The Journal of Systems and Software 210 (2024) 111961

Contents lists available at ScienceDirect

The Journal of Systems & Software

journal homepage: www.elsevier.com/locate/jss

An empirical assessment of different word embedding and deep learning models for bug assignment☆

Rongcun Wang [a,*], Xingyu Ji [a], Senlei Xu [a], Yuan Tian [b], Shujuan Jiang [a], Rubing Huang [c]

[a] School of Computer Science and Technology, China University of Mining and Technology, No. 1 Daxue Road, Xuzhou, 221116, Jiangsu, China
[b] School of Computing, Queen's University, Kingston, Canada
[c] School of Computer Science and Engineering, Macau University of Science and Technology, Taipa, Macao Special Administrative Region of China

ARTICLE INFO

Dataset link: https://github.com/AI4BA/dl4ba

Keywords:
Bug assignment
Bug report
Word embedding
Deep learning
Classification

ABSTRACT

Bug assignment, or bug triage, focuses on identifying the appropriate developers to repair newly discovered bugs, thereby managing them more effectively. Several deep learning-based approaches have been proposed for automated bug assignment. These approaches view automated bug assignment as a text classification task — the textual description of a bug report is utilized as the input and the potential fixers are regarded as the output labels. Such approaches typically depend on the classification performance of natural language processing and machine learning techniques. Various word embedding and deep learning models have emerged continuously. The effectiveness of those approaches depends on the chosen deep learning model, used for classification, and the word embedding model, used for representing bug reports. However, prior research does not empirically evaluate the impacts of various word embedding and deep learning models for automated bug assignment. In this paper, we conduct an empirical study to analyze the performance variations among 35 deep learning-based automated bug assignment approaches. These approaches are based on five word embedding techniques, *i.e.*, Word2Vec, GloVe, NextBug, ELMo, and BERT, and seven text classification models, *i.e.*, TextCNN, LSTM, Bi-LSTM, LSTM with attention, Bi-LSTM with attention, MLP, and Naive Bayes. We evaluated these combinations across three benchmark datasets, namely Eclipse JDT, GCC, and Firefox, and their mergence i.e., a cross-project dataset. Our main observations are: (1) Bi-LSTM with attention and Bi-LSTM using ELMo are significantly superior to other deep learning models on bug assignment tasks in terms of top-k (k = 1, 5, 10) accuracy and MRR; (2) Both the summary and description of bug reports are useful for bug assignment, but the description is more useful than the summary; (3) The training corpus for word embedding models has a significant impact on the performance of deep learning-based bug assignment methods. Our results show the importance of tuning different components (e.g. word embedding model, classification model, and textual input) in deep learning-based automated bug assignment methods and provide important insights for practitioners and researchers.

1. Introduction

Bugs are inevitable during software development and maintenance, posing significant challenges to large-scale open-source software projects (Xuan et al., 2015). To shorten the software development cycle and reduce maintenance costs, large-scale open-source software projects leverage bug tracking systems (*e.g.* JIRA,[1] Bugzilla,[2] and BugNet[3]) for highly efficient bug management such as submission, fixing, and verification. These systems store bugs in the form of bug reports, which contain detailed information that could help developers reproduce and fix the bugs, such as summaries, descriptions, and relevant code. The workflows of these bug tracking systems are similar (Jahanshahi et al., 2021): (1) When a developer encounters an error, she/he writes the corresponding issue report and submits it to the bug tracking system. (2) The triager determines whether the issue

☆ Editor: Aldeida Aleti.

* Corresponding author.
E-mail address: rcwang@cumt.edu.cn (R. Wang).

[1] https://www.atlassian.com/software/jira/bug-tracking
[2] https://www.bugzilla.org/
[3] https://bugnetproject.com/
[4] https://github.com/dotnet/aspnetcore
[5] https://gcc.gnu.org/bugzilla

https://doi.org/10.1016/j.jss.2024.111961
Received 17 August 2023; Received in revised form 10 November 2023; Accepted 3 January 2024
Available online 6 January 2024
0164-1212/© 2024 Elsevier Inc. All rights reserved.

is a bug and the type of this bug based on the content of the issue report. (3) The triager assigns the bug to the appropriate fixer. (4) When the assignee cannot fix the bug, this bug is put back into the bug-tracking system. Meanwhile, the status of the bug report is changed to "New" waiting for the next round of assignment. (5) when the bug is successfully fixed, the bug is closed. As seen in this workflow, it is tedious and time-consuming work for triagers (Jahanshahi and Cevik, 2022). Particularly, as software continually evolves, the daily number of bug reports submitted to these systems greatly rises. For example, for ASP.NET Core project,[4] approximately 115.8 bug reports were posed per month from May to October 2023 on average. The GCC Bugzilla[5] received 85 reports in the recent week, from October 3 to October 10, 2023. Faced with such a substantial volume of bug reports, triagers need to make great efforts to understand them and assign them to the appropriate developers for bug resolution. Particularly, for complex open-source software products, bug assignment is more challenging because the number of involved bug fixers is big and they have different skills.

To address the above challenge, researchers have proposed two main types of automated bug assignment approaches, *i.e.*, information retrieval-based (Yang et al., 2014; Hu et al., 2018; Matter et al., 2009; Sajedi-Badashian and Stroulia, 2020; Xia et al., 2017; Alazzam et al., 2020) and text classification-based bug assignment methods (Murphy and Cubranic, 2004; Xuan et al., 2015; Dedík and Rossi, 2016; Ahsan et al., 2009; Naguib et al., 2013; Sbih and Akour, 2018; Sarkar et al., 2019; Sawarkar et al., 2019; Lee et al., 2017; Mani et al., 2019; Guo et al., 2020). Bug assignment methods based on information retrieval recommend bug fixers according to the similarity between historical bug reports and newly-arrived bug reports, and the corresponding relationship between historical bug reports and fixers. Unlike information retrieval-based bug assignment methods, bug assignment methods based on text classification extract textual features from bug reports and build classifiers to predict the labels, *i.e.*, the actual fixers of bugs. The text classification-based bug assignment methods consist of two-stage operations. One is to generate word vector representations by various techniques such as the term frequency-inverse document frequency (TF-IDF) (Xuan et al., 2015; Dedík and Rossi, 2016; Ahsan et al., 2009; Sarkar et al., 2019) and the word to vector (Word2Vec) (Lee et al., 2017; Mani et al., 2019; Guo et al., 2020). The other is to predict bug fixers by different classification models such as Naive Bayes (NB) (Xuan et al., 2015; Murphy and Cubranic, 2004), Support Vector Machines (SVM) (Dedík and Rossi, 2016; Ahsan et al., 2009), and ensemble learning methods (Sbih and Akour, 2018).

