Hindawi
Journal of Sensors
Volume 2022, Article ID 4347004, 19 pages
https://doi.org/10.1155/2022/4347004

Research Article

Toward an Effective Bug Triage System Using Transformers to Add New Developers

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, Republic of Korea*
[2]*Department of Computer Science and Engineering, Chung-Ang University, Seoul 06974, Republic of Korea*
[3]*Department of Software, Sungkyunkwan University, Suwon 16419, Republic of Korea*

Correspondence should be addressed to Chan-Gun Lee; cglee@cau.ac.kr

Received 21 December 2021; Accepted 2 March 2022; Published 8 April 2022

Academic Editor: Sangsoon Lim

As defects become more widespread in software development and advancement, bug triaging has become imperative for software testing and maintenance. The bug triage process assigns an appropriate developer to a bug report. Many automated and semiautomated systems have been proposed in the last decade, and some recent techniques have provided direction for developing an effective triage system. However, these techniques still require improvement. Another open challenge related to this problem is adding new developers to the existing triage system, which is challenging because the developers have no listed triage history. This paper proposes a transformer-based bug triage system that uses bidirectional encoder representation from transformers (BERT) for word representation. The proposed model can add a new developer to the existing system without building a training model from scratch. To add new developers, we assumed that new developers had a triage history created by a manual triager or human triage manager after learning their skills from the existing developer history. Then, the existing model was fine-tuned to add new developers using the manual triage history. Experiments were conducted using datasets from well-known large-scale open-source projects, such as Eclipse and Mozilla, and top-k accuracy was used as a criterion for assessment. The experimental outcome suggests that the proposed triage system is better than other word-embedding-based triage methods for the bug triage problem. Additionally, the proposed method performs the best for adding new developers to an existing bug triage system without requiring retraining using a whole dataset.

1. Introduction

Bug triage is a challenging problem in software engineering, as it aims to assign an appropriate developer to a specific bug report. Bugs include defects, errors, or loopholes that should be fixed quickly to improve the software quality. Testing or quality assurance engineers identify bugs in the testing and maintenance phase, reporting bugs in the form of a document known as a bug report. Then, they report bugs in open bug repositories, such as Bugzilla or JIRA. Afterward, the developers and engineers use open bug repositories to explore ways to repair the bugs. Well-known large-scale open-source projects, such as Mozilla, Eclipse, and NetBeans, use open bug repositories for bug report management [1].

A bug report has considerable information about the bug, including a short description (summary), long description, creation date and time, components, product, creator, assignee, comments, status, resolution, priority, severity, and other information. A manual triager or triage manager assigns developers to specific bug reports, which is very time-consuming. It is challenging for the triage manager to remember the developers' skills and assign a bug to a suitable developer.

Many automatic bug triage techniques have been proposed to solve this problem in the last decade. Nevertheless, these techniques are still far from satisfactory and require improvement. Several researchers have solved bug triage problems using mining repositories, social network analysis,

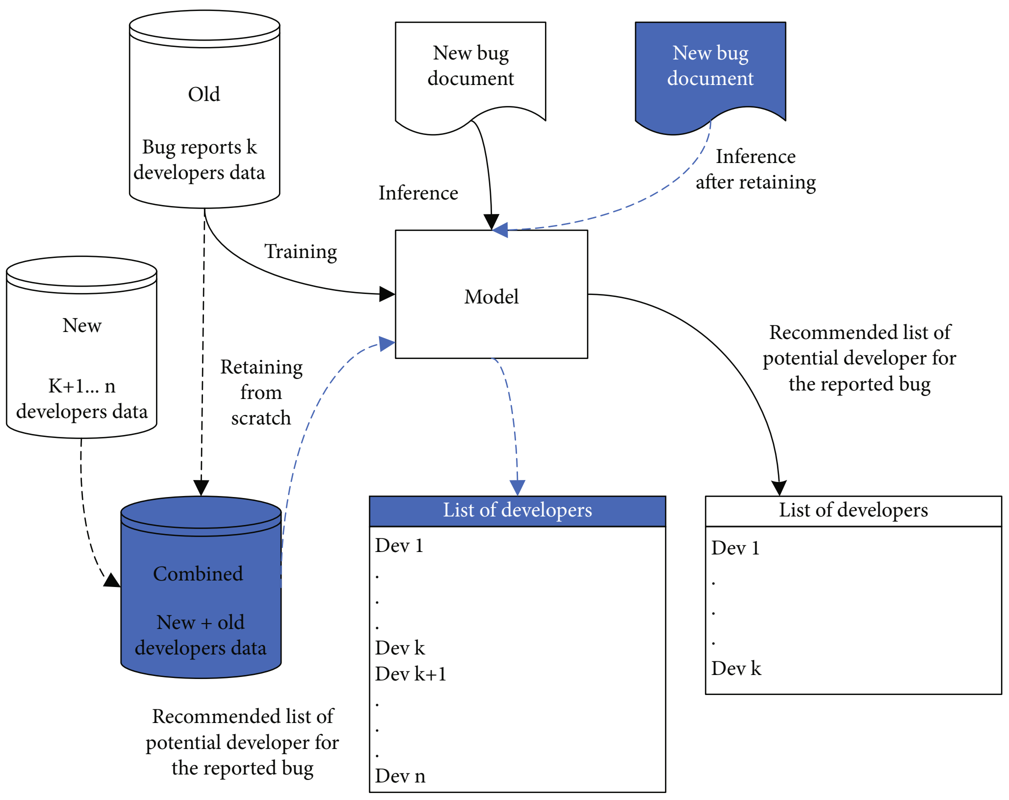

Old
Bug reports k developers data
New bug document
New bug document
Inference
Inference after retaining
Training
Model
New
K+1... n developers data
Retaining from scratch
Recommended list of potential developer for the reported bug
Combined
New + old developers data
List of developers
Dev 1
Dev k
Dev k+1
Dev n
List of developers
Dev 1
Dev k
Recommended list of potential developer for the reported bug

FIGURE 1: The process of existing deep learning-based bug triage methods.

topic modeling, dependency-based methods, traditional machine learning methods, and deep learning methods, demonstrating promising results for small datasets. The detailed literature is provided in Section 2.

Since the evolution of machine learning, deep learning-based methods have presented promising results in many fields, for example, the recent results for large-scale datasets. Deep learning methods use natural language processing (NLP) to convert text into vectors, and word-embedding techniques are crucial tools for NLP methods. Vectors are input into deep learning models for training. S. Lee et al. [2] were the first to use a convolutional neural network (CNN) method for bug triage. They used word2vec embedding for vectorization, a CNN with three different kernels to extract features, and a dense network with a SoftMax classifier for the classification task.

Mani et al. [3] proposed an attention-based bidirectional recurrent neural network (RNN) that also used word2vec embedding. S. Guo et al. [4] and Zaidi et al. [5] used a CNN model. The first model was based on developer activities. Moreover, S. Guo et al. applied their methods on large datasets with numerous developers. The latter model adopted different word-embedding techniques, including word2vec, global vector (GloVe), and the Embeddings from Language Model (ELMo), to improve the triage accuracy. The models exhibited promising results compared to the state-of-the-art methods. Thus, deep learning models perform better for large datasets and numerous developers. Nevertheless, top-1 accuracy is still far from perfect. Furthermore, top-10 accuracy requires improvement to create a reliable recommendation system.

Some typical bug fixer recommendation systems can add newly hired developers for triage bugs. These recommendation systems add new developers' expertise to the recommendation system. Then, the recommendation systems can assign new developers to bug reports. However, for neural network-based and deep learning-based bug triage methods, adding newly hired developers to an existing trained model is challenging because new developers do not have a triage history. Moreover, the existing deep learning-based triage techniques require retraining the model from scratch on the entire dataset (new and old class data). Figure 1 is the pictorial representation of the existing machine learning-based bug triage solutions. The model is built after training on k developers' data. When a new bug report arrives, the model recommends a ranked list of k developers. If a company hires new developers, the model cannot assign bug reports to new developers until the model is retrained on both old and new developer datasets. After retraining from scratch, the model can assign reports to new developers. Retraining a model from scratch is a very time-consuming approach. In addition, whenever new developers are available, retraining from scratch is not a good choice. The blue color in the figure demonstrates the process of the existing deep learning-based approaches when the new developer data are available.

In the literature, we did not find an appropriate bug triage solution based on deep learning techniques that can add newly hired developers to the existing model without retraining from scratch. Zaidi et al. [6] aimed to solve this problem using a one-class classification technique, which is a machine learning technique. They used a one-class support vector machine (SVM) classifier for one-class classification and trained a separate classifier for each developer. The selected developer for the classifier was labeled as a positive example, and all others were labeled as negative examples. The classifier learned only the positive examples and classified the negative examples as outliers. Their method was overfitted to the training data and classified some parts of the testing data of a positive class as outliers. Furthermore, the method could not rank developers for a bug report because the classifier output was binary, and the classifier only classified samples as +1 or −1. This technique failed to offer an appropriate solution for the stated problem.

The aim of this research is twofold. First, we focus on increasing the accuracy of the bug triage system. Second, the proposed triage system can add newly hired developers to the existing trained model. Therefore, a transformer-based bug triage method that can add new developers to an existing triage model is proposed to overcome the mentioned challenges. The proposed method has two parts: the base model and fine-tuned model. The base model trains on base data and recommends a ranked list of developers. The fine-tuned model is used to add new developers to the existing system and uses the concept of continuous fine-tuning, which was used by Käding et al. [7] for image data. The fine-tuned model learns new developers when their data are available. Various context-insensitive and context-aware word representation techniques have been used for bug triage in the literature. However, the bidirectional encoder representations from transformer (BERT) embedding is effective compared to other embedding techniques for classification tasks. Therefore, we chose the BERT embedding model for effective word representation. To validate the proposed fine-tuned method, we constructed datasets from publicly available data. As discussed above, new developers do not have any triage history. For a realistic scenario, we separated the developers who have three bug reports at most from the data. We assumed that these developers are new and that a manual triager assigns bug reports according to the developers' skills to create the triage history. Furthermore, this history is used to add new developers to the existing model.

The key advantage of the proposed method is to accommodate new developers to the existing model without retraining from scratch. Our triage system uses the continuous fine-tuning technique, which extends the base model with extended developer classes. The fine-tuning technique requires less iterations for training than a method that builds the model from scratch for new and old developers' classes. It can be applied to any software project managing bug reports with summaries and descriptions.

