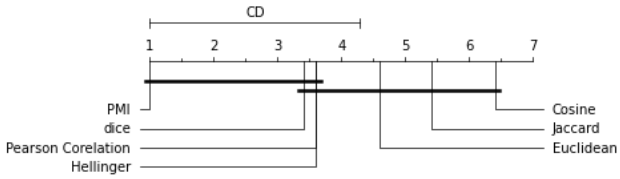

FIGURE 2. Demšar diagram compares different word-word weighing techniques for top-1 accuracy. They horizontal line shows the average rank scale. The critical distance (CD) is calculated with 95% confidence interval, which is 3.290.

PMI-GCN showed high accuracy results over the dice similarity, Pearson correlation, and Hellinger similarity based GCN methods; however, no significant difference was found. The Pearson correlation and Hellinger had the same average ranks. There was no significant difference between the dice, Pearson correlation, Hellinger, Cosine, Jaccard, and Euclidean based GCN.

RQ 2: Is the graph embedding better than context-insensitive word-embeddings?

Graph embedding is the transformation of a graph into a vector or set of vectors. The context-insensitive word embeddings have a constant vector for a word in any context. In contrast, the graph embedding technique comprehends the vertex-to-vertex relationship and other relative properties in graph representation or graph embedding. The graph representation of bug reports has bug documents and unique words as vertices or nodes. The context-insensitive embeddings do not take into account the context when converting a word into a vector. In the graph representation, the embedding transforms the word into vectors by learning the relation of vertices, which shows the relation of words with a bug document and other words. Therefore, a graph representation is a better option than context-insensitive embedding techniques such as GloVe and Word2vec. The experimental results in Table 1 also support our findings, which show that the proposed method with PMI outperforms all the context-insensitive embedding methods: Word2Vec-CNN, GloVe-CNN, DA-CNN, and Deeptriage.

Friedman's test was performed to check the significance of the experimental results. Then, Nemenyi post-hoc tests were performed to identify significant differences between the proposed method and other context-insensitive based triage methods (word2vec-CNN and GloVe-CNN). The test was performed for the top-1, top-5, and top-10 accuracies with a 95% confidence interval. The significance test showed a p value of less than 0.05, that indicated significance in the accuracy results. The Nemenyi test showed a significant difference between the proposed triage method and word2vec-CNN. The proposed method had better accuracy than the Glove-CNN. However, the Nemenyi test did not show a significant difference between the proposed triage method and GloVe-CNN, because the testing was conducted using limited datasets.

Thus, the significance test partially supports this research question. Overall, the proposed method showed good accuracy compared to GloVe-CNN and Word2Vec-CNN. However, the proposed triage system only showed significant differences from the word2vec-CNN triage method.

RQ 3: Is the proposed triage technique faster than the word representation-based approaches?

The graph has nodes and edges, where nodes are words and documents, and edges represent the relationships between them. Word vectors are generated against each word in context-aware and context-insensitive word representations. Each vector dimension is fixed; for example in the case of word2vec and Glove, the dimensions are 100 or 300, and in case of ELMo we can get 512 and 1024 dimensions. These methods require a large corpus of text data for efficient training. After training on a large corpus, these methods produce a vector in which each unique word is represented by a real valued vector.

Embedding approaches in graph learning move nodes to a high-dimensional vector space to optimize the chance of retaining node neighbors. One approach to do this is to establish an acceptable neighborhood by performing random walks starting from each node [39]. Another approach is to establish neighborhoods by edge creation between two nodes if both words/nodes are similar.

The graph embedding approach is faster than word representation techniques. As mentioned earlier, word representation techniques require large amounts of data for training. However, heterogeneous graph learning methods do not require much data for training. These methods calculate TF-IDF for word to document relation and similarity for word2word relation using existing information, which is a faster process than word embedding or word representation. It does not take significant time to build heterogeneous graph and learn the graph by GCN. The proposed method took on average of 35–40 minutes to build a graph and train for the Firefox [1] dataset, and 1 hour and 30 minutes for the Firefox [2] dataset. On a core i7 machine with 64 GB RAM and a Graphical Processing Unit (GPU), the recorded execution time decreased to 20 minutes for the Firefox [1] dataset. The heterogeneous graph generation task was performed on the central processing unit (CPU). The training task can be performed on either CPU or GPU.