Recently, various deep neural networks have been successfully applied in many tasks related to software engineering such as code comment generation (Hu et al., 2020), vulnerability detection (Chakraborty et al., 2022), and software defect prediction (Giray et al., 2023). Moreover, they have achieved better performance compared with traditional machine learning-based bug assignment methods (Lee et al., 2017; Lee and Seo, 2019). For example, both Lee et al. (2017) and Guo et al. (2020) utilized Word2Vec to generate textual representations of bug reports. They employed convolutional neural networks (CNN) for text classification to predict fixers. Xi et al. (2018) proposed a sequence-to-sequence model for bug assignment. Recurrent Neural Networks (RNN) and Gate Recurrent Unit (GRU) are used for feature extraction and model prediction, respectively. Choquette-Choo et al. (2019) used Dual-Output Deep Neural Networks (DNN) for bug assignment. Bug assignment based on deep learning has made remarkable progress. This makes bug assignment to be more precise and cost-effective, ultimately enhancing the efficiency of software maintenance and reducing overall bug resolution costs.

In previous studies (Lee et al., 2017; Mani et al., 2019; Guo et al., 2020), Word2Vec (Mikolov et al., 2013) was predominantly used to generate text vector representations, and CNN (Zhang and Wallace, 2017) was commonly employed as the classifier for bug assignment. Existing studies have demonstrated the effectiveness of representation learning and deep learning in bug assignment. It is worth noting that the promising and recently popular representation learning methods, like global vectors (GloVe), embeddings from language models (ELMo), and bidirectional encoder representations from Transformers (BERT), have been rarely utilized for bug assignment. Likewise, the deep learning classification models widely used in natural language processing such as multi-layer perceptron (MLP) (Taud and Mas, 2018) and long short-term memory (LSTM) (Graves, 2012) have been rarely applied for bug assignment. The bug assignment methods based on different word embedding and deep learning classification models could result in different performances. Nevertheless, no previous studies empirically investigated the effects of the methods using various representation learning and deep learning classification models on automated bug assignment tasks. This restricts further development of the bug assignment task, especially in picking the relatively optimal deep learning models for addressing this task.

In this context, we designed and conducted an empirical assessment of different word embedding and deep learning models for bug assignment. Three word-level embedding models, *i.e.*, Word2Vec, GloVe, and a fine-tuned skip-gram model based on the bug-specialized domain called NextBug (Du et al., 2022), and two sequence-based representation learning models, *i.e.*, ELMo and BERT, were empirically accessed. Seven popular deep learning classifiers including TextCNN (Kim, 2014), LSTM (Graves, 2012), bidirectional LSTM (Bi-LSTM), LSTM with attention, Bi-LSTM with attention, MLP (Taud and Mas, 2018), and Naive Bayes (NB) (Xu et al., 2017), were also empirically evaluated. To our knowledge, we are first to empirically investigate the effects of the models based on different word embedding techniques and deep learning classification models on automated bug assignment. The experiments were designed and conducted on three widely-used bug assignment datasets, namely *Eclipse JDT*, *GCC*, and *Firefox*, and a *cross-project* dataset. The following three research questions are set to better understand the effects of three key components, *i.e.*, learning models, bug report elements, and source of training corpus on bug assignment based on text classification.

RQ1: Are there differences in the performance of various deep learning models for automated bug assignment? We consider four representation learning models: word-level models with local sensitivity (*e.g.* GloVe and Word2Vec) and sentence-level models with global sensitivity (*e.g.* BERT and ELMo) for generating word embeddings. Moreover, seven deep learning classification models including TextCNN, LSTM, Bi-LSTM, LSTM with attention, Bi-LSTM with attention, MLP, and Naive Bayes are used for predicting bug fixers. The extracted features could vary with the models, resulting in different predictive results. Therefore, we set up this research question to investigate which deep learning models are more suitable for bug assignment tasks.

RQ2: Is the description useful for bug assignment? What is the optimal weight between the summary and description? The textual information of bug reports typically consists of a summary and description. Compared with the description, the summary is more concise and contains semantic connections that are more closely related to a bug. Contrarily, the description provides more details related to a bug. In view of the above differences, we proposed this research question to investigate their contributions to bug assignment tasks.

RQ3: To what extent does the training corpus influence the performance of the representation learning models for bug assignment tasks? Most representation learning models are trained based on a general domain corpus. The models trained based on general corpus tend to have stronger generalization capabilities. However, the models trained based on bug-specialized domain corpus (*e.g.* NextBug) may be more suitable for bug-related tasks. Therefore, we proposed this research question to investigate whether there are differences in the performance of the models trained on general corpus and bug-specialized domain corpus in bug assignment tasks.

The whole contributions of this paper are listed as follows.

- We comprehensively investigated the effects of different word embedding and deep learning models on bug assignment. The differences among all investigated models were qualitatively and quantitatively accessed.
- We designed nine strategies representing different weights between the summary and description of the bug report. The nine strategies were empirically evaluated for the study of their relative importance between the summary and description.
- We conducted an empirical evaluation of the impacts of training corpus from general and bug-specialized domains on bug assignment tasks.

The results and source code related to this study are available at https://github.com/AI4BA/dl4ba.

The rest of this paper is organized as follows. Section 2 summarizes the related work. The experimental design and analysis of results are presented in Sections 3 and 4, respectively. We discuss the threats to validity in Section 5. Finally, the paper concludes with the future work in Section 6.

2. Related work

Previous studies on automatic bug assignment can be roughly categorized into three types based on the used techniques: information retrieval-based, text classification-based, and hybrid methods. The first aims to assign bugs based on the similarity between the newly arrived bug reports and historical bug reports. The second regards bug assignment as a classification issue. In other words, a classification model is trained for bug assignment using the text features extracted from the bug reports with the assignees as labels. The last mainly combines multiple techniques including IR-based methods, learning-based methods, and tossing graphs for automated bug assignment.