The main contributions of this research are shown in the following:

(i) We propose a transformer-based triage method that recommends a ranked list of 10 developers with notable top-k accuracy. To the best of our knowledge, no method in the literature used transformers for the bug triage problem

(ii) Adding new developers to the existing deep learning-based model is a challenge in bug triage because existing studies require retraining the model on old and new developer datasets. To the best of our knowledge, no published approach addresses the problem of adding new developers to the existing model based on the deep neural network in the context of bug triage. The proposed study suggests one possible solution to adding new developers in the existing trained deep learning model using the continuous fine-tuning concept

(iii) We evaluate the performance of the proposed bug triage technique using state-of-the-art triage techniques with publicly available data

2. Related Work

Many studies in the last decade have assessed bug triage problems. In this section, we present some recent studies on bug triage classified into four main categories: information retrieval-based bug triage systems, social network analysis-based bug triage systems, dependency-based bug triage systems, and machine learning-based bug triage systems. The machine learning-based bug triage methods are further categorized as conventional machine learning methods and deep learning methods.

Table 1 lists selected recent triage methods proposed in the last five years. Only a few selected recent papers used accuracy or top-k accuracy metrics to evaluate bug assignment to developers or fixer recommendations for bug reports. The extensive literature proposed in the last decade related to the bug triage problem is described in this section.

2.1. Information Retrieval-Based Bug Triage Systems. Information retrieval-based approaches consider developers with similar sufficient expertise for a specific type of bug report to solve new bug reports of a similar type. Different information retrieval-based bug triage techniques, including mining software repositories and topic modeling, have been proposed to solve the bug triage problem. This section presents relevant literature on information retrieval techniques for bug triage.

Software repositories contain historical information on system development and maintenance. Researchers have mined this historical information, such as the source code and version control repository, to assign an appropriate developer to a bug report. Kagdi et al. created a corpus for every source code file using descriptions, commits, source code, and class or method identifiers from bug reports as data sources. They used latent semantic indexing as an information retrieval technique to index the corpus. An indexed corpus was used to compute the similarity in the bug report descriptions to predict the related source files. Their approach recommends developers based on their activities with predicted source files in the version repository [8].

Table 1: Literature review related to bug triage in the last five years.

Author	Year	Selected information from bug reports	Word representation or vectorizer	Method	Dataset	Top-1 Acc. %	Top-5 Acc. %	Top-10 Acc. %
Dedik and Rossi [41]	2016	Summary and description	TF-IDF	Feature extracted by TF-IDF and classification task done by SVM	Firefox OSS	59	—	97
					Industrial data	54	—	90
Jonsson et al. [35]	2016	Summary and description	TF-IDF	Feature extracted by TF-IDF and assign developer by stacked generalization of a classifier ensemble	Industrial	89	—	—
Xuan et al. [42]	2017	Summary and description	TF-IDF	Tokenized as list of words and used Naïve Bayes classifier and expectation-maximization	Eclipse	21.04	48.07	—
Peng et al. [43]	2017	Summary and description	—	Used inverted indexing to sort list of terms from bug reports and relevant search technique for developer recommendation	Mozilla Eclipse	22.3 30.7	51.8 61.3	— —
S. Lee et al. [2]	2017	Summary and description	Word2Vec	Used Word2Vec for word representation and CNN-based classifier for assigning developers	Eclipse (JDT) Eclipse (platform)	46.6 36.1	74.3 56.7	— —
					Firefox	27.1	50.5	—
Yin et al. [1]	2018	Summary, description and comments of developer	TF-IDF	Used TF-IDF weighting for constructing feature space and genetic algorithm based optimal extreme learning machine (ELM) for classification tasks with diversified features	Bugzilla Eclipse NetBeans	62.4 65.8 71.2	— — —	— — —
					GCC	63.3	—	—
Mani et al. [3]	2019	Summary and description	Word2Vec	Proposed bidirectional RNN with attention mechanism and Word2Vec for conversion in vector	Mozilla Firefox Mozilla Core	— —	— —	46.6 ± 6.4 38.8 ± 3.2
					Google Chrome	—	—	42.7 ± 3.5
Yadav et al. [18]	2019	Meta data including priority, product, and time-stamp	—	Computed developer expertise scores (DES) from meta data and assign developers to new bug reports based on similarity measure	Mozilla Eclipse NetBeans	90.38 89.97 88.53	— — —	— — —
S. Guo et al. [4]	2020	Summary and description	Word2Vec	Developer activity-based CNN	Eclipse Mozilla NetBeans	25.20 12.44 35.54	46.03 26.93 62.66	53.37 34.46 74.89
Zaidi et al. [5]	2020	Summary and description	Word2vec, GloVe, and ELMo	Proposed triage methods based on three different word representation techniques with CNN model: Word2vec-CNN, GloVe-CNN, and ELMo-CNN	JDT Platform Firefox Mozilla NetBeans	49.750 43.622 30.418 16.73 40.17	78.261 63.822 55.738 36.47 66.07	87.234 74.264 64.696 45.40 74.23
Zaidi and Lee [44]	2021	Summary and description	TF-IDF for word-document edge and cosine similarity for word-word edge weighing	Built a heterogeneous graph and used GCN for fixer assignment	JDT Platform Firefox	47.01 39.54 29.37	76.86 62.01 57.03	84.00 72.11 66.50

Shokripour et al. proposed an automatic bug assignment approach using extracted information from software repositories. Their method used a version control repository to extract the bug-required information and best bug fixer. The authors employed the phrase composition technique from commits and descriptions. The system recommends developers based on the activity histories in the files with the most similar phrase composition [9]. In another study [10], the authors used a noun extraction method on several information sources, such as commit messages, comments, and source code, that determined the bug location. The term-weighting scheme to predict the files belongs in the new bug report. Finally, their proposed triage system recommends a developer based on the developer's expertise with the predicted files.

Linares-Vásquez et al. used identifiers, comments, and author information on source code files to create a corpus. Latent semantic indexing indexes the corpus and computes the similarity between the files and bug description, recommending the author of the top-N similar files [11].

Some techniques model developer expertise using their comments, reports, and bug-fixing activities. Naguib et al. proposed a technique that leverages topic modeling and the developer's activities, such as fixing, reviewing, and assigning bug reports. Their technique recommends developers based on their association scores on the topic determined using the developers' activities [12].

Yang et al. introduced topic modeling and multiple feature-based approaches that extract a set of candidate developers who have participated in bug reports with the same topics and features. Then, the developers are ranked by a score determined using their activities [13]. S. Wang et al. proposed an unsupervised method that caches the developers based on their component-level activities. The approach calculates the score of activeness for specific periods in the cache. The approach recommends the appropriate developer for a bug report using an activeness score [14].

In addition, Zhang et al. [15] combined the topic model and the relationship between a bug fixer and the report. The authors calculated the correlation score of the developer with the report and topic models. Their approach recommends developers based on the correlation score.

Furthermore, Xia et al. focused on activity model techniques and extended the latent Dirichlet allocation (LDA) topic modeling algorithm. They proposed a multifeature LDA algorithm, which includes components and products. Developers are recommended based on their affinity score [16].

Zhang et al. proposed an entropy-based optimized LDA approach to build a topic model for automatic bug report assignments. They used the Stanford topic modeling toolbox to train the topic model with the optimized LDA. The developer's comments facilitated modeling the developer's expertise and interest in a topic. The entropy-optimized LDA recommends a ranked list of developers [17].

Yadav et al. proposed an approach that ranks developers based on their expertise in triaging bugs. They reduced the bug tossing length with their approach and built developer profiles based on their contribution performance. This approach generates developer expertise scores using the average fixing time, priority-weighted fixed issues, and index metrics. Additionally, this approach determines feature-based, cosine, and Jaccard similarities to compute the expertise scores. A ranked list of developers for new incoming bug reports is recommended [18].

Further, Lee and Seo [19] proposed a method to improve triage performance by improving LDA. To improve the existing LDA topic sets, they built two adjunct topic sets using multiple LDA-based topic sets. Their method achieved good accuracy compared to the existing LDA-based methods.

2.2. Social Network Analysis-Based Bug Triage Systems. Some researchers have used the social network analysis approach to solve the bug triage problem. In the software industry, developers collaborate on the bug resolution process. Social network analysis techniques consider the relationships between developers and bug reports to determine an appropriate developer. Researchers have used developers as nodes and collaboration as edges to model this problem. For example, Banitaan and Alenezi proposed the Developers Communities in Bug Assignment (DECOBA), building a developer social network based on comments on bugs and detecting developer communities. Their system recommends developer communities for new bug reports by ranking developers [20].

In addition, Zhang et al. combined social network analysis with machine learning to triage bugs. The numbers of bug fixes, comments, and reports to compute the developer's contribution score are added to the classifier score to assign the appropriate developer [21]. In another study, Zhang and Lee used an information retrieval technique to determine similar bugs. The system recommends a fixer based on the fixing probability and fixing experience. The fixing probability is determined using a social network technique, and the fixing experience is calculated based on the number of fixed and assigned bugs [22].

Moreover, Hu et al. proposed a bug-fixing technique that computes the similarities to other bug reports. The technique recommends developers based on associations between developers, components, and bugs [23].

2.3. Dependency-Based Bug Triage Systems. Kumari et al. solved the bug triage problem using a bug dependency-based mathematical model because bug dependencies exist due to coding errors, architecture faults, and misunderstandings between users and developers. Furthermore, this model uses the summary, description, and comments from the bug report to measure entropy. This model assigns developers based on entropy [24].

Etemadi et al. proposed a scheduling-driven approach that efficiently assigned bug-fixing tasks to developers. They used a task dependency graph; each task is associated with a node in a task dependency graph. A starting time and an ending time are assigned to each task. The method used an embedded greedy search that operates over the schedules to explore more parts of search space. Their approach showed good accuracy as well as reduced the bug-fixing time [25].

Recently, Almhana and Kessentini [26] proposed an automated bug triage method that considers dependencies between bug reports. Then, they localized the files to be inspected for each open bug report. The multiobjective search is used to rank bug reports for programmers based on dependencies for other reports and priorities. Their approach demonstrated a significant time reduction of over 30% in localizing bugs simultaneously compared to traditional bug prioritizing techniques.

Jahanshahi et al. proposed a dependency-aware bug triage method unlike the previous dependency-based methods. They considered NLP and integer programming to assign bugs appropriately. The method used textual information, dependency between bugs, and the cost associated with each bug. Their technique reduced the 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 [27].

2.4. Machine Learning-Based Bug Triage Systems. Many methods based on machine learning and deep learning have been proposed throughout the last decade. These techniques consider a bug report to be a learning problem and solve the bug triage problem as a classification problem. We divided the machine learning methods into conventional machine learning-based bug triage systems and deep learning-based bug triage systems.