RQ 4: Is the graph representation memory efficient for bug reports?

The experiments were carried out on a Core i7 machine with a GTX 1080Ti Nvidia GPU and 64GB RAM. The GPU had 12 GB dedicated memory. The proposed GCN based method was executed on a GPU for the Platform [1] and Firefox [1] datasets. The memory insufficient error occurred when the proposed method was executed for the large datasets, such as the Firefox threshold [3] and the Firefox [2] datasets. The heterogeneous graph is very large and requires a considerable amount of memory for execution. The GPU memory is limited; therefore, we cannot run the proposed method for large datasets.

The proposed method was executed on the CPU and produce results. We observed that the proposed method required

TABLE 2. Average top-1 to top-10 accuracy obtained on platform, Firefox-small [1], Firefox-thresholded [3] and Firefox [2] datasets.

Dataset	Techniques	top-1 Acc. %	top-2 Acc. %	top-3 Acc. %	top-4 Acc. %	top-5 Acc. %	top-6 Acc. %	top-7 Acc. %	top-8 Acc. %	top-9 Acc. %	top-10 Acc. %
Platform [1]	Lee et al. [1]	36.1	45.8	50.5	53.7	56.7	-	-	-	-	-
	Word2Vec-CNN [4]	38.036	48.870	55.160	58.220	62.492	65.604	67.632	69.324	70.832	71.714
	GloVe-CNN [4]	40.132	51.004	56.234	60.318	63.770	65.964	68.252	70.058	71.554	72.930
	ELMo-CNN [4]	43.622	51.830	56.366	**60.704**	**63.822**	**66.528**	**69.068**	**70.964**	**72.794**	**74.264**
	Previous Work [7]	39.54	50.00	56.09	59.90	62.01	64.40	65.42	67.49	69.87	72.11
	Our-PMI	**44.410**	**52.588**	**57.35**	60.559	63.458	64.7001	67.495	70.001	70.222	73.188
Firefox [1]	Lee et al. [1]	27.1	36.7	42.8	47.1	50.5	-	-	-	-	-
	Word2Vec-CNN [4]	27.396	37.894	44.256	48.346	51.604	54.422	57.006	58.858	60.430	61.750
	GloVe-CNN [4]	28.614	38.660	45.062	50.094	53.024	56.374	58.496	60.522	62.410	64.224
	ELMo-CNN [4]	30.418	41.104	47.81	52.508	55.738	58.362	60.404	62.314	63.598	64.696
	Previous Work [7]	29.37	41.81	47.84	53.89	57.03	59.52	61.91	63.82	68.16	66.50
	Our-PMI	**35.120**	**48.705**	**55.024**	**59.904**	**63.349**	**65.550**	**67.751**	**69.856**	**71.866**	**73.110**
Firefox [3] 0-Threshold	Deep Triage [3]	-	-	-	-	-	-	-	-	-	38.1
	Word2Vec-CNN [4]	15.2	23.52	28.77	32.07	35.15	37.34	39.56	41.37	42.89	43.63
	GloVe-CNN [4]	16.84	25.22	31.78	33.78	35.76	39.19	42.19	43.96	43.96	45.32
	ELMo-CNN [4]	**20.86**	**29.04**	**34.332**	**38.03**	**41.18**	**43.57**	45.72	47.68	49.28	50.73
	Previous Work [7]	15.48	22.18	27.67	30.89	34.03	36.63	38.49	40.68	42.09	43.60
	Our-PMI	19.849	28.414	34.041	37.544	40.784	43.51	**46.014**	**47.941**	**50.248**	**51.216**
Firefox [3] 10-Threshold	Deep Triage [3]	-	-	-	-	-	-	-	-	-	51.4
	Word2Vec-CNN [4]	19.19	27.55	33.58	39.05	41.41	43.85	45.13	46.02	47.90	51.06
	GloVe-CNN [4]	19.25	29.69	34.65	40.23	42.64	44.68	46.68	48.35	49.44	51.67
	ELMo-CNN [4]	31.01	42.85	49.83	54.50	57.96	60.77	63.78	64.92	66.62	67.90
	Previous Work [7]	25.661	34.55	36.458	40.004	44.078	48.415	52.845	56.474	60.78	63.14
	Our-PMI	**32.343**	**44.639**	**51.390**	**56.318**	**61.374**	**63.197**	**65.559**	**68.304**	**69.998**	**71.008**
Firefox [2]	DA-CNN [2]	12.44	19.09	22.54	24.91	26.93	28.26	30.17	31.71	33.00	34.46
	One-Hot+CNN [2]	8.16	15.69	19.67	20.768	23.57	25.19	27.18	28.35	30.11	32.86
	BOW+NB [2]	8.15	11.43	13.69	15.78	17.49	18.68	19.76	21.29	22.65	23.69
	Word2Vec-CNN [4]	12.74	17.83	21.96	25.70	27.41	29.31	31.24	32.92	34.53	35.76
	GloVe-CNN [4]	16.09	24.12	29.02	31.82	34.11	36.31	38.47	40.08	41.27	42.63
	ELMo-CNN [4]	16.73	25.18	30.15	33.98	36.47	38.95	40.89	42.37	44.05	45.40
	Previous Work [7]	18.126	27.035	32.719	36.866	40.015	42.704	45.161	46.237	47.619	48.201
	Our-PMI	**21.352**	**30.031**	**35.714**	**39.555**	**43.395**	**45.852**	**47.542**	**50.693**	**52.845**	**54.379**