2.1. Information retrieval-based bug assignment methods

Information retrieval-based bug assignment methods are based on the observation that the fixers who previously solved similar bugs are regarded as the most suitable candidates for newly arrived bug reports (Zhang et al., 2015). Most of the methods search for bug reports in the repository that are the most similar to the newly arrived report. The bug fixers corresponding to the retrieved bug reports are recommended as the appropriate fixers. For instance, Yang et al. (2014) extracted topics from historical bug reports based on Latent Dirichlet Allocation (LDA). According to the extracted topics, a newly-arrived bug report can be determined to the topics it belongs to. The historical bug reports having the same features and topics as the newly-arrived report can be retrieved. The fixers corresponding to the retrieved bug reports are regarded as the fixers of the newly arrived bug report. Hu et al. (2018) proposed a document embedding model-based method for improving the performance of Yang et al.'s method (Yang et al., 2014). Similarly, Matter et al. (2009) presented an expertise model of developers based on their contributed source code. According to the cosine similarity between the vocabulary of bug reports and the vocabulary in source code contributions, the newly arrived bug report can be assigned to developers.

Sajedi-Badashian and Stroulia (2020) presented a vocabulary and time-aware bug assignment method. The vocabulary consists of programming keywords in bug reports belonging to Stack Overflow tags. They argued developers' recent expertise is more important than past expertise. Furthermore, they considered the importance of the keywords and time of usage to improve the TF-IDF metric. Xia et al. (2017) proposed a bug assignment method, in which they utilized a multi-feature topic model (MTM) to conduct topic modeling based on product and component information from historical bug reports. They employed TopicMiner to obtain the topic distribution of newly arrived bug reports, enabling the retrieval of similar reports. Alazzam et al. (2020) proposed a feature augmentation approach that leverages graph partitioning based on neighborhood overlap to augment features. By considering the similarity between summaries of bug reports and each term cluster, the approach concatenates terms from strong clusters into the feature vectors of the summaries. This approach enhances bug assignment methods based on machine learning.

2.2. Text classification-based bug assignment methods

Bug assignment methods based on text classification can be further classified into traditional machine learning-based methods and deep learning-based methods.

2.2.1. Traditional machine learning-based methods

Murphy and Cubranic (2004) first treated the problem of bug assignment as a case of text classification. The NB algorithm was employed to predict the developer based on the bug's description. Similarly, Xuan et al. (2015) used TF-IDF to generate word vector representations for the summaries and descriptions of bug reports, and utilized the NB classifier for bug assignment. Compared with the study (Xuan et al., 2015), Dedik et al. (Dedík and Rossi, 2016) extracted more features from bug reports including summaries, descriptions, keywords, and stack traces. They combined TF-IDF with SVM for automated bug triaging in an industrial context. Ahsan et al. (2009) extracted textual information such as summaries and descriptions from bug reports and parsed the names of developers as labels. They represented the textual information using the TF-IDF and applied feature selection and latent semantic indexing methods to reduce the dimension of a term document matrix. Six machine learning classifiers were compared. The best one is based on SVM and latent semantic indexing. Naguib et al. (2013) utilized developers' historical activities from the bug tracking system to create an activity profile for each developer describing her/his roles, expertise, and level of involvement in the project. Subsequently, LDA was utilized to generate the topic feature vectors of bug reports. SVM was employed to determine the appropriate fixer.

Sbih et al. (Sbih and Akour, 2018) proposed a model for bug assignment. The model was evaluated using ensemble learning methods such as bagging, boosting, and decorating, along with classifiers including Bayes Net, Naive Bayes, Decision Table, Random Tree, and J48. Sarkar et al. (2019) considered three types of attributes, *i.e.*, textual, categorical, and log attributes. Several traditional classifiers such as K-Nearest Neighbor (KNN) and Logistic Regression(LR) were empirically evaluated. Sawarkar et al. (2019) utilized the Bag-of-Words (BOW) model to extract feature vectors from textual information. They employed multi-label classification algorithms, including Random Forest, SVM, and J48, to predict the fixers for the given bug reports.

2.2.2. Deep learning-based methods

Lee et al. (2017) utilized the Word2Vec representation learning model to learn representations of summaries and descriptions in bug reports. The CNN was used to extract text features and predict bug fixers. Similarly, Mani et al. (2019) also used Word2Vec and introduced an attention mechanism to learn sentence-level semantic features for bug assignment. They proposed a bug assignment method based on a Bidirectional Recursive Neural Network. Guo et al. (2020) considered developer activity in addition to summaries and descriptions and proposed a bug assignment method based on Word2Vec and CNN. Xi et al. (2018) utilized the RNN to generate textual features from the text in bug reports and model the dependencies among developers depending on tossing sequences. After that, a sequence of candidate fixers was generated. Choquette-Choo et al. (2019) employed the Dual-Output Deep Neural Networks (DNN) to perform bug assignment. They utilized the model to predict the team category, which in turn, helps in predicting the appropriate fixers. Existing studies predominantly use Word2Vec and CNN to generate word embeddings and predict the right fixers,

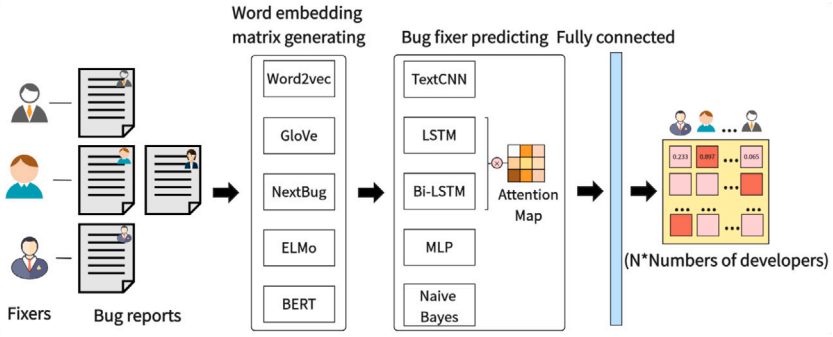


Fig. 1. Overview of automated bug assignment based on representation learning models and deep learning classifiers.

respectively. Although recent representation learning models such as GloVe, ELMo, and BERT, and deep learning classifiers like LSTM and Bi-LSTM are popular in the field of natural language processing, they have been rarely applied for addressing bug assignment. Moreover, the word embeddings vary with the embedding models because they use different algorithms. Likewise, different classifiers could generate different classification results. This drives us to conduct this study. Different from the above studies, we empirically investigated many more word embedding and deep learning classification models for automated bug assignment.

2.3. Hybrid methods

Bhattacharya et al. (2012) investigated the impact of bug tossing graphs and machine learning classifiers on the prediction of bug assignment. They utilized a probabilistic graph-based model to represent bug reports and investigated its impact on bug assignment tasks across Naive Bayes, Bayesian Networks, C4.5, and SVM classifiers. Xia et al. (2015) proposed a composite method called DevRec for developer recommendation to resolve bugs using Euclidean distance metric, multi-label k-nearest neighbor (ML-KNN), and topic modeling. Wu et al. (2011) proposed a developer recommendation method with K-Nearest-Neighbor Search and Expert Rating (DREX) for bug assignment. This method utilizes KNN and expert rating to assign bugs and evaluates the model using precision and social network metrics.