2.4.1. Conventional Machine Learning-Based Bug Triage Systems. These methods use different feature extraction techniques, such as term frequency-inverse document frequency (TF-IDF), discriminating terms using the chi-square, term selection, and mutual information. These techniques use different well-known machine learning algorithms to assign a suitable fixer to the reported bug. For example, Bhattacharya and Neamtiu proposed a bug triage solution using titles, summaries, descriptions, and additional attributes. They extracted features from these selected fields using TF-IDF and the bag-of-words (BOW) model. The Naïve Bayes classifier with a tossing graph assigns the developer [28].

In addition, Tamrawi et al. used titles and descriptions from bug reports and extracted critical terms to use as features. The bug is triaged using fuzzy set features for each word [29]. Further, Anvik and Murphy extracted features using the normalized TF-IDF from the titles and descriptions. They used different machine learning algorithms, such as the Naïve Bayes, expectation-maximization, SVM, decision tree (C4.5), k-nearest neighbor (KNN), and conjunctive rules. Moreover, they generalized the recommendation of components and other appropriate developers rather than a single bug fixer [30].

Xuan et al. used developer prioritization with TF-IDF for feature extraction from titles and descriptions. They triaged the bug using Naïve Bayes and SVMs [31]. Moreover, Banitaan and Alenezi proposed a bug triage approach that uses bug report metadata. They aimed to improve the prediction accuracy of triaging using discriminatory terms from bug reports. They used the TF-IDF and chi-squared methods for feature extraction. The Naïve Bayes classifier performs the assignment task [32].

Similarly, Alenezi et al. proposed an automatic approach to assign developers with relevant experience to new bug reports. The authors used a five-term selection method to choose discriminating terms, including the chi-square, log odds ratio, term frequency relevance frequency, mutual information, and distinguishing feature selector. The predictive model built using the Naïve Bayes classifier selects bug fixers for new bug reports. The chi-squared term selection method outperformed other selection methods [33].

Additionally, Xuan et al. used a data reduction technique to select instances and features, which helps achieve high accuracy. They used the Naïve Bayes classifier to select features and assign bugs to developers [34]. Jonsson et al. [35] used TF-IDF to extract features from titles and descriptions. They created a stacked generalizer classifier by creating an ensemble of the Bayes net, Naïve Bayes, SVM, KNN, and decision tree classifiers. The ensemble technique improved their results.

Furthermore, Florea et al. [36] proposed the SVM method with TF-IDF and chi-squared for the bug recommender system. They tested their model on three different datasets. They preserved nouns from the text attributes (summaries and descriptions) and used TF-IDF for vectorization. Alenezi et al. [37] used categorical features and metadata instead of only textual data in mining the bug tracking system. They used the gain ratio to determine the essential features, providing a normalized measure of each feature's contribution to the classification. They selected operating systems and determined the priority from the metadata because of the high gain ratio. Their results revealed poor performance when only categorical features were used. In contrast, the results of combining textual data and categorical features were promising.

In addition, Sarkar et al. proposed a bug triage system with high-confidence prediction at Ericsson. They replicated the existing models and used a high-confidence prediction level. The authors also used alarms and crash dumps with textual and categorical attributes. The normalized TF-IDF and line-IDF extracted features from the textual data and alarm and crash dumps, respectively. They used the logistic regression, SVM, KNN, and Naïve Bayes classifier. Logistic regression demonstrated good results compared to the others with a 90% confidence interval [38].

Zhao et al. [39] combined the vector space model and topic model to vectorize the textual data. The authors computed TF-IDF for the vector space model and used LDA for topic modeling. Furthermore, the SVM and neural network were used for the classification task. Their results revealed that SVM performed better than the neural network.

Software defect prediction is a similar problem to bug triage. Khurma et al. [40] 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. For classification, they used IsBMFO for feature selection and three different classifiers, SVM, KNN, and Naïve Bayes. The SVM with IsBMFO performed best among KNN and Naïve Bayes.

2.4.2. Deep Learning-Based Bug Triage Systems. Recently, bug triage has been addressed using NLP and deep learning methods. Many deep learning methods employing NLP techniques for word-embedding or word representation have been proposed for bug triage problems in the last few years. For example, S. Lee et al. [2] proposed a CNN-based model that uses the word2vec model for word representation. This model uses summaries and descriptions from bug reports. In addition, they validated this technique on two open-source projects and an industrial project. It was the first study that used CNN for bug triage. Before this, the CNN technique was used for different software engineering problems, such as bug detection, severity classification, and code smells.

In addition, Choquette-Choo et al. proposed a multilabel and dual-output deep neural network for a bug triage system. The authors used latent semantic analysis for latent space representation, proposing a two-output deep neural network architecture. This architecture first predicts team classes and then predicts the developer from the predicted team. Their technique uses a heuristic approach that learns from the bug fixer and contribution level for each developer [45].

Mani et al. [3] proposed a deep bidirectional RNN with an attention mechanism (DBRNN-A) that learns semantic and synthetic features from summaries and descriptions. The authors used a representation based on the DBRNN-A for classifier training. They used word2vec to vectorize textual data and validated their technique on a significantly large dataset. Additionally, the authors publicized the data to make benchmark datasets for further research.

Furthermore, S. Guo et al. proposed an activity-based bug triage technique that uses the CNN model. According to the bug report creation time, the data were sorted, and the last 10% of the data were used for testing. Their technique exhibited promising results on large datasets [4].

Moreover, Zaidi et al. [5] proposed a CNN-based bug fixer recommendation system. The authors used both small and large datasets to validate their technique, using word2vec, GloVe, and ELMo. The CNN employed multiple convolutional kernels to extract diversified features, and their technique demonstrated state-of-the-art performance. Recently, a heterogeneous graph-based bug triage method [44] that uses a graph CNN was designed to create heterogeneous graphs from triage history. This method is quicker than the CNN and RNN methods and revealed results comparable to the existing methods.

Aung et al. [46] proposed a multitriage model that assigns developer and issue types simultaneously. They used two different deep learning models for feature extraction. The text encoder module was based on the CNN model, and an abstract syntax tree encoder module was based on biLSTM. Then, they concatenated features of both encoders and trained the two different classifiers for developer assignment task and bug issue type task. Their model showed good accuracy and performed both tasks simultaneously. However, the model required more training time than other methods due to training two different encoders and models.

Recently, Zaidi et al. proposed a bug triage system using graph neural network (GCN) and heterogeneous graph. They created a heterogeneous graph that has word-word and word-bug document edges. TF-IDF was used for weighting the word-bug document edges. Different similarity metrics were used for weighting word-word edges. Then, a simple two-layer GCN was used to train the model and model recommended list of 10 developers for the unseen reported bug [47].

3. Motivation

Bug triage is a crucial problem in software engineering. A significant number of bugs are reported daily, which is very difficult to triage by a manual triage manager. Many triage systems, which are discussed in Section 2, have been proposed in the last decades to overcome these issues. Some researchers used source code files and fixer information with fixer activities to detect and triage bugs. Some of them used the social network analysis technique, employing bug report descriptions and developer comments to build the social relations for triaging bugs. Although these methods were good choices at the time, advancement is still necessary for effective bug triage.

Because mining software repositories is an arduous task, more than one developer is involved in a source code file, and a repository contains many source code files. Therefore, identifying an appropriate developer for a bug is very difficult. Moreover, mining repository techniques are not scalable because every software project has its own file management structure and requires modification for other projects. The social network analysis-based methods require additional costs to maintain the developer network. In addition, these methods are computationally intensive.

This field has evolved since the use of topic modeling techniques to extract topics from the summary and description of the bug reports for bug triage. Bug reports were assigned to developers according to their expertise identified using their comments and relation to the bug reports. When a new bug was reported, they assigned developers with high correlation scores. However, these methods considered small datasets for testing and missed some crucial features.

Machine learning is a powerful tool widely used in various fields, such as networks, image and signal processing, text classification, language translation, and sentiment analysis. Moreover, machine learning techniques have been applied to various software engineering problems, such as defect prediction, bug localization, code smell detection, bug prioritization, bug duplicate detection, and bug triage.

Since the use of machine learning, the bug triage problem has been solved using text classification techniques. Most researchers have used the summary and description to extract information using various tools (e.g., TF-IDF, BOW, one-hot encoding, and vectorizers). The fixer information was used as the class attribute. Many approaches have been proposed using ensemble techniques and classifiers, such as decision trees, SVM, Naïve Bayes, and random

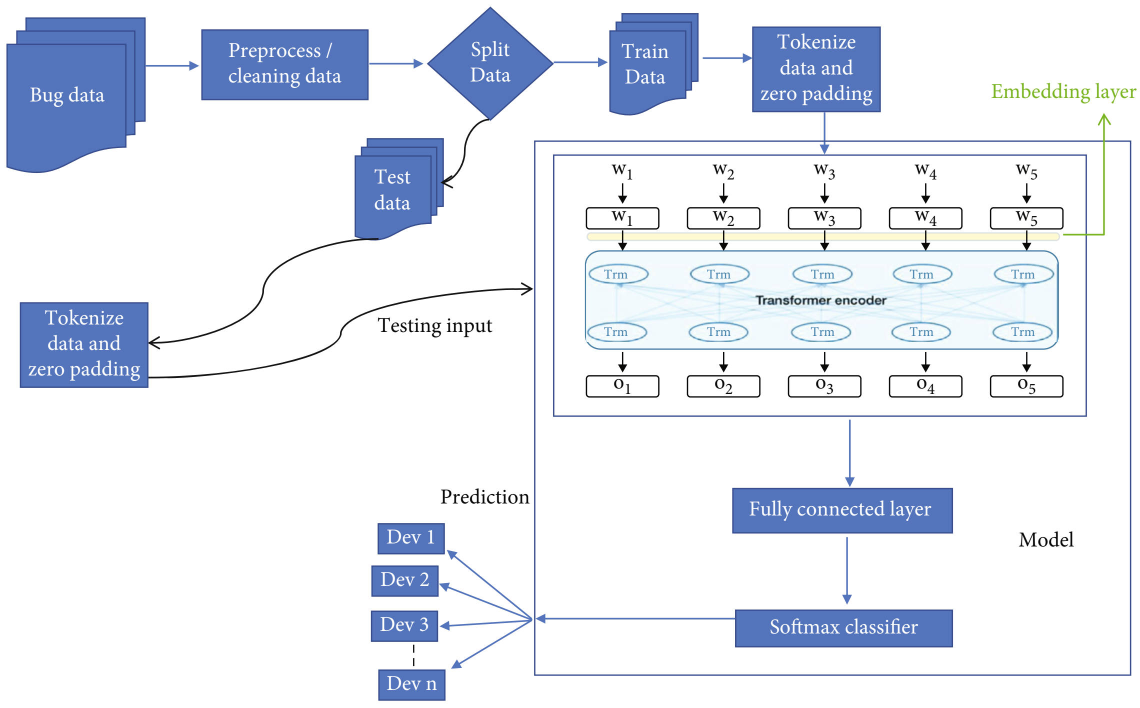


Figure 2: Proposed base model.

forest. These triage methods perform well for small datasets, with a limited number of developer classes. In reality, open-source projects have many developers. However, the performance of these methods decreases with the increase in the size of the dataset and developer classes.