more memory. However, it is computationally more efficient than the other CNN-based and RNN-based triage methods, because they require a GPU f or quick training. Otherwise, the CNN and RNN take a significant amount of time to train the models and are very slow to train on a CPU.

In summary, the proposed method required a significant amount of main memory for large-scale datasets because the whole heterogeneous graph has to be loaded into the main memory. However, it trains rapidly and does not require the GPU for training on large datasets.

E. COMPARISON WITH OTHER RESEARCH

We compared out method with some state-of-the-art methods using the same datasets. The comparison with the other

TABLE 3. The average top-1 to top-10 accuracy obtained on Wu *et al.*'s Eclipse dataset [6]. The dataset is split and 80:20 ratio for training and testing. The best performing values are shown in bold.

Top-k Accuracy	ITriage [6]	ST-DGNN [6]	Our PMI
Top-1	0.407 ± 0.013	0.680 ± 0.017	$\mathbf{0.6954 \pm 0.010}$
Top-2	-	-	0.8091 ± 0.013
Top-3	0.611 ± 0.016	0.774 ± 0.018	$\mathbf{0.8517 \pm 0.011}$
Top-4	-	-	0.8773 ± 0.010
Top-5	0.642 ± 0.014	0.875 ± 0.012	$\mathbf{0.8917 \pm 0.009}$
Top-6	-	-	0.9011 ± 0.007
Top-7	-	-	0.9181 ± 0.007
Top-8	-	-	0.9317 ± 0.006
Top-9	-	-	0.9353 ± 0.005
Top-10	-	-	0.9483 ± 0.005

papers is very complicated, because every researcher used different datasets, and their datasets are not publicly available. Although they described the data collection and cleaning process, it is difficult to get the same data used in their research.

We used a cosine similarity metric in previous work. However, different similarity matrixes are used in current research for word–word edge weighting. Our proposed triage method with PMI showed results superior to the comparative studies. For the Platform dataset, the proposed method showed good results for top-1 to top-3 accuracy. From top-4 to Top-10, ELMo-CNN [4] beat the proposed triage method with PMI. Firefox [1] is a larger dataset than the Platform, with a large number of developers, and in this case the proposed method beat other methods with a noticeable difference. These observations show that the proposed method performed well for large datasets where the average ratio of bug reports per developer was at least 20. The Firefox-0 Threshold data is a very big dataset, and includes developers that have at least one bug report in the history. The proposed method demonstrated lower performance than ELMo-CNN method for top-1 to top-6 accuracy. After top-6 accuracy, the proposed method showed good performance up to the top 10 accuracy, with a noticeable difference. Similar findings were found for the 10 threshold and other datasets.