Furthermore, there are other methods for automated bug assignment. For example, the approach (Jeong et al., 2009) trained a tossing graph model, which effectively reduces the tossing length of the assignment process and improves overall accuracy. Yadav et al. Yadav and Singh (2020) constructed a developer recommendation list based on individual developers' expertise and proficiency in bug fixing, thereby optimizing the assignment process. Sun et al. (2017) proposed a method for developer recommendation based on data mining. The commits relevant to the issue requests were extracted by the mining software repository. The cosine similarity between historical commits and new bug reports is calculated for deciding the related historical commits to the changed source code. The contributors corresponding to those determined commits are given the scores by collaborative topic modeling. According to the scores, the appropriate contributors are recommended as the fixers of the new issue. To address the problem of limited informative contents of bug reports, Zhang et al. (2017) proposed a bug report enrichment method to improve the performance of automated fixer recommendations. Different from the above studies, Aung et al. (2022) used code snippets in addition to bug reports and proposed a transformer-based model for bug triage.

3. Experimental design

This section describes the experiments in detail including the used datasets, data pre-processing, technical framework, evaluation metrics, and statistical analysis methods. All experiments were conducted on Ubuntu 20.04 equipped with an Intel(R) Xeon(R) Platinum 8338C CPU @ 2.60 GHz, 80 GB RAM, and an RTX 3090–24 GB video card.

3.1. Datasets

The experiments were conducted on three widely used projects, *i.e.*, *Eclipse JDT*,[6] *GCC*,[7] and *Firefox*.[8] There are two primary reasons for the choice of the *Eclipse JDT* and *Firefox* datasets. Firstly, Lee et al. (Lee and Seo, 2019) enumerated 18 relevant papers, out of which 16 studies used the data from *Eclipse JDT* and *Firefox*. Secondly, both the two projects adopt Bugzilla as their bug tracking system, ensuring the consistency of bug report structures. The *Eclipse JDT* dataset focusing on Java Development Tooling (JDT) comprises 842 developers and 1,465 bug reports spanning from October 20, 2013 to October 20, 2016. The *Firefox* dataset contains 13,668 bug reports contributed by 69 developers during the period from August 1, 2013, to July 31, 2016. Additionally, we also incorporated the *GCC* dataset commonly used in previous studies (Zhang et al., 2017; Anvik et al., 2006; Xia et al., 2015; Yin et al., 2018). This *GCC* dataset (Yin et al., 2018) we used contains 2103 bug reports involving 82 developers. The bug reports from the three datasets contain essential information such as bug ID, assignee, summary, description, and status. To ensure the reliability of the experiments, we only used bug reports with the status 'RESOLVED' or 'FIXED'. The severity levels of those bug reports are 'P1 Critical' or 'P2 Normal'. We also conducted a cross-project validation, *i.e.*, the three datasets are merged into a dataset, to evaluate comprehensively the performance of the studied models in bug assignments.

3.2. Technical framework

The process of bug assignment based on text classification mainly involves three steps: data pre-processing, word embedding matrix generation, and bug fixer prediction. The overall technical framework is illustrated in Fig. 1. The bug reports are first pre-processed. Then, the summary and description of each bug report are input to one of the studied representation learning models to obtain embedding vectors. Embedding vectors are subsequently fed to one of the studied deep learning classifiers. Finally, the correlation probabilities between each bug report and all fixers are computed via a fully connected layer to obtain the recommended top-k fixers.

3.2.1. Data pre-processing

The bug report consists of various attributes, such as bug ID, component, bug status, bug assignee, summary, and description. Like previous studies (Lee et al., 2017; Mani et al., 2019; Guo et al., 2020), we also focus on the summary and description attributes. These extracted bug assignees are considered the labels for a supervised classifier. To pre-process the bug report data, we removed digits, punctuation marks, URLs, spaces, and non-ASCII characters from the text. Additionally, we

[6] https://bugs.eclipse.org/bugs
[7] https://gcc.gnu.org/bugzilla
[8] https://bugzilla.mozilla.org

utilized the NLTK toolkit (Bird et al., 2009) to remove stopwords and converted all uppercase letters to lowercase. The above operations aim at cleaning the text and eliminating irrelevant information, making it more suitable for further training and classification.

3.2.2. Word embedding matrix generating

After pre-processing, the next step is the tokenization of the text. The tokens are then fed into the representation learning models.

For the word-level embedding models, an additional operation is to generate a vocabulary dictionary corresponding to the text and form word vectors using Word2Vec, GloVe, or NextBug. The vocabulary dictionary is a collection of all unique words in the training corpus, each of which has a unique index. Building the vocabulary dictionary involves the following steps. The first step is to count word frequencies i.e., traversing the entire corpus to count the occurrence frequency of each word. Then, the words are sorted in descending order of frequency, i.e., placing the most common words at the front of the vocabulary dictionary. The last step is to assign a unique index to each word, thus generating a vocabulary dictionary by mapping words to IDs. The vocabulary dictionary is used to map words in the text to ID sequences, which are then mapped to a high-dimensional vector space through Wor2Vec, GloVe, or NextBug. Since the text could contain words that are not trained by the above models, the word embedding matrix is initialized randomly following a normal distribution before using the word-level embedding models. Subsequently, the text is traversed to generate the corresponding text vectors according to the vocabulary dictionary. Different from the word-level embedding models, the sequence-based models (*e.g.* ELMo and BERT) can accept the token list as their inputs for generating the corresponding text vectors directly, rather than an additional operation such as generating a vocabulary in the case of word-level embedding models.

Note that for Word2Vec,[9] and GloVe[10] we use 300-dimensional word embeddings. As for NextBug,[11] it is a pre-trained binary file with 500-dimensional word embeddings. For BERT and ELMo, we used the BERT-BASE[12] model with 2 layers, 768 hidden layers, 12 multi-heads, and 118M parameters. Thus, the dimension of word embeddings generated by BERT is 768. Due to the limit of the experimental budget, the maximum sequence length for BERT is 512. Note that we did not fine-tune the BERT. There are two main reasons. One is that the limited experimental budget cannot provide sufficient resources for fine-tuning BERT. The other is more fairly to compare the studied models as not all studied models support the fine-tuning mode. On the other hand, ELMo[13] generates word embeddings with a dimension of 1024. To preserve semantic information without loss or distortion as possible as we can, the vectors are not truncated or padded.