Furthermore, deep learning techniques have advanced the research using CNN and RNN methods. These methods use relatively massive datasets and report the top-k accuracy. Deep learning methods use word representation and embedding techniques to convert text attributes into vectors. As mentioned in Section 2, S. Lee et al. [2] first applied CNN with word2vec for bug triage. Then, Mani et al. [3] proposed an RNN-based representation for bug reports. Later, S. Guo et al. [4] also used the CNN with word2vec and focused on developers' engagement with triage bugs. Zaidi et al. [5] proposed a CNN-based model with three embedding techniques, word2vec, GloVe, and ELMo. These techniques used massive datasets and displayed notable top-k accuracy compared to conventional machine learning methods. However, top-1 accuracy is still very far from satisfactory.

Large-scale open projects allow new developers to join the system/company at any time. In addition, companies can hire new developers at any time and appoint them to existing ongoing projects. Therefore, adding new developers to an existing trained model is challenging, and no solution has been proposed yet. The existing deep learning-based triage methods have performed well for bug triage. However, these methods fail when a new developer or list of developers must be added to the existing system.

We are very impressed by the continuous learning in the literature. For example, Käding et al. [7] applied the concept of fine-tuning to data streams or when the new classes are added continuously. They validated their method on Stanford40Actions and the MS-COCO datasets. They achieved continuous learning through continuous fine-tuning, and the results indicate that continuous fine-tuning is superior to one-step fine-tuning.

Inspired by their idea, we propose a continuous learning-based bug triage method that learns new developers as continuous data. The triage system is trained on the available data first. Then, we add new developers to the pretrained model when the new developer data are available.

4. Methodology

Bug data have a considerable number of bug reports. Each bug report is treated as a bug document. A bug document has information about the bug, including textual and categorical attributes. The proposed bug triage method uses summaries and descriptions from the bug document as input data and uses owner or assignee information as the developer, fixer, or class attribute. The proposed method has two main modules: the base and fine-tuned models. The base model trains on the training data and can provide a ranked list of 10 developers for a new bug report. The fine-tuned model was aimed at learning features for new developers who are not part of the base model. The fine-tuned model trains on the training set of new developer data and can predict the ranked list of appropriate developers for new bug reports for developers using the base and fine-tuned models.

4.1. Base Model. The base model has four main modules: preprocessing, the tokenizer, the BERT embedding model, and the fully connected layer and classifier. Figure 2 illustrates the base model of the proposed bug triage method.

4.1.1. Preprocessing. The textual data are preprocessed after extracting summaries and descriptions from the bug documents. Descriptions contain extra information, including white space, stack trace, URLs, special characters, hexadecimal codes, punctuation marks, code snippets, and directory paths. The extra information and stop words are filtered from the summaries and descriptions in the preprocessing phase.

4.1.2. Tokenizer. The tokenizer prepares the inputs for the model. The "BERTbase-uncased" tokenizer from transformers is used to tokenize the textual data. The BERT tokenizer splits the text strings into subwords and token strings, converts the token strings into IDs, and encodes or decodes the strings into integers. The tokenizer can truncate and zero-pad the textual data. Furthermore, this method manages special tokens, such as the start, end, [SEP], and [CLS] tokens required for the BERT model.

4.1.3. BERT Model. The BERT model is a new language representation model introduced by the Google AI language. Devlin et al. [48] designed the BERT model to pretrain deep bidirectional representations from unlabeled data by jointly conditioning the right and left contexts in all layers. The BERT model can be pretrained and fine-tuned, and the model is trained on unlabeled textual data during the pretraining step. While the BERT model is initialized with pretrained parameters during fine-tuning, the parameters are fine-tuned using the labeled data from the downstream task, such as question answering, text classification, and next sentence prediction.

Since the advancement of NLP and deep learning methods, bug triage problems have been solved using text classification methods. The pretrained BERT model was chosen to learn effective word representation. The BERT base model has positional embedding, 12 transformer layers, 768 hidden states, and 12 attention heads. The pretrained BERT model was trained using 800 million words from BookCorpus and 2500 million words from the English Wikipedia.

For the bug triage tasks, we applied the sequence classification technique. The parameters of the BERT base model were fine-tuned end to end using the tokenized input bug data.

4.1.4. Fully Connected Layer and Classifier Layer. The fully connected layer is fully connected to the output of the BERT base model and has 768 neurons, which are further connected to the classification layer. The classification layer output is equivalent to the total number of developers ($\{\text{dev}_1, \text{dev}_2, \cdots, \text{dev}_n\}$). The cross-entropy loss function was chosen as the cost function because it is beneficial for unbalanced training data and higher dimension inputs. Cross-entropy loss combines LogSoftmax and the negative loglikelihood loss into a single class. The formulation can be simplified as follows:

$$\text{loss}\,(X, \text{class}) = -\log\left(\frac{\exp\,(x[k])}{\sum_j \exp(x[j])}\right), \quad (1)$$

where k is the selected class and j is the jth class.

The Xavier initializer was used to initialize the weights of the fully connected and classification layers. In addition, the Xavier initializer maintains activation variance and backpropagated gradients at a controlled level using a uniform distribution for all network layers.

Overfitting, which customizes the neural network weights for training data and exposes worsened accuracy on testing data, is a significant challenge in neural networks. Dropout counters the overfitting issue by randomly removing neurons during the training process with some probability to limit coadaptation.

4.2. Fine-Tuned Model. Transfer learning is a machine learning approach in which a trained model is reused as the starting point. As mentioned, adding a new developer is an open challenge in the bug triage problem, and it is difficult to add a new developer to an existing model. However, one possible method to solve this problem is fine-tuning, which is a method of transfer learning [49]. The currently trained model on existing developers can be fine-tuned using the data on new developers. Before the tokenization phase, we used the exemplar and replaced the classifier layer with incremental classes to implement this model. The methodology is presented in Figure 3.

4.2.1. Exemplar. The exemplar module forms the training set for the fine-tuned model with data on new and existing developers. This technique has been used for image data. Exemplar management is based on two main objectives: the initial exemplar set should closely resemble the class mean vector, and exemplars should be removable without breaching this property during the algorithm execution [50]. It is challenging to determine the mean vector to choose the exemplar set for textual data because each developer has different skills and has triaged different types of bugs in the past. Previous data are also needed to fine-tune the model because the extended classifier layer replaces the final layer in the base model. Fine-tuning is a quick process that takes fewer iterations to train the model, saving time.

Using all training data on existing developers with new developers for each iteration to fine-tune a model is a bad practice. Moreover, using all data from the previous datasets prevents learning new developers due to highly imbalanced data between the existing and new developer classes. This research was aim at including newly added developers in the existing trained model within less time and fewer iterations.

Therefore, we chose a random selection strategy that enhances data on new developers with randomly selected data from existing or previous developers to make the exemplar set. In each training iteration, the exemplar module selects 40% of the data on existing developers and all data on new developers. The model observes almost all data in chunks in different iterations, allowing the model to obtain good results.

4.2.2. Extended Classification Layer. Like the base model, the exemplar set is tokenized and zero-padded. The trained base model is loaded with trained weights and parameters. Neurons equivalent to the sum of the base and new developer

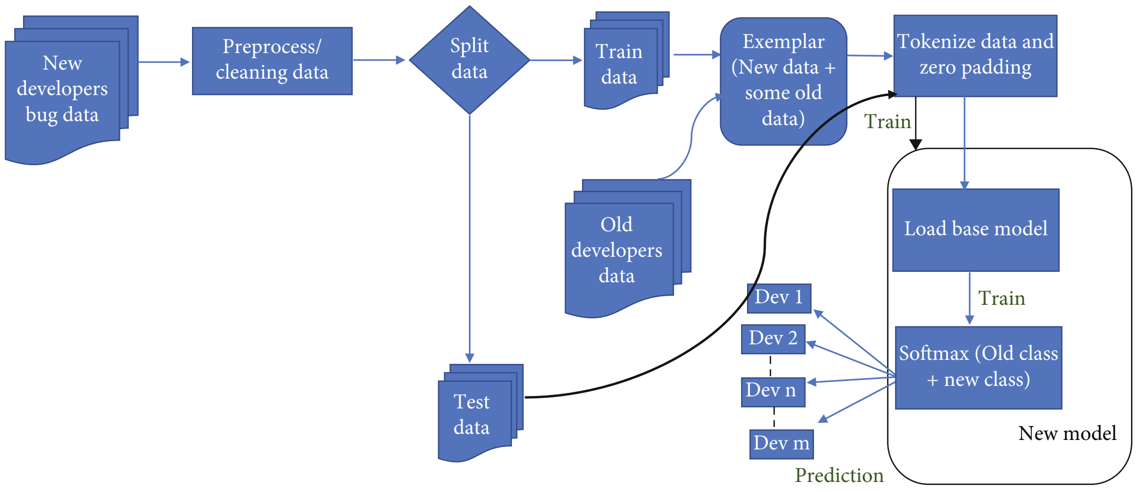


FIGURE 3: Proposed fine-tune model to add newly added developer to the existing bug triage model.

classes $(\{\text{dev}_1, \text{dev}_2, \cdots, \text{dev}_n, \text{dev}_{n+1}, \cdots, \text{dev}_m\})$ replace the last fully connected classification layer. The Xavier initializer was used to assign the initial weights, the same as the base model, and dropout was used to overcome overfitting.

4.3. Training. The AdamW optimizer was used to train the models, computing the updated step of the Adam optimizer and decay variables. Regularization in AdamW is different from L2 regularization, as AdamW regularizes variables with large gradients that have better training loss. The BERT model uses a scheduler for learning rate management. A linear scheduler with warmup steps creates a schedule with a learning rate to optimize the learning process. In the warmup steps, the learning rate increases linearly from zero to the initial learning rate. Afterward, the learning rate decreases linearly from the initial learning rate, set at 1.0005, to zero. The dropout and weight decay values are set to 0.5 and 0.001, respectively. The model learns with a batch size of 32 and 10 epochs.

5. Evaluation and Results

This section evaluates the proposed bug triage system and addresses the following research questions:

(i) Is the BERT-based bug triage method superior to other context-insensitive and context-sensitive ELMo-based bug triage methods?

(ii) Does the BERT-based base model have better top-k accuracy on bug data?

(iii) Is fine-tuning a pretrained model better than training the model from scratch?

(iv) Can the continuous fine-tuning approach solve the newly added developer problem?

(v) What problem is continuous fine-tuning facing, and what is its effect on the accuracy results?