The Eclipse dataset [6] is a massive dataset that has a significant number of bug reports and has many developer classes. The proposed method had good top-k accuracy compared to a spatial-temporal dynamic graph neural network (ST-DGNN) [6] and ITriage [40]. In ST-DGNN, the authors used the JRWalk mechanism to embed nodes in homogeneous networks. Thus, their approach was limited to homogeneous dynamic graph networks, and could not be directly applied to heterogeneous graph networks. In contrast, the proposed approach used a heterogeneous graph, which has recently attracted more attention.

In summary, the proposed method demonstrated better performance than the comparative studies. The proposed method achieved better top-k accuracy than comparative studies for all datasets. The proposed method showed top-1 to top-5 accuracy comparable to that of other studies for datasets with a smaller number of bug reports per developer, while a noticeable difference was found in top-6 to top-10 accuracy for all datasets. The proposed method shows better top-10 accuracy than other considered methods. However, the Friedman test and Nemenyi post hoc test shows insignificant difference between the proposed method and ELMo-CNN.

VI. LIMITATIONS

Different types of graph-based approaches, including tossing, dynamic, relational, and homogeneous approaches, have been proposed for bug triage. To the best of our knowledge, no heterogeneous graph-based approach has been proposed until the present study. However, he proposed approach is limited because it cannot add new developers to the trained model without retraining from scratch. Retraining from scratch is required to build a new model to add new developers/classes, a process which is very time consuming. In the future, we intend to find a solution based on a heterogeneous graph for bug triage that can add new developer classes without retraining from scratch.

The proposed approach is not cost-effective. It requires significant memory and time for very large datasets. The heterogeneous graph has word-to-word and word-to-report associations, which make the graph significantly giant. The entire graph is loaded into memory for training the GCN, which entails significant memory costs for massive datasets. Moreover, the proposed approach requires significant training time for significantly large datasets, because training is not possible on GPU systems. The GPU has limited memory and cannot be extended externally. Furthermore, we cannot load heterogeneous graphs in chunks to GPU memory. Therefore, the proposed approach takes considerable time to train on the CPU.

VII. THREATS TO VALIDITY

A. CONSTRUCT VALIDITY

Bug reports are publicly available in a bug repository. Researchers download bug reports from repositories using REST API to make a dataset. Then, they filter or clean the dataset. Reproducing data is difficult, because some bug reports are duplicated, and their status changes over time. Therefore, we used publicly available or published datasets to estimate the performance of the proposed bug triage method. The same protocols used in other studies were used for splitting datasets into training and test sets. Therefore, we expected no threats to construct validity for this research.

B. INTERNAL VALIDITY

The performance of the method was validated on published data by comparative studies. No new data were collected for this research. Previous studies' researchers collected data

from open bug repositories with closed and fixed status. They ensured that all bug reports were publicly available. We also ensured that the bug reports were publicly available for the specific open-source large-scale projects. Therefore, we expected no internal threats to validity.

C. EXTERNAL VALIDITY

Only a few open-source projects were used in our research. Therefore, the result may not be applicable to all open-source and industrial projects. The datasets were collected from the Bugzilla open bug repository. Therefore, we can say the proposed method is scaleable for those projects which use the Bugzilla repository. However, industrial projects are entirely different, and their triage process may also differ from that applicable to other open-source projects. The proposed method was not tested on industrial projects. Nevertheless, we hope that the method can be applied to industrial projects, because every bug report has a summary and description across the platforms. Thus, we hope that there is no external threats to the validity of this research.

Another limitation of this research is the comparison with few studies and datasets. The comparison of the proposed method with all the recent studies was impossible because most researchers use their own datasets, which are not publicly available to other researchers. Most researchers provide the data source URL, time interval, resolution status, and the number of bug reports. They often do not explain the data cleaning process and parameters, which makes it challenging to reproduce the same dataset. Also, no one can prove the authenticity of the dataset. Therefore, we only used publicly available datasets to evaluate the performance of the methods, despite limited comparison to those studies with published data.