3.2.3. Bug fixer predicting

Once the word embedding matrix is generated by a certain representation learning model, it can be fed into the deep learning classifiers, including TextCNN, LSTM, Bi-LSTM, LSTM with attention, Bi-LSTM with attention, MLP, and Naive Bayes for bug fixers prediction.

3.3. Classification model

3.3.1. TextCNN (Text Convolutional Neural Network)

TextCNN is a text classification model based on CNN. It consists of convolutional layers, pooling layers, and fully connected layers. As shown in Fig. 1, the model takes the word embedding matrix as an input and performs convolution operations with convolutional kernels of different sizes for extracting feature maps from different regions. Based on the study (Zhang and Wallace, 2017), we selected convolutional kernels of size [3, 4, 5] for convolution operation. The work (Zhang and Wallace, 2017) investigated the impact of the number of convolutional kernels on multiple datasets such as the Sentence Polarity dataset (MR), Stanford Sentiment Treebank (SST-1), Subjectivity dataset (Subj), and Question Classification (TREC). Given the similarity between the bug assignment problem and question classification, we opted for the same number of convolutional kernels as TREC, i.e., 512.

After convolution, the feature maps have a shape of (N, L-[3, 4, 5]+1, 512). These feature maps are then subjected to max-pooling to reduce dimension while preserving important features. The pooled vectors are concatenated, flattened, and passed through fully connected layers to compute the probability scores for each bug report with respect to each developer, thereby completing the classification task.

3.3.2. LSTM (Long Short-Term Memory)

LSTM as a variant of RNNs, can process sequential data. Compared with the traditional RNN model, LSTM can not only better capture long-term dependencies within a sequence by gate mechanism, but also effectively address the issues of gradient disappearance and exploding when dealing with long sequences. Therefore, we also employed the LSTM model for predicting bug fixers. During the training of the LSTM model, the loss between the model's predictions and the ground truth is computed. The loss is back-propagated to update the weights and biases in the network. This process is repeated until the model converges and achieves optimal performance. Then, the outputs of the last hidden layers are fed into a linear layer for computing the probability scores.

Increasing the size of hidden layers can enhance the network's learning ability, enabling it to better adapt to complex patterns and data. However, more complex networks easily suffer from more computational resources and longer training time. In the same way, an excessive size of hidden layers may lead to over-fitting and training more difficult. Therefore, it is necessary to choose an appropriate size of hidden layers for achieving a balance between accuracy and efficiency. A prior study (Nowak et al., 2017) illustrates that LSTM can get better performance for the text classification task when the hidden layer size is equal to 128. Considering the bug assignment task is highly similar to the text classification task, we set the same hidden layer size i.e.,128.

3.3.3. Bi-LSTM (Bi-directional Long Short-Term Memory)

Bi-LSTM is another variant of RNNs that incorporates memory cells and gate mechanism. Similar to LSTM, Bi-LSTM excels in modeling sequential data. As shown in Fig. 1, the Bi-LSTM consists of two LSTM layers. One is responsible for processing the input sequence in the forward direction, while the other is used in the backward direction. The final output is generated by concatenating the outputs of the two layers. Subsequently, the output of the hidden layers is passed through a linear layer, which performs the computation of probability scores.

Compared with LSTM, Bi-LSTM is capable of capturing both the preceding and succeeding context information in the input sequence. By utilizing the forward and backward LSTM layers, it can simultaneously consider the information before and after the current time step, providing a more comprehensively contextual understanding. The size of hidden layers of Bi-LSTM is the same as that of LSTM, *i.e.*, 128.

3.3.4. MLP (Multi-layer Perceptron)

MLP is a feed-forward neural network model, also known as a fully connected neural network. It consists of multiple layers, *i.e.*, an input layer, multiple hidden layers, and an output layer. The input layer is responsible for receiving the word embedding matrix. The MLP utilized in this paper comprises two hidden layers, each of which consists of multiple neurons. These neurons are used for computing the weighted sums of the outputs from the previous layer. The linear factors are eliminated by the ReLU activation function. The final results are then outputted through the output layer. During the model training process, the parameters of the MLP are updated using back-propagation to minimize the loss. Similar to LSTM, the hidden layer size of MLP is also 128.

[9] https://github.com/mmihaltz/word2vec-GoogleNews-vectors
[10] https://nlp.stanford.edu/projects/glove/
[11] https://github.com/xiaotingdu/DeepSIM
[12] https://github.com/google-research/BERT/
[13] https://s3-us-west-2.amazonaws.com/allennlp/models/elmo

3.3.5. Naive Bayes

Naive Bayes is a statistical classification algorithm, which has been widely used in fields such as text classification (Dai et al., 2007; Frank and Bouckaert, 2006), spam filtering (Metsis et al., 2006; Rusland et al., 2017), and sentiment analysis (Tan et al., 2009; Wongkar and Angdresey, 2019). This algorithm is built upon the principles of Bayesian probability theory. When dealing with classification problems, Naive Bayes computes the conditional probabilities of different labels for a given sample and selects the label with the highest probability as the classification result. In this paper, after obtaining the word embedding matrix, the model is fitted using the feature vectors from the training data.

3.3.6. Attention mechanism

The attention mechanism is used for allocating different attention weights to different parts of the sequence input, thereby capturing key and contextual relevance information more effectively. It plays an important role in processing sequential data.

For both LSTM and Bi-LSTM using the attention mechanism, two linear layers are applied for calculating attention scores based on the outputs of the LSTM or Bi-LSTM. The linear layers can learn the importance of different positions based on the input feature. The attention scores are then normalized using the softmax function to obtain attention weights. Next, the attention weights are multiplied element-wise with the outputs of the LSTM or Bi-LSTM and summed to get the final text vectors. This weighted sum process allows the model to better focus on multiple important parts of the sequence, regardless of the sequence length. By introducing the attention mechanism, the model can automatically learn and concentrate its attention on the important parts of the input sequence for a given task, thereby improving the model's performance.