(vi) Is the proposed model efficient?

5.1. Data Collection. Large-scale open-source project data were used to evaluate the performance of the proposed bug triage system. S. Lee et al. [2] and Zaidi et al. [5] used two datasets, the Eclipse Platform and Mozilla Firefox, to evaluate the performance of their triage systems. Mani et al. [3] and Zaidi et al. [5] used another Mozilla Firefox dataset, including variants with a minimum of 0, 5, 10, and 20 bug reports per developer. We used the same datasets to evaluate and validate the proposed bug triage system. Two more datasets (Mozilla and NetBeans), which were also used by S. Guo et al. [4] and Zaidi et al. [5], were used to validate this work. Datasets have many bug reports and a significant number of developers, and these datasets are publicly available on GitHub (https://github.com/farhan-93/bugtriage).

5.2. Evaluation Measure. Top-k accuracy was used to evaluate the proposed bug triage system. We calculated the top-1 to top-10 accuracy values for a valid comparison with state-of-the-art triage methods. Equation (2) was used to calculate the top-k accuracy.

$$\text{Top-k accuracy} = \frac{\sum_{i=1}^{N} I(\text{rec}_i@k, \text{dev}_i)}{|N|}. \tag{2}$$

Function I returns 1 if the recommendation list has the correct developer for the ith bug report. The reported results are an average of five trials. For the performance evaluation, the same validation methods were used as those in the comparative papers. Time-split validation was used to compare the method with deep triage [3], where 10% of the total data were separated from the end for validation, and the remaining 90% of the data were used for training. Similarly, 20% of the data from the end were used for validation, and the remaining 80% were used to train for the DA-CNN [4].

5.3. Addressing the Research Questions

5.3.1. RQ 1: Is the BERT-Based Bug Triage Method Superior to Other Context-Insensitive and Context-Sensitive ELMo-Based Bug Triage Methods? Many embedding techniques, including context-insensitive (word2vec and GloVe) and context-sensitive (ELMo) methods, have been used for bug triage systems. The pretrained vectors of word2vec and GloVe are also available and can be trained on new datasets. Moreover, the ELMo-pretrained model can be used for embedding and fine-tuning using current data during embedding. Zaidi et al. [5] and S. Lee et al. [2] used word2-vec for word representation. In addition, S. Lee et al. trained the word2vec model using bug report data, while Zaidi et al. used the pretrained word2vec vectors. Furthermore, Zaidi et al. used GloVe-pretrained vectors and a pretrained ELMo embedding model. Their results demonstrated the superiority of ELMo compared to word2vec and GloVe.

Similarly, the pretrained BERT model was used for better word representation. The BERT is deeper model than ELMo, which yields good performance. Further, BERT uses a transformer-based architecture, and ELMo uses a bidirectional language model. Moreover, ELMo extracts context-aware features in both directions and concatenates them to make contextual representations. This concatenation limits the ELMo technique to take advantage of both left and right context information simultaneously. In contrast, BERT uses the masked language model in which it replaces words in a sentence randomly with a small probability (mask). Then, the transformers generate a prediction for masked words based on their left and right surrounding unmasked words [48].

The BERT method has demonstrated state-of-the-art performance in many NLP-based tasks. Moreover, BERT has also performed better in bug triage than the existing bug triage systems. Tables 2 and 3 list the top-1 to top-10 accuracy results of the BERT-based triage system for different datasets. The experimental results reveal a noticeable increase in the top-1 to top-10 accuracy values for all datasets compared to the state-of-the-art method, evidencing BERT-based triage's superiority over the word2vec, GloVe, and ELMo methods.

Some statistical tests were conducted to check the significance of the results. The proposed method was validated on eight distinct open-source datasets. First, the Kruskal–Wallis test was conducted for the top-1, top-5, and top-10 accuracy of the proposed model, including the ELMo-CNN, GloVe-CNN, and word2vec-CNN. The Kruskal–Wallis test has a p value of less than 0.05 for top-1, top-5, and top-10 accuracy, exhibiting the significance of the accuracy results. The Nemenyi post hoc test was conducted to identify the significance between various bug triage methods. The post hoc test demonstrates that the proposed method has significantly different accuracy values than GloVe and word2vec for top-1 and top-5 accuracy. However, the proposed method has significantly different results than the word2vec-CNN for top-10 accuracy. Moreover, no significant difference exists between the proposed method and the ELMo-CNN.

We also performed the analysis of variance (ANOVA) single-factor test to double-check the significance of the accuracy values. The same observations were found in the Kruskal–Wallis test, with p values of less than 0.05 for top-1, top-5, and top-10 accuracy. The Tukey honest significance difference (HSD) test is the post hoc test performed after ANOVA to identify the significant difference between two groups statistically. The Tukey HSD demonstrates that the proposed method has significantly different values than the word2vec-CNN and GloVe-CNN for top-1 and top-5 accuracy. The proposed method has significantly different values than word2vec-CNN for top-10 accuracy. However, no significant difference exists between the ELMo-CNN and BERT-CNN for top-1, top-5, and top-10 accuracy.

Figure 4 is the Demšar diagram demonstrating the comparison of all bug triage methods with different word-embedding techniques on all datasets. The critical distance value is 1.382 with $\alpha = 0.05$. The connection between methods indicates the insignificant difference in accuracy results. The Demšar diagram also partially supports the research question.

Thus, the statistical tests partially support our research questions. Overall, the proposed method demonstrates a considerable difference in top-k accuracy compared to various bug triage methods using diverse word-embedding techniques. The proposed method has significantly different accuracy values than the word2vec-CNN and GloVe-CNN, but no significant difference exists between the proposed and ELMo-CNN methods.

5.3.2. RQ 2: Does the BERT-Based Base Model Have better Top-K Accuracy on Bug Data? The proposed base model was validated on various datasets used in state-of-the-art methods. The experimental results indicate a significant increase in top-1 accuracy by 4% and 10% for the Eclipse Platform and Firefox small datasets, respectively. The Firefox dataset with a threshold of zero presented similar results for top-1 accuracy. The difference in accuracy decreased as the threshold increased. Top-10 accuracy exhibited a negligible difference from the ELMo-CNN for thresholds of 10 and 20. However, the base model displayed a notable difference in the top-k accuracy for thresholds of zero and five, implying that the BERT-based bug triage is a good option when the number of samples for each developer is very low. Overall, the BERT-based bug triage system demonstrated promising results for the top-1 to top-10 accuracy values. Tables 2 and 3 present detailed results for the top-k accuracy metric for all datasets. In summary, the BERT-based triage model is efficient at gaining good top-k accuracy.

The Friedman test was performed to test the significance of the experimental results at a 5% α value (significance level). The research question is addressed as a hypothesis that can be confirmed or rejected. As mentioned, the reported results are an average of five trials. The statistical test had a p value of less than 0.05 for all datasets. However, an α value of > 0.05 was recorded for the top-9 and top-10 accuracy values of the Firefox dataset with 20 thresholds. Overall, the statistical analysis has a p value of less than 0.05 most of the time. Therefore, the statistical test confirms the stated research question—the results of the proposed method have significant top-k accuracy. However, the significance is lost for top-9 and top-10 accuracy when the number of samples per class increases.

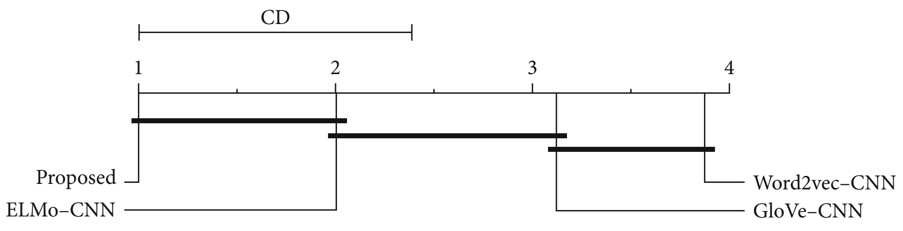
Table 2: Average top-1 to top-10 Accuracy obtained on Platform, Firefox small [2], and Firefox-thresholded [3] datasets.

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 [2]	S. Lee et al. [2]	36.1	45.8	50.5	53.7	56.7	—	—	—	—	—
	Word2Vec-CNN [5]	38.036	48.870	55.160	58.220	62.492	65.604	67.632	69.324	70.832	71.714
	GloVe-CNN [5]	40.132	51.004	56.234	60.318	63.770	65.964	68.252	70.058	71.554	72.930
	ELMo-CNN [5]	43.622	51.830	56.366	60.704	63.822	66.528	69.068	70.964	72.794	74.264
	Our	47.88	57.62	63.34	65.89	68.22	69.06	69.91	71.39	73.72	74.36
Firefox [2]	S. Lee et al. [2]	27.1	36.7	42.8	47.1	50.5	—	—	—	—	—
	Word2Vec-CNN [5]	27.396	37.894	44.256	48.346	51.604	54.422	57.006	58.858	60.430	61.750
	GloVe-CNN [5]	28.614	38.660	45.062	50.094	53.024	56.374	58.496	60.522	62.410	64.224
	ELMo-CNN [5]	30.418	41.104	47.81	52.508	55.738	58.362	60.404	62.314	63.598	64.696
	Our	40.22	49.29	53.56	57.83	60.37	62.25	53.37	64.94	66.07	67.34
Firefox [3] 0 threshold	Deep Triage [3]	—	—	—	—	—	—	—	—	—	38.1
	Word2Vec-CNN [5]	15.2	23.52	28.77	32.07	35.15	37.34	39.56	41.37	42.89	43.63
	GloVe-CNN [5]	16.84	25.22	31.78	33.78	35.76	39.19	42.19	43.96	43.96	45.32
	ELMo-CNN [5]	20.86	29.04	34.332	38.03	41.18	43.57	45.72	47.68	49.28	50.73
	Our	28.96	37.86	43.44	46.80	49.48	51.74	53.91	55.87	57.24	58.47
Firefox [3] 5 threshold	Deep Triage [3]	—	—	—	—	—	—	—	—	—	44.5
	Word2Vec-CNN [5]	17.43	25.01	29.58	33.28	37.50	40.87	42.98	44.32	45.76	45.91
	GloVe-CNN [5]	18.47	26.80	32.62	36.94	38.29	41.34	43.41	45.31	46.01	47.64
	ELMo-CNN [5]	26.51	37.02	43.29	48.17	51.77	54.20	56.05	58.12	59.78	61.41
	Our	30.19	40.67	46.99	51.67	54.77	56.94	59.19	60.87	62.46	63.79
Firefox [3] 10 threshold	Deep triage [3]	—	—	—	—	—	—	—	—	—	51.4
	Word2Vec-CNN [5]	19.19	27.55	33.58	39.05	41.41	43.85	45.13	46.02	47.90	51.06
	GloVe-CNN [5]	19.25	29.69	34.65	40.23	42.64	44.68	46.68	48.35	49.44	51.67
	ELMo-CNN [5]	31.01	42.85	49.83	54.50	57.96	60.77	63.78	64.92	66.62	67.90
	Our	34.87	44.55	51.15	55.49	58.44	61.74	63.03	65.34	67.45	68.02
Firefox [3] 20 threshold	Deep triage [3]	—	—	—	—	—	—	—	—	—	55.8
	Word2Vec-CNN [5]	24.34	34.40	39.73	44.62	48.19	58.68	52.93	54.78	56.74	58.94
	GloVe-CNN [5]	23.16	33.43	39.63	44.72	48.58	52.10	54.34	56.74	58.50	59.92
	ELMo-CNN [5]	35.89	47.03	53.93	58.94	62.14	65.04	67.49	69.39	70.89	72.65
	Our	38.42	48.44	53.88	59.32	62.74	65.49	67.75	69.46	71.21	72.72