VIII. CONCLUSION

This paper extends our previous work by adopting JS, Euclidean similarity, dice similarity, Hellinger similarity, Pearson correlation, and point-wise mutual information for weighting word–word edges. TF-IDF was used for word-document edge weighting. Then, a simple GCN was used to learn the heterogeneous graph of bug reports that generated a graph representation of bug data and assigned a list of developers to a reported bug.

The experimental results suggest that point-wise mutual information is the best method for weighting the word–word edges. Experimental results with PMI showed significant distance with cosine, Jaccard, and Euclidean similarities. PMI was not found to be significantly different from Pearson correlation, dice similarity, and Hellinger similarity according to the calculated critical distance. However, PMI showed the best results on all datasets and showed 3 to 6% higher top-1 accuracy and 5 to 8% top-10 accuracy than state-of-the-art-methods.

The platform dataset is the smallest dataset. The proposed method showed results comparable to those of ELMo-CNN. The other datasets are larger than the platform dataset. The proposed method showed better top-k accuracy on all other datasets. The proposed method showed a slight difference in top-1 accuracy; however, it showed a very large difference from the other methods on all datasets except the Platform data for top-10 accuracy.

The proposed method was found to be faster than the other deep learning methods, because graph embedding techniques do not require as much data for training as CNN and RNN techniques. Moreover, sophisticated GPUs are not required to train a GCN, because it can be easily trained on a CPU. In contrast, the proposed method requires more primary memory, because the whole heterogeneous graph is loaded into memory for execution. The heterogeneous graph is large for massive datasets, and GPUs have limited dedicated memory; therefore, we could not train the heterogeneous graph of large datasets on GPUs.

The proposed method is not memory efficient, because it requires significant memory and time for very large datasets. We intend to find a possible solution to make it cost-effective in the future. We also intend to extend our work to add new developer classes to the existing model without retraining from scratch.

REFERENCES

[1] 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.*, New York, New York, USAs, 2017, pp. 926–931.

[2] 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, 2020.

[3] 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.

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

[5] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," 2016, *arXiv:1609.02907*.

[6] H. Wu, Y. Ma, Z. Xiang, C. Yang, and K. He, "A spatial-temporal graph neural network framework for automated software bug triaging," 2021, *arXiv:2101.11846*.

[7] S. F. A. Zaidi and C.-G. Lee, "Learning graph representation of bug reports to triage bugs using graph convolution network," in *Proc. Int. Conf. Inf. Netw. (ICOIN)*, Jan. 2021, pp. 504–507.

[8] H. Kagdi, M. Gethers, D. Poshyvanyk, and M. Hammad, "Assigning change requests to software developers," *J. Softw., Evol. Process*, vol. 24, no. 1, pp. 3–33, Jan. 2012.

[9] R. Shokripour, Z. M. Kasirun, S. Zamani, and J. Anvik, "Automatic bug assignment using information extraction methods," in *Proc. Int. Conf. Adv. Comput. Sci. Appl. Technol. (ACSAT)*, Nov. 2012, pp. 144–149. [Online]. Available: http://ieeexplore.ieee.org/document/6516342/

[10] 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 Workshop Conf. Mining Softw. Repositories (MSR)*, May 2013, pp. 2–11. [Online]. Available: http://ieeexplore.ieee.org/document/6623997/

[11] S. Banitaan and M. Alenezi, "DECOBA: Utilizing developers communities in bug assignment," in *Proc. 12th Int. Conf. Mach. Learn. Appl. (ICMLA)*, vol. 2, Dec. 2013, pp. 66–71.

[12] T. Zhang and B. Lee, "A hybrid bug triage algorithm for developer recommendation," in *Proc. 28th Annu. ACM Symp. Appl. Comput.*, 2013, pp. 1088–1094.