3.4. Evaluation metric

The model's output is the likelihood of each bug report being associated with all potential fixers, and a ranked list of recommended bug fixers is generated based on this likelihood in descending order. The top-k accuracy metric gauges the likelihood of the bug fixers recommended by a bug assignment method including the actual bug fixers. In contrast to traditional metrics such as accuracy, precision, recall, and F1 score, the metrics such as top-k accuracy and top-k precision focus on the performance of the top-k bug fixers, rather than the overall performance. Assessing the overall performance could be less meaningful for bug assignment because the bug assignment problem concerns primarily the recommendation of the top-k fixers, especially the top 1 fixer. Furthermore, even if the overall performance is high, if the quality of the top-k recommended bug fixers is low, the model is considered to be unsuccessful. Compared with top-k accuracy, top-k precision measures how many fixers in the top-k recommended list are truly relevant. In a top-k recommendation list, there can only be 0 or 1 truly relevant fixer. Therefore, calculating top-k precision also becomes meaningless. Similarly, top-k recall and F1 score cannot also be inferred. Finally, as previous studies (Aung et al., 2022; Yang et al., 2014; Jeong et al., 2009; Xia et al., 2015), we also employed top-k accuracy to evaluate the impacts of various deep learning models on bug assignment tasks. It is defined as follows:

$$Top-k = \frac{\sum_{i=1}^{N} predict_i(truth, top-k\ list)}{N}, \tag{1}$$

where N, $truth$, and $top-k\ list$ represent the number of bug reports, a real bug fixer, and a recommendation list containing k bug fixers generated by a bug assignment method, respectively. $predict_i(truth, top-k\ list)$ denotes the recommended result of the ith bug report. If $truth$ belongs to $top-k\ list$, it is equal to 1, otherwise 0.

Top-k accuracy does not consider the specific positions of the recommended bug fixers. To address the limitation of top-k accuracy, we also employed mean reciprocal rank (MRR) (Voorhees et al., 1999). MRR represents the mean of the reciprocals of the ranks of the true bug fixer in the recommendation list. It is defined as follows:

$$MRR = \frac{1}{|N|}\sum_{i=1}^{|N|}\frac{1}{rank_i}, \tag{2}$$

where N and $rank_i$ represent the number of bug reports and the rank of the true fixer in the ith bug report's recommendation list, respectively.

Table 1
The key parameters for training deep learning models.

Parameter	Value
Epochs	12
Batch size	32
Learning rate	0.001
Filter size	3,4,5
Nums of filters	512
Size of hidden layers	128

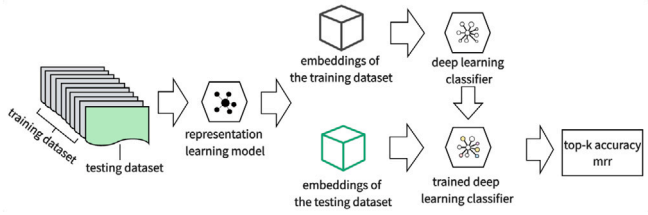


Fig. 2. Model training and testing process.

3.5. Model training and testing

The parameters used during training deep learning models are shown in Table 1.

The training and testing process is shown in Fig. 2. The ten-fold cross-validation as a commonly used cross-validation method was employed for evaluating the performance of a model. The original dataset is divided into 10 equal-sized subsets at random, out of which 9 subsets are used as training data, and the remaining is used as testing data. For each iteration, the dataset is fed into one of the studied representation learning models to generate embedding vectors for both the training and testing datasets. Subsequently, one of the studied deep learning classifiers is trained on the training dataset, and training is stopped once the model reaches convergence. The trained model is tested using the test dataset. The top-k accuracy and MRR values are calculated. The above process is repeated 10 times until each different subset is used as the testing set. The mean values of top-k accuracy and MRR were computed to evaluate the performance of models. By repeatedly training and evaluating a model on different subsets of the dataset, we can understand more comprehensively its performance. This approach tends to mitigate the potential bias or variability caused by a single data partition. By averaging the performance results from multiple folds, we can obtain a more robust and reliable assessment of the performance of models.

3.6. Statistical analysis

The Wilcoxon signed-rank test (Wilcoxon, 1946), a non-parametric hypothesis test, is used for qualitatively analyzing the significant difference between two bug assignment methods with a confidence level of 95%. Compared to other parametric test methods such as the t-test, the Wilcoxon signed-rank test does not assume that the test data follows a certain distribution (e.g. normal distribution). This method is also robust to outliers, as it operates based on rankings rather than values. A null hypothesis is formulated, *i.e.*, there is no significant

Table 2
The relation between δ and effect size.

Value	Effect size
$\|\delta\| < 0.147$	negligible
$\|\delta\| \geq 0.147$ *and* $\|\delta\| < 0.33$	small
$\|\delta\| \geq 0.33$ *and* $\|\delta\| < 0.474$	medium
$\|\delta\| \geq 0.474$	large

difference between the top-k accuracy values generated by the two models. When the *p-value* is less than 0.05, we can reject the null hypothesis, indicating a significant difference between the two models.

Additionally, to quantify the magnitude of the differences between the two different methods, we employ a non-parametric effect size measure, *i.e.*, Cliff's Delta (Cliff, 1993). It can be calculated by the following formula

$$\delta = \frac{(U - D)}{n_1 * n_2}, \tag{3}$$

where U and D represent the number of times that the data in the first group is greater than that in the second group and the data in the second group is greater than that in the first group, and n_1 and n_2 denote the total number of samples in two control groups, respectively. The value of δ ranges from −1.0 to +1.0. The extreme value 0.0 indicates that the values in two control groups are identical, whereas ± 1.0 indicates all values in one group are larger than those in the other group. The absolute value of δ indicates the magnitude of the effect size (Romano et al., 2006). The relation between δ and effect size is shown in Table 2.

4. Experiment result and analysis

In this section, we report the top-k (k = 1, 5, 10) accuracy and MRR of each model. According to the experiment results, we answered the proposed four research questions.

4.1. RQ1: Are there differences in the performance of various deep learning models for the bug assignment problem?

Motivation: Word2Vec and GloVe operate at the word level and generate word embeddings based on tokens. Different from Word2Vec, GloVe incorporating a global co-occurrence matrix can better capture global information. In contrast with the above two models, ELMo and BERT are both sequence-based models, enabling them to capture sentence-level semantic information. Compared to the word embedding models, sequence-based models can capture more contextual information. Similarly, the classification models including TextCNN, LSTM, Bi-LSTM, LSTM with attention, Bi-LSTM with attention, MLP, and Naïve Bayes are different from each other. This means that they could have varied performance on bug assignments. This drives us to set this research question for accessing the difference in their performance.