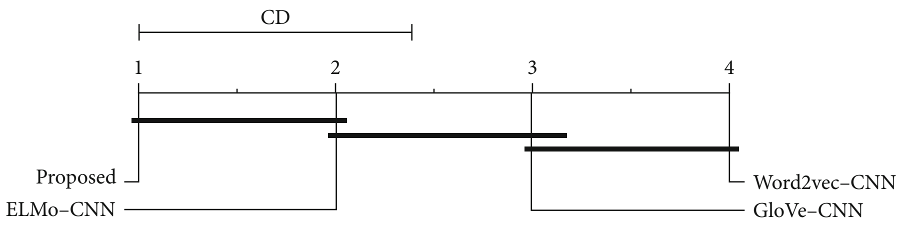
5.3.3. RQ 3: Is Fine-Tuning a Pretrained Model Better than Training the Model from Scratch? Training a deep learning model from scratch is very costly because it requires more time. Fine-tuning is more efficient than allowing a model to learn from scratch if the model was aimed at learning the same domain task and sufficient new data are available

Table 3: The average top-1 to top-10 accuracy obtained on S. Guo et al.'s dataset [4]. The dataset is split and 20% of data is used for testing. The best performing values are shown in bold.

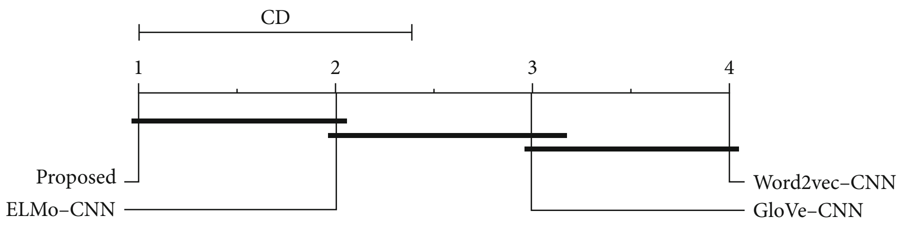
Dataset	Techniques	Top-1 Acc.	Top-2 Acc.	Top-3 Acc.	Top-4 Acc.	Top-5 Acc.	Top-6 Acc.	Top-7 Acc.	Top-8 Acc.	Top-9 Acc.	Top-10 Acc.
Mozilla	CNN-DA [4]	12.44	19.09	22.54	24.91	26.93	28.26	30.17	31.71	33.00	34.46
	One-Hot+CNN [4]	8.16	15.69	19.67	20.768	23.57	25.19	27.18	28.35	30.11	32.86
	BOW+NB [4]	8.15	11.43	13.69	15.78	17.49	18.68	19.76	21.29	22.65	23.69
	Word2Vec-CNN [5]	12.74	17.83	21.96	25.70	27.41	29.31	31.24	32.92	34.53	35.76
	GloVe-CNN [5]	16.09	24.12	29.02	31.82	34.11	36.31	38.47	40.08	41.27	42.63
	ELMo-CNN [5]	16.73	25.18	30.15	33.98	36.47	38.95	40.89	42.37	44.05	45.40
	Our	**27.61**	**37.12**	**41.08**	**44.80**	**47.37**	**50.36**	**53.06**	**55.33**	**56.19**	**57.37**
NetBeans	CNN-DA [4]	35.54	48.02	54.36	58.93	62.66	65.39	68.11	71.11	72.73	74.89
	One-Hot+CNN [4]	30.49	44.62	48.76	53.91	56.18	60.29	63.79	66.61	67.93	69.15
	BOW+NB [4]	21.57	30.41	.36.49	40.39	43.59	45.72	46.25	50.18	51.95	54.00
	Word2Vec-CNN [5]	35.73	48.40	53.99	59.10	61.34	62.71	64.58	66.18	67.80	68.92
	GloVe-CNN [5]	38.51	49.27	55.17	59.11	62.01	64.81	67.99	70.87	71.26	72.84
	ELMo-CNN [5]	40.17	53.13	59.05	63.02	66.07	68.25	70.06	71.54	72.92	74.23
	Our	**48.85**	**56.07**	**61.61**	**64.61**	**67.99**	**70.73**	**72.10**	**75.89**	**77.97**	**79.81**



(a) Top-1



(b) Top-5



(c) Top-10

Figure 4: The Demšar diagram compares proposed method with Word2vec-CNN, GloVe-CNN, and ELMo-CNN for top-1, top-5, and top-10 accuracy. The CD is the critical distance value, which is 1.382 with $\alpha = 0.05$. Subfigures (a), (b), and (c) show the average rank of triage methods for top-1, top-5, and top-10 accuracy, respectively.

for training. In our case, sufficient bug report data were available, and the pretrained model could be fine-tuned when the new data were available.

For evidence, we trained two separate models: the model from scratch and the fine-tuned model. The BERT model was trained using bug data to train the model from scratch. In contrast, the pretrained BERT model was used for the fine-tuned model, trained on 800 million words from Book Corpus and 2,500 million words from the English Wikipedia. Bug data have many bug reports and a considerable amount of text. However, after removing the stop words and cleaning the data, the length of many sentences decreased. Training a BERT model from scratch requires much more data. However, the fine-tuned model was efficiently trained using the pretrained model and converged into fewer iterations.

Furthermore, the experimental results support the research question. The top-1 accuracy of the scratch model was 19%, whereas the top-1 accuracy of the fine-tuned model was 28.96%. The scratch model took 20 epochs for training, and the fine-tuned model demonstrated better results in just 10 epochs. To validate this observation, we observed the training times for both the from-scratch and fine-tuned models. We executed the code on an RTX 2080 graphics processing unit (GPU) system. Each epoch of the scratch model took an average of 2.5 minutes for training using the training set from the Firefox [4] dataset. However, each epoch of the fine-tuned model took an average of 2 minutes for training. Therefore, the scratch model took an average of 48 minutes for 20 epochs, and the fine-tuned model took an average of 20 minutes for 10 epochs. Thus, using the pretrained model is a better option for training a model from scratch.

5.3.4. RQ 4: Can the Continuous Fine-Tuning Approach Solve the Newly Added Developer Problem?

Adding a set of new developers to an existing model is an ongoing problem. The existing methods cannot add new developers and require training from scratch for all developers (new developers and previous developers). As mentioned in Section 3, Käding et al. applied the continuous fine-tuning concept to allow the method to add new classes when they are available. Similarly, new developers can be added via continuous fine-tuning whenever data on new developers are available. In addition, the dataset was prepared for this task. The Firefox 20 threshold dataset has 169 developers separated randomly into sets of 100 developers, six sets of 10 developers, and nine developers. The 100-class dataset was used to train the base model, and the remaining sets were used for continuous fine-tuning and adding new developers to the existing model.

Realistically, new developers have no triage history; thus, no data would be available for the training model. Therefore, we assumed that 10 developers were new, and a manual triage manager created their histories according to their skills. Then, the manual triage history could fine-tune the model to add new developers to the existing model. After adding new developers, the experimental results revealed good top-1 to top-10 accuracy results for the continuous fine-tuning method. The base model for all 169 classes exhibited 38.42% top-1 accuracy, whereas the base model demonstrated 42.77% accuracy for the 100-class data. By incrementing the remaining developer sets, the fine-tuning model had 39.20% top-1 accuracy.

Creating a manual triage history of at least 10 bug reports is problematic because it is very challenging for a manual triager to determine at least 10 reports from another developer's history to create a triage history for a new developer. Therefore, the Eclipse Platform dataset was divided into the base and incremental sets to make it a more realistic scenario.

The Platform dataset has 225 developers divided into 200 and 25 developers for the base and fine-tuned incremental models, respectively. In addition, the 25 developers have three bug reports, at most, in their history. From these bug reports, one bug report was chosen for testing. The experimental results of the base model indicate 47.88% top-1 accuracy for all 225 developer classes, whereas the base model exhibited 51.20% top-1 accuracy for 200 classes. However, the fine-tuned model had 49.17% top-1 accuracy for the base and incremental testing sets (200 plus 25 classes). The top-1 accuracy of the fine-tuned model was better than the base model for all 225 classes; however, it was not significant.

The detailed results of the Firefox 20 threshold and Eclipse Platform datasets are presented in Table 4. The table lists the top-1 to top-10 accuracy values for each iteration. In the technique column, the base demonstrates the scratch model for the base classes, and Inc_k indicates the fine-tuning method for each incremental iteration, where the column "classes" presents the number of base and increment developer classes in each iteration.

Comparing the first and second scenarios, developers from the first scenario have a minimum of 20 bug reports in their histories. Some have more than 20 bug reports, as many as 100 to 150 bug reports. These bug reports are manually triaged, and a developer may fix many bug reports of a different nature. Moreover, it is also possible that two or more developers may have fixed bug reports of the same nature in the past. Therefore, it can be a challenge for efficient learning, causing low accuracy. In the second scenario, new developers have a maximum of three bug reports. During the fine-tuning of the pretrained model, new classes exhibit good accuracy results compared to the base method results for all 225 classes. Additionally, different developers may have bug reports of the same nature. Nevertheless, this situation has no significant effect on learning and is not the cause of accuracy loss because the model learns the features of new developers from their data and does not lose knowledge when learning about additional developers. In summary, the continuous fine-tuning method presents promising results for adding new developers to an existing model. However, the accuracy results are not significant; thus, they require improvement.

5.3.5. RQ 5: What Problem Is Continuous Fine-Tuning Facing, and What Is Its Effect on the Accuracy Results?

Continuous learning can be achieved by continuous fine-tuning [7]. In the bug triage problem, continuous fine-tuning demonstrated better results than training a model from scratch.

Table 4: The average top-1 to top-10 accuracy obtained on Firefox 20 threshold data and Platform dataset. “Base” in Technique column shows the first trained model and Inc_k shows the class increment iterations, where column “classes” shows the number of base classes +increment classes.