[13] H. Hu, H. Zhang, J. Xuan, and W. Sun, "Effective bug triage, based on historical bug-fix information," in *Proc. IEEE 25th Int. Symp. Softw. Rel. Eng. (ISSRE)*, Nov. 2014, pp. 122–132. [Online]. Available: http://ieeexplore.ieee.org/document/6982620/

[14] H. Naguib, N. Narayan, B. Brugge, and D. Helal, "Bug report assignee recommendation using activity profiles," in *Proc. 10th Work. Conf. Mining Softw. Repositories (MSR)*, May 2013, pp. 22–30.

[15] 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.

[16] T. Zhang, G. Yang, B. Lee, and E. K. Lua, "A novel developer ranking algorithm for automatic bug triage using topic model and developer relations," in *Proc. 21st Asia–Pacific Softw. Eng. Conf.*, Dec. 2014, pp. 223–230.

[17] S. Wang, W. Zhang, and Q. Wang, "FixerCache: Unsupervised caching active developers for diverse bug triage," in *Proc. 8th ACM/IEEE Int. Symp. Empirical Softw. Eng. Meas. (ESEM)*, 2014, pp. 1–10.

[18] X. Xia, D. Lo, Y. Ding, J. M. Al-Kofahi, T. N. Nguyen, and X. Wang, "Improving automated bug triaging with specialized topic model," *IEEE Trans. Softw. Eng.*, vol. 43, no. 3, pp. 272–297, Mar. 2017.

[19] W. Zhang, Y. Cui, and T. Yoshida, "En-LDA: An novel approach to automatic bug report assignment with entropy optimized latent Dirichlet allocation," *Entropy*, vol. 19, no. 5, p. 173, Apr. 2017.

[20] A. Yadav, S. K. Singh, and J. S. Suri, "Ranking of software developers based on expertise score for bug triaging," *Inf. Softw. Technol.*, vol. 112, pp. 1–17, Aug. 2019.

[21] M. Kumari, A. Misra, S. Misra, L. F. Sanz, R. Damasevicius, and V. B. Singh, "Quantitative quality evaluation of software products by considering summary and comments entropy of a reported bug," *Entropy*, vol. 21, no. 1, p. 91, Jan. 2019.

[22] P. Bhattacharya and I. Neamtiu, "Fine-grained incremental learning and multi-feature tossing graphs to improve bug triaging," in *Proc. IEEE Int. Conf. Softw. Maintenance*, Sep. 2010, pp. 1–10.

[23] 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.

[24] J. Xuan, H. Jiang, Z. Ren, and W. Zou, "Developer prioritization in bug repositories," in *Proc. 34th Int. Conf. Softw. Eng. (ICSE)*, 2012, pp. 25–35.

[25] S. Banitaan and M. Alenezi, "TRAM: An approach for assigning bug reports using their metadata," in *Proc. 3rd Int. Conf. Commun. Inf. Technol. (ICCIT)*, Jun. 2013, pp. 215–219.

[26] R. Johnson and T. Zhang, "Supervised and semi-supervised text categorization using LSTM for region embeddings," 2016, *arXiv:1602.02373*.

[27] A. C. Florea, J. Anvik, and R. Andonie, "Spark-based cluster implementation of a bug report assignment recommender system," in *Artificial Intelligence and Soft Computing* (Lecture Notes in Computer Science), vol. 10246, L. Rutkowski, M. Korytkowski, R. Scherer, R. Tadeusiewicz, L. Zadeh, and J. Zurada, Eds. Cham, Switzerland: Springer, 2017, doi: 10.1007/978-3-319-59060-8_4.

[28] M. Alenezi, K. Magel, and S. Banitaan, "Efficient bug triaging using text mining," *J. Softw.*, vol. 8, no. 9, pp. 2185–2191, 2013. [Online]. Available: http://www.jsoftware.us/vol8/jsw0809-12.pdf

[29] M. Alenezi, S. Banitaan, and M. Zarour, "Using categorical features in mining bug tracking systems to assign bug reports," 2018, *arXiv:1804.07803*.

[30] Y. Zhao, T. He, and Z. Chen, "A unified framework for bug report assignment," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 29, no. 4, pp. 607–628, Apr. 2019.

[31] T. S. Mian, "Automation of bug-report allocation to developer using a deep learning algorithm," in *Proc. Int. Congr. Adv. Technol. Eng. (ICOTEN)*, 2021, pp. 1–7.