Method: The text after pre-processing is used to generate word embedding matrices using the representation learning models. The word embedding matrices serve as inputs to different deep learning classifiers for generating a recommendation list of bug fixers as predictors. The performance of different models is evaluated using the top-k accuracy and MRR. The results of 28 deep learning models based on four representation learning models and seven classifiers, are shown in Fig. 3, Fig. 4, Fig. 5, and Fig. 6, where the x-axis of each subgraph represents various models, whereas the y-axis denotes the top-k accuracy or MRR. To clearly illustrate the results, the violin charts with the same classification model are placed in the same group. The violin charts with different classification models are split by a vertical line. Each violin chart represents the distribution of the top-k (k = 1, 5, 10) accuracy and MRR. Wider sections of the violin charts indicate a greater concentration of data in the area. The upper and lower bounds of a violin chart represent the maximum and minimum values of top-k accuracy (k = 1, 5, 10) or MRR. The central short line of a violin chart represents the mean of top-k accuracy (k = 1, 5, 10) or MRR.

Table 3
Corresponding relation of the studied models and the flags in Fig. 7, Fig. 8, Fig. 9, and Fig. 10.

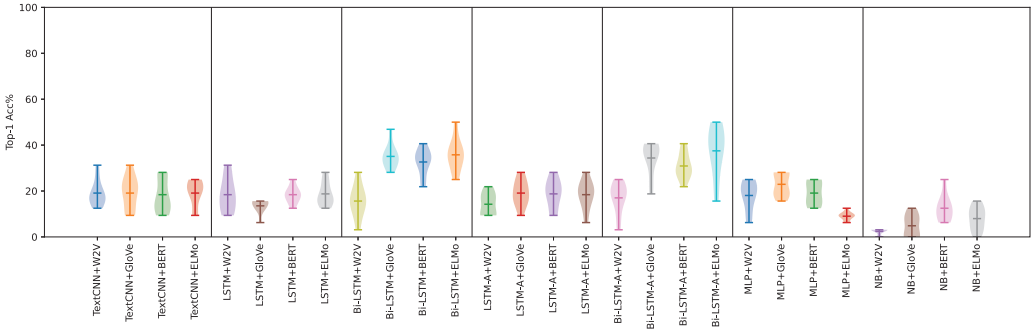
No.	Model	No.	Model
0	Bi-LSTM-A+ELMo	14	Bi-LSTM+W2V
1	Bi-LSTM+ELMo	15	LSTM-A+ELMo
2	Bi-LSTM-A+BERT	16	MLP+W2V
3	Bi-LSTM+BERT	17	TextCNN+GloVe
4	Bi-LSTM-A+GloVe	18	LSTM-A+GloVe
5	Bi-LSTM+GloVe	19	LSTM+ELMo
6	MLP+GloVe	20	LSTM+BERT
7	LSTM-A+BERT	21	LSTM-A+W2V
8	LSTM+W2V	22	TextCNN+BERT
9	LSTM+GloVe	23	MLP+ELMo
10	MLP+BERT	24	NB+BERT
11	Bi-LSTM-A+W2V	25	NB+ELMo
12	TextCNN+W2V	26	NB+GloVe
13	TextCNN+ELMo	27	NB+W2V

Findings: Bi-LSTM-A+ELMo and Bi-LSTM+ELMo are significantly superior to other deep learning models on bug assignment tasks in terms of top-k accuracy and MRR metrics. Moreover, the two models have large or medium effect sizes over other models in most cases. Particularly, compared with the methods using Word2Vec and TextCNN (Lee et al., 2017; Zaidi et al., 2020), **the methods using the above combinations have over 30% improvement in top-k (k = 1, 5, 10) accuracy across all datasets.**

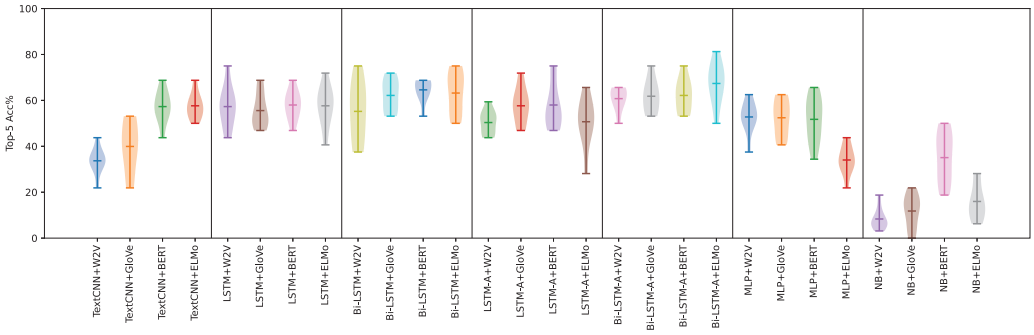
From Fig. 3, Fig. 4, Fig. 5, and Fig. 6, it can be seen that the top-k accuracy and MRR vary among the models. The top-k and MRR values are different from the same representation learning models using different classification models. Likewise, the same classification models using different representation learning models are different from each other. In other words, both representation learning and classification models can influence the accuracy of bug assignment. Bi-LSTM-A using ELMo and Bi-LSTM using ELMo always have higher mean values than other models across all datasets with respect to the top-k accuracy and MRR. Compared with the top-k and MRR values on the three datasets, their values on the cross-project dataset are lower. A likely explanation is that the data distributions of the three datasets are different. The project with the maximum samples plays a critical role in the prediction of the cross-project dataset. The results on the cross-project dataset tend to be the ones on the project with the maximum samples. Furthermore, we conducted the Wilcoxon signed-rank test and Cliff's Delta test. The results are shown in Fig. 7, Fig. 8, Fig. 9, and Fig. 10 where each cell in the left and right subplots shows p-value and δ value, respectively.

To enhance the readability of the result graphs, different values are differentiated with different colors. In the left subplots, each cell contains a *p-value*. A gray cell represents that the *p-value* is less than 0.05. In this case, we can reject the null hypothesis, i.e., there is a significant difference between the two models. Contrarily, a white cell means that the *p-value* is greater than or equal to 0.05, i.e., we cannot reject the null hypothesis. In the right subplots, color represents the magnitude of the effect size, i.e., green for negligible, yellow for small, red for medium, and blue for large. The flags from 0 to 27 denote 28 models. The detailed indicators are shown in Table 3.