Dataset	Methods	Classes	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.
Firefox 20 threshold	Base	169	38.42	48.44	53.88	59.32	62.74	65.49	67.75	69.46	71.21	72.72
	Base	100	42.77	55.21	63.02	67.37	71.73	74.87	76.42	78.77	80.02	82.54
	Inc_1	100 + 10	41.61	52.89	62.74	67.26	69.16	71.35	74.01	74.40	78.13	79.58
	Inc_2	110 + 10	40.89	53.45	61.32	66.01	68.00	71.98	74.19	74.69	76.98	78.11
	Inc_3	120 + 10	40.16	52.98	60.74	65.42	67.34	70.13	73.43	74.23	75.45	76.77
	Inc_4	130 + 10	39.86	52.40	60.01	64.99	66.70	69.32	73.00	74.12	75.13	75.96
	Inc_5	140 + 10	39.90	53.01	59.61	64.01	65.40	68.01	72.68	74.01	74.11	75.26
	Inc_6	150 + 10	39.54	52.46	58.74	63.48	64.40	67.77	71.00	72.45	73.89	74.36
	Inc_6	160 + 9	39.20	52.17	57.98	62.78	64.20	67.00	70.21	71.40	72.74	74.10
Platform	Base	225	47.88	57.62	63.34	65.89	68.22	69.06	69.91	71.39	73.72	74.36
	Base	200	51.20	62.34	66.92	68.92	70.20	72.70	73.46	75.01	76.96	78.87
	Inc_1	200 + 25	49.17	60.97	64.55	67.93	69.83	71.94	73.41	74.25	75.10	76.37

As listed in Table 4, the from-scratch model (base model technique) has 38.42% top-1 accuracy on all 169 classes. Furthermore, continuous fine-tuning exhibits 39.20% accuracy for the same dataset. First, the base model was trained on 100 developers’ data to make a pretrained base model for fine-tuning. Then, the remaining developers were added continuously, as in continuous learning.

However, continuous fine-tuning has a catastrophic forgetting problem, common in deep neural networks and sequential transfer learning. This problem happens due to the forgetting of previous knowledge and overfitting of new data.

In our bug triage case, the model learns new developer classes more efficiently and trains on complete base data in chunks in a few iterations. However, some bug reports were misclassified from the base testing data, demonstrating catastrophic forgetting and affecting the overall efficiency of the continuous fine-tuning model. However, our model was also trained on some parts of the base data with new data during fine-tuning. Still, the proposed model lost some knowledge and was overfitted to the new data. As presented in Table 4, the top-k accuracy decreased in each iteration from Inc_1 to Inc_6, indicating the forgetting of knowledge from the previous model. In the future, we aim to develop a more efficient triage model that can address the catastrophic forgetting problem using the knowledge distillation method.

Knowledge distillation transfers learned knowledge from the parent model to the child model without losing knowledge. We can implement this concept using the knowledge distillation loss function. The base model knowledge is transferred to a fine-tuned model to increase the triage system accuracy and efficiency.

5.3.6. RQ 6: Is the Proposed Model Efficient? Deep learning models require considerable memory to store weights, long training times, and computation times. The use of the pretrained language model reduces training time. The BERT model on the bug dataset requires more iterations to converge; however, the proposed pretrained BERT model exhibits good results and converges in fewer iterations. Furthermore, the BERT model requires more memory to restore the pretrained model because it has 11 transformer layers. Nevertheless, it does not require sophisticated GPUs. The experiments were conducted on two systems: (1) a Core i9-9900X CPU with 64 GB RAM and Nvidia RTX 2080 with 12 GB of dedicated memory and (2) a Core i7-8700 CPU with 64 GB RAM and Nvidia GeForce GTX 1080 Ti.

We applied the proposed method to a set of datasets with varying sizes from different open-source projects (i.e., Eclipse, Mozilla, and NetBeans). The Platform [2] is the smallest dataset with 4825 bug reports and 225 developers. The Firefox [2], Mozilla [4], and NetBeans [4] are massive datasets that have 13667 reports with 848 developers, 15502 reports with 1022 developers, and 19149 reports with 265 developers, respectively. The proposed method showed good performance on all the datasets. By increasing dataset size, we can expect that the training time and training steps increase, such as each epoch of Platform dataset [2] took 30 seconds for training and trained in 109 training steps. Firefox dataset [2] took 1 minute and 20 seconds for training each epoch in 308 training steps. In comparison, each epoch took approximately 430 training steps and 1 minute and 50 seconds to train the model on the NetBeans [4] dataset. Further, the model grows gradually when the dataset of new developer classes is available. Table 4 shows the experimental results, indicating that the model gradually grows when adding new developer classes.

However, we did not test the proposed method for industrial projects because the bug data are not publicly available. Nevertheless, we hope that the proposed method can be adapted to industrial projects because their bug reports also include a summary, description, and developer information.

5.4. Comparison with Other Research. Recently, many studies have been proposed for bug triage, demonstrating promising results. The proposed bug triage method outperformed some of these recent methods. A comparison was made with just a few studies whose datasets are publicly available. Comparisons with the most recent studies were not possible because many studies have not published their datasets, and most studies only shared the Bugzilla link and stated the bug data collection period. Reproducing a dataset is challenging and time-consuming because the reproduced dataset may have changes affecting the results and causing unfair comparisons. Moreover, these studies' source codes are also publicly unavailable, so we cannot apply their techniques to our datasets. Thus, we compared the proposed triage method with only the few methods that have published datasets for a fair comparison.

The proposed method outperformed the Deep Triage [3], DA-CNN [4], one-hot CNN [4], BOW with Naïve Bayes [4], GCN-based bug triage [44], word2vec-CNN [5], GloVe-CNN [5], and ELMo-CNN [5] methods. In addition, the proposed method demonstrated notable top-1 to top-10 accuracy results except on the Firefox 20 threshold dataset because it was better than the Firefox 20 threshold by a negligible margin. Tables 2 and 3 present the detailed experimental results and a comparison with the stated bug triage techniques. Overall, the proposed methods demonstrated good top-k accuracy values compared to all techniques. The GCN-based bug triage research is not complete because it is a conference paper, and we still require extended results for a fair comparison. However, the proposed method has better results than the GCN-based bug triage technique. Our method demonstrated 40.22% top-1 accuracy, whereas the GCN-based method exhibited 29.37% accuracy for the Firefox [2] dataset. Comparing the top-10 accuracy, the GCN-based method had 66.50% accuracy, and the BERT-based method was at 67.34%, which is not a substantial difference. Therefore, we did not add the results to the comparison table.

6. Threats to Validity

6.1. Construct Validity. The model performance was estimated using the same protocols as the previous methods to split the dataset for training and testing. Moreover, dataset creation for newly added developers was operated under several assumptions. The proposed and existing bug triage methods require the developers' triage histories. To provide this, we created two datasets: one from Firefox 20 and the other from the Eclipse Platform dataset. The second dataset explains a more realistic scenario in which developers with a maximum of three bug reports are selected. We assumed these bug reports were manually triaged according to their skills and input into the model to learn the new developers. Additionally, the proposed method was validated using published data from previous studies; therefore, we expected no construct validity for this research. Furthermore, we intended to determine a more realistic scenario for new developer dataset construction in future research.

6.2. Internal Validity. As mentioned, the data published by previous researchers were used to validate the proposed model. They collected data using APIs from open-source projects and incorporated all reports in a specific closed and fixed period. They extracted all required information using regular expressions to complete the dataset. We ensured that all bug reports were publicly available for specific projects. We hope that no internal threats to validity exist because we used published data.

6.3. External Validity. The three open-source projects (Mozilla, Eclipse, and NetBeans) were considered in Bugzilla. Bugzilla is compatible with previous studies; therefore, the results may not apply to all other open-source and industrial projects. However, these open-source projects are long-lived systems, are large, and have possible biases. Additionally, the method should apply to all other open-source and industrial projects because each report has a summary and description. Moreover, replicating this study with open-source and industrial various projects could be beneficial.

Another possible limitation of this study is that the proposed research was only compared with a few datasets. It was impossible to compare the findings with those of many other studies because most employed different datasets. The data-gathering duration or time frames were not the same even when two projects were compared. Most researchers only provided limited data, such as the Bugzilla main page/source URL, time interval, resolution, and status, among other aspects. Furthermore, most previous research did not clarify the cleaning methods in-depth, making it challenging to ensure that we were using the same dataset. Therefore, we used only publicly available datasets to validate this method.

7. Conclusion

Bug triage is a crucial problem in software engineering, which requires an appropriate tool for assigning developers/fixers to reported bugs. A significant amount of research has been done by various researchers. Nevertheless, these methods are still lacking good top-k accuracy for large-scale datasets. Furthermore, these methods cannot add new developers to the trained model, which is a considerable challenge in the bug triage problem.

This paper provides a solution for adding new developers to an existing model and achieving higher top-k accuracy than existing triage techniques. Therefore, we proposed a transformer-based bug triage system that recommends a ranked developer set for a bug report. In the system, the context-sensitive word representation technique BERT is used for effective word representation. Moreover, the proposed system can add new developers to the existing system. The proposed method was validated on datasets from well-known open-source projects, such as Eclipse, Mozilla, and NetBeans. We used the top-k accuracy metric as the criterion for comparing performance with state-of-the-art triage methods. The few developers with small triage histories were separated to make a dataset for new developers. We assumed that these developers were new, and a manual triager assigned bug reports to them. The experimental results demonstrated that the BERT-based triage method is better than the state-of-the-art methods discussed in comparison and is a good choice for adding new developers to existing models.

Additionally, the method can add a new developer or set of developers and assign a new bug report to a new developer. The BERT-based model performed better than ELMo, GloVe, and word2vec because BERT is deeper and uses a masked language model. The proposed triage method demonstrates notable top-k accuracy values for all datasets. The top-9 and top-10 accuracy values exhibited a negligible difference in accuracy results when the reports per developer were a minimum of 20.

We performed statistical tests, the Kruskal–Wallis and single-factor ANOVA tests, to support the significance of the experimental results. The significance test supports the reported results and research questions with a 95% confidence interval. The proposed base model has significant top-k accuracy. However, the significance test indicates no significant accuracy results for top-9 and top-10 accuracy, where the number of samples per developer is at least 20. In addition, the Nemenyi and Tukey HSD tests were performed to check the significance of the difference for the proposed and existing triage models with different word representations. The proposed method exhibited a significant difference from the word2vec-CNN and GloVe-CNN. No significant difference exists between the proposed and ELMo-CNN triage methods. Nevertheless, the proposed method has higher top-1 accuracy than the ELMo-CNN technique.