[32] T. W. W. Aung, Y. Wan, H. Huo, and Y. Sui, "Multi-triage: A multi-task learning framework for bug triage," *J. Syst. Softw.*, vol. 184, Feb. 2022, Art. no. 111133.

[33] 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.

[34] 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.

[35] H. Jahanshahi, K. Chhabra, M. Cevik, and A. Bąbar, "DABT: A dependency-aware bug triaging method," in *Proc. Eval. Assessment Softw. Eng.*, 2021, pp. 221–230.

[36] R. A. Khurma, H. Alsawalqah, I. Aljarah, and M. A. Elaziz, "An enhanced evolutionary software defect prediction method using island moth flame optimization," *Mathematics*, vol. 9, no. 15, p. 1722, Jul. 2021.

[37] L. Yao, C. Mao, and Y. Luo, "Graph convolutional networks for text classification," in *Proc. AAAI Conf. Artif. Intell.*, vol. 33, 2019, pp. 7370–7377.

[38] S. Niwattanakul, J. Singthongchai, E. Naenudorn, and S. Wanapu, "Using of Jaccard coefficient for keywords similarity," in *Proc. Int. Multiconf. Eng. Comput. Sci.*, 2013, vol. 1, no. 6, pp. 380–384.

[39] R. Cappuzzo, P. Papotti, and S. Thirumuruganathan, "Creating embeddings of heterogeneous relational datasets for data integration tasks," in *Proc. ACM SIGMOD Int. Conf. Manage. Data*, Jun. 2020, pp. 1335–1349.

[40] S.-Q. Xi, Y. Yao, X.-S. Xiao, F. Xu, and J. Lv, "Bug triaging based on tossing sequence modeling," *J. Comput. Sci. Technol.*, vol. 34, no. 5, pp. 942–956, Sep. 2019.

SYED FARHAN ALAM ZAIDI was born in Lahore, Pakistan, in 1993. He received the B.S. degree in information technology from the University of Education, Lahore, Pakistan, in 2014, and the M.S. degree in computer science from COMSATS University Islamabad, Pakistan, in 2017. He is currently pursuing the Ph.D. degree in computer science and engineering with Chung-Ang University, Seoul, South Korea. After completing the B.S. degree, he worked as a Web Developer with a software development company in Pakistan. After completing the M.S. degree, he involved with teaching as a Visiting Lecturer in various universities. Along with the Ph.D. degree, he is also working as a Research Assistant with the Real-Time Software Engineering Laboratory. His research interests include software engineering-based problems, natural language processing, deep/machine learning, data mining, image processing, and medical imaging.

HONGUK WOO was born in Seoul, South Korea. He received the B.S. degree in computer science from Korea University, Seoul, in 1995, and the M.S. and Ph.D. degrees in computer sciences from The University of Texas at Austin, Austin, TX, USA, in 2002 and 2008, respectively. From 2008 to 2018, he worked at Samsung Research of Samsung Electronics as a Principal Engineer and the Vice President. Since 2018, he has been an Assistant Professor with the Department of Software, Sungkyunkwan University, Suwon, South Korea. He is the coauthor of more than 30 research papers and ten patents. His research interests include data-centric application, analytic monitoring & intelligence, and networked cyber-physical systems.

CHAN-GUN LEE was born in Seoul, South Korea, in 1972. He received the B.S. degree in computer engineering from Chung-Ang University, Seoul, in 1996, the M.S. degree in computer science from the Korea Advanced Institute of Science and Technology (KAIST), Daejeon, in 1998, and the Ph.D. degree in computer science from The University of Texas at Austin, Austin, TX, USA, in 2005. From 2005 to 2007, he was a Senior Software Engineer with Intel, Hillsboro, Oregon. Since 2007, he has been a Professor with the Department of Computer Science and Engineering, Chung-Ang University. He is the author of more than 30 articles and conference papers. His research interests include software engineering and real-time systems. He was a recipient of the Korea Foundation of Advanced Studies (KFAS) Fellowship, from 1999 to 2005.