Furthermore, fine-tuning a pretrained language model demonstrated better results than training a model from scratch. Training a model from scratch requires more time, more training iterations, and a significant amount of data. Continuous fine-tuning provides a solution for adding new developers, who can be added when their bug data become available. Then, the method fine-tunes the pretrained model on new data and some parts of the previous data in just a few iterations. However, it has a catastrophic memory problem and requires improvement because the pretrained model loses some knowledge. We do not argue that the proposed approach is best for handling new developers, but it is a solution. Significant research is required to find a better solution to add new developers to the existing model.

In the future, we intend to use the knowledge distillation technique to overcome the catastrophic memory problem and determine a more efficient approach to adding a new developer. Moreover, to acquire a more appropriate triage system, we plan to enhance this work using various word-embedding techniques with continuous fine-tuning and knowledge distillation.

Data Availability

The datasets used for the experiments are available on https://github.com/farhan-93/bugtriage. We used publicly available datasets, and their link can also be found in relative articles.

Conflicts of Interest

The authors declare that there is no conflict of interest regarding the publication of this paper.

Acknowledgments

This research was supported partly by the Chung-Ang University research grant in 2021 and partly by the National Research Foundation of Korea (NRF) grant (NRF-2021R1F1A1059492).

References

[1] 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.

[2] 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 *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering - ESEC/FSE 2017*, pp. 926–931, New York, New York, USA, 2017.

[3] S. Mani, A. Sankaran, and R. Aralikatte, "DeepTriage: exploring the effectiveness of deep learning for bug triaging," in *Proceedings of the ACM India Joint International Conference on Data Science and Management of Data*, pp. 171–179, Kolkata, India, 2019.

[4] S. Guo, X. Zhang, X. Yang et al., "Developer activity motivated bug triaging: via convolutional neural network," *Neural Processing Letters*, vol. 51, no. 3, pp. 2589–2606, 2020.

[5] 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.

[6] S. F. A. Zaidi and C.-G. Lee, "One-class classification-based bug triage system to assign a newly added developer," *IEEE2021 International Conference on Information Networking (ICOIN)*, pp. 738–741, Jeju Island, Korea (South), 2021.

[7] C. Käding, E. Rodner, A. Freytag, and J. Denzler, "Fine-tuning deep neural networks in continuous learning scenarios," in *Asian Conference on Computer Vision*, pp. 588–605, Cham, 2017.

[8] H. Kagdi, M. Gethers, D. Poshyvanyk, and M. Hammad, "Assigning change requests to software developers," *Journal of Software: Evolution and Process*, vol. 24, no. 1, pp. 3–33, 2012.

[9] R. Shokripour, Z. M. Kasirun, S. Zamani, and J. Anvik, "Automatic bug assignment using information extraction methods," in *2012 International conference on advanced computer science applications and technologies (ACSAT)*, pp. 144–149, Kuala Lumpur, Malaysia, 2012.

[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 *2013 10th Working Conference on Mining Software Repositories (MSR)*, pp. 2–11, San Francisco, CA, USA, 2013.

[11] M. Linares-Vásquez, K. Hossen, H. Dang, H. Kagdi, M. Gethers, and D. Poshyvanyk, "Triaging incoming change requests: bug or commit history, or code authorship?," in *2012 28th IEEE International Conference on Software Maintenance (ICSM)*, pp. 451–460, Trento, Italy, 2012.

[12] H. Naguib, N. Narayan, B. Brügge, and D. Helal, "Bug report assignee recommendation using activity profiles," in *2013 10th Working Conference on Mining Software Repositories (MSR)*, pp. 22–30, San Francisco, CA, USA, 2013.

[13] 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 *2014 IEEE 38th Annual Computer Software and Applications Conference*, pp. 97–106, Vasteras, Sweden, 2014.

[14] S. Wang, W. Zhang, and Q. Wang, "Fixercache: unsupervised caching active developers for diverse bug triage," in *Proceedings of the 8th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pp. 1–10, Torino, Italy, 2014.

[15] 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 *2014 21st Asia-Pacific Software Engineering Conference*, pp. 223–230, Jeju, Korea (South), 2014.

[16] 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 Transactions on Software Engineering*, vol. 43, no. 3, pp. 272–297, 2017.

[17] 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, 2017.

[18] A. Yadav, S. K. Singh, and J. S. Suri, "Ranking of software developers based on expertise score for bug triaging," *Information and Software Technology*, vol. 112, pp. 1–17, 2019.

[19] D.-G. Lee and Y.-S. Seo, "Improving bug report triage performance using artificial intelligence based document generation model," *Human-Centric Computing and Information Sciences*, vol. 10, no. 1, pp. 1–22, 2020.

[20] S. Banitaan and M. Alenezi, "Decoba: utilizing developers communities in bug assignment," *12th International Conference on Machine Learning and Applications*, 2013, pp. 66–71, Miami, FL, USA, 2013.

[21] W. Zhang, S. Wang, Y. Yang, and Q. Wang, "Heterogeneous network analysis of developer contribution in bug repositories," in *2013 International Conference on Cloud and Service Computing*, pp. 98–105, Beijing, China, 2013.

[22] T. Zhang and B. Lee, "A hybrid bug triage algorithm for developer recommendation," in *Proceedings of the 28th annual ACM symposium on applied computing*, pp. 1088–1094, Coimbra, Portugal, 2013.

[23] H. Hu, H. Zhang, J. Xuan, and W. Sun, "Effective bug triage based on historical bug-fix information," in *2014 IEEE 25th International Symposium on Software Reliability Engineering*, pp. 122–132, Naples, Italy, 2014.

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

[25] V. Etemadi, O. Bushehrian, R. Akbari, and G. Robles, "A scheduling-driven approach to efficiently assign bug fixing tasks to developers," *Journal of Systems and Software*, vol. 178, article 110967, 2021.

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

[27] H. Jahanshahi, K. Chhabra, M. Cevik, and A. Baþar, "Dabt: a dependency-aware bug triaging method," in *Evaluation and Assessment in Software Engineering*, pp. 221–230, Trondheim Norway, 2021.

[28] P. Bhattacharya and I. Neamtiu, "Fine-grained incremental learning and multi-feature tossing graphs to improve bug triaging," in *2010 IEEE International Conference on Software Maintenance*, pp. 1–10, Timisoara, Romania, 2010.

[29] A. Tamrawi, T. T. Nguyen, J. Al-Kofahi, and T. N. Nguyen, "Fuzzy set-based automatic bug triaging (nier track)," in *Proceedings of the 33rd International Conference on Software Engineering*, pp. 884–887, Waikiki, Honolulu, HI, USA, 2011.

[30] J. Anvik and G. C. Murphy, "Reducing the effort of bug report triage," *ACM Transactions on Software Engineering and Methodology (TOSEM)*, vol. 20, no. 3, pp. 1–35, 2011.

[31] J. Xuan, H. Jiang, Z. Ren, and W. Zou, "Developer prioritization in bug repositories," in *2012 34th International Conference on Software Engineering (ICSE)*, pp. 25–35, Zurich, Switzerland, 2012.

[32] S. Banitaan and M. Alenezi, "Tram: An approach for assigning bug reports using their metadata," in *2013 Third International Conference on Communications and Information Technology (ICCIT)*, pp. 215–219, Beirut, Lebanon, 2013.

[33] M. Alenezi, K. Magel, and S. B. JSW, "Efficient bug triaging using text mining," *Journal of Software*, vol. 8, no. 9, pp. 2185–2191, 2013.

[34] J. Xuan, H. Jiang, Y. Hu et al., "Towards effective bug triage with software data reduction techniques," *IEEE Transactions on Knowledge and Data Engineering*, vol. 27, no. 1, pp. 264–280, 2015.

[35] 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 Software Engineering*, vol. 21, no. 4, pp. 1533–1578, 2016.

[36] A.-C. Florea, J. Anvik, and R. Andonie, "Spark-based cluster implementation of a bug report assignment recommender system," in *International Conference on Artificial Intelligence and Soft Computing*, pp. 31–42, Cham, 2017.

[37] M. Alenezi, S. Banitaan, and M. Zarour, "Using categorical features in mining bug tracking systems to assign bug reports," https://arxiv.org/abs/1804.07803.

[38] A. Sarkar, P. C. Rigby, and B. Bartalos, "Improving bug triaging with high confidence predictions at ericsson," in *2019 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pp. 81–91, Cleveland, OH, USA, 2019.

[39] Y. Zhao, T. He, and Z. Chen, "A unified framework for bug report assignment," *International Journal of Software Engineering and Knowledge Engineering*, vol. 29, no. 4, pp. 607–628, 2019.

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

[41] V. Dedik and B. Rossi, "Automated bug triaging in an industrial context," in *2016 42th Euromicro conference on software engineering and advanced applications (SEAA)*, pp. 363–367, Limassol, Cyprus, 2016.

[42] J. Xuan, H. Jiang, Z. Ren, J. Yan, and Z. Luo, "Automatic bug triage using semi-supervised text classification," https://arxiv.org/abs/1704.04769.

[43] X. Peng, P. Zhou, J. Liu, and X. Chen, *Improving Bug Triage with Relevant Search*, SEKE, 2017.

[44] S. F. A. Zaidi and C.-G. Lee, "Learning graph representation of bug reports to triage bugs using graph convolution network," in *2021 International Conference on Information Networking (ICOIN)*, pp. 504–507, Jeju Island, Korea (South), 2021.

[45] C. A. Choquette-Choo, D. Sheldon, J. Proppe, J. Alphonso-Gibbs, and H. Gupta, "A multi-label, dual-output deep neural network for automated bug triaging, pp. 937–944, Boca Raton, FL, USA, 2019.

[46] T. W. W. Aung, Y. Wan, H. Huo, and Y. Sui, "Multi-triage: a multi-task learning framework for bug triage," *Journal of Systems and Software*, vol. 184, article 111133, 2022.

[47] S. F. A. Zaidi, H. Woo, and C. -G. Lee, "A graph convolution network-based bug triage system to learn heterogeneous graph representation of bug reports," *IEEE Access*, vol. 10, pp. 20677–20689, 2022.

[48] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: pre-training of deep bidirectional transformers for language understanding," https://arxiv.org/abs/1810.04805.

[49] Y. Guo, H. Shi, A. Kumar, K. Grauman, T. Rosing, and R. Feris, "Spottune: transfer learning through adaptive fine-tuning," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pp. 4805–4814, Long Beach California, 2019.

[50] S.-A. Rebuffi, A. Kolesnikov, G. Sperl, and C. H. Lampert, "icarl: Incremental classifier and representation learning," in *Proceedings of the IEEE conference on Computer Vision and Pattern Recognition*, pp. 2001–2010, Honolulu, Hawaii, 2017.