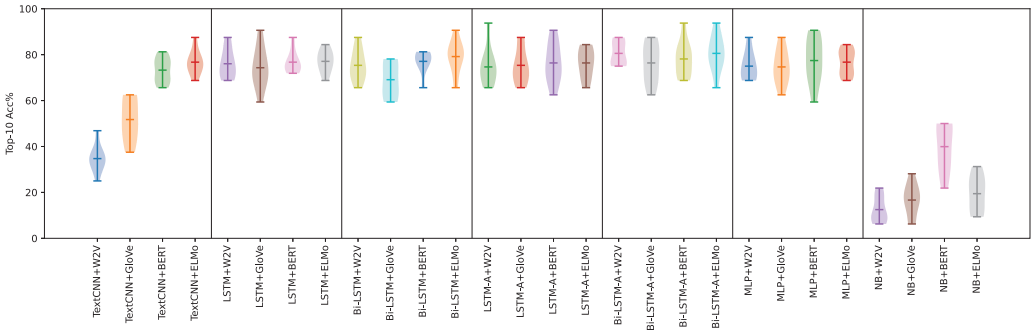
From Fig. 7, Fig. 8, Fig. 9, and Fig. 10, it can be seen that Bi-LSTM-A using ELMo and Bi-LSTM using ELMo are significantly superior to other models in most cases across all datasets in terms of top-1 accuracy and MRR. Moreover, they have large or medium effect sizes over all other models. The above two models are also significantly better than other models over *GCC*, *Firefox*, and the cross-project dataset with respect to the top-5 and top-10 accuracy. Likewise, they also have large or medium effect sizes over other models. This shows that both



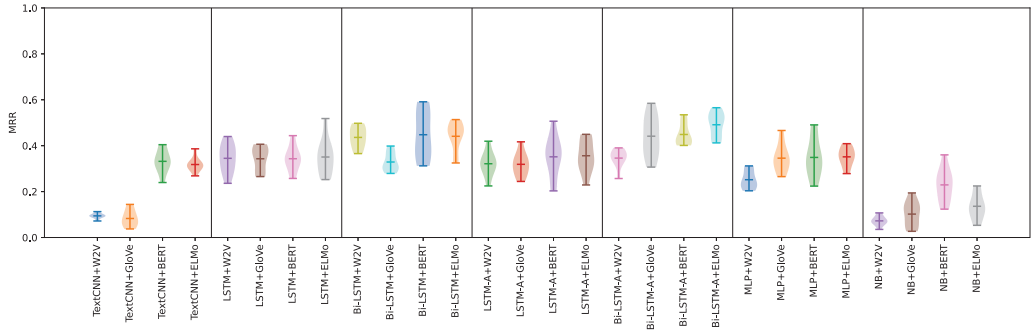
(a) Top-1 accuracy



(b) Top-5 accuracy



(c) Top-10 accuracy

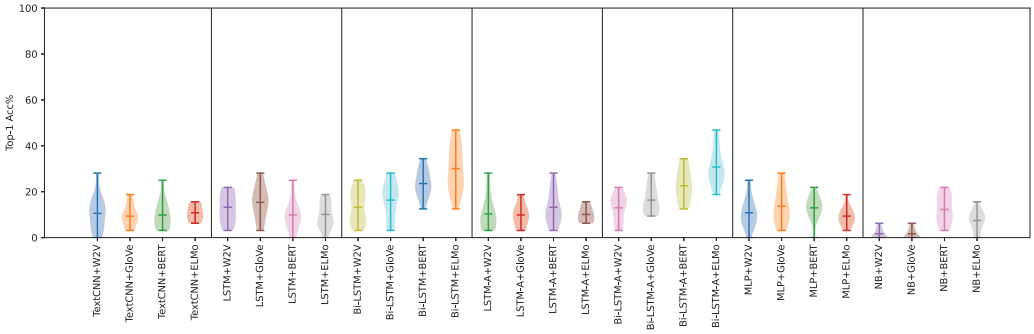


(d) MRR

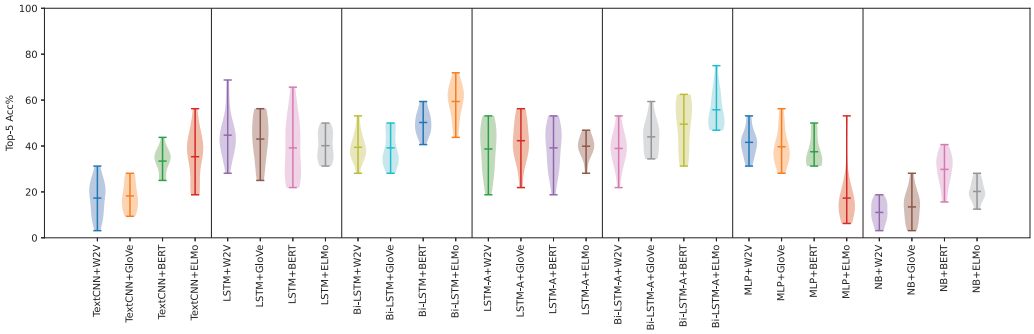
Fig. 3. The Top-k (k = 1, 5, 10) accuracy and MRR over *Eclipse JDT*.

Bi-LSTM-A+ELMo and Bi-LSTM+ELMo are more suitable for solving the bug assignment task than other investigated models. In particular, compared with existing text classification-based bug assignment methods (Lee et al., 2017; Zaidi et al., 2020) using Word2Vec and TextCNN, Bi-LSTM-A+ELMo and Bi-LSTM+ELMo have over 30% improvement in top-k (k = 1, 5, 10) accuracy across all subjects.

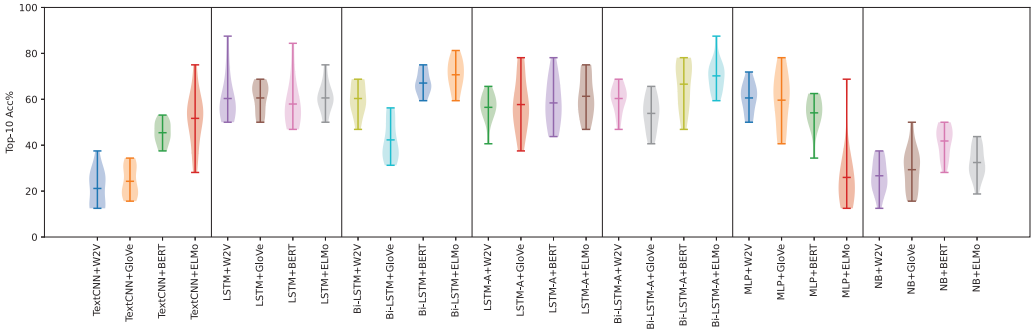
Despite Bi-LSTM-A+ELMo using the attention mechanism, there is no significant difference between it and Bi-LSTM+ELMo. The likely reason is that the attention mechanism could concern the features that are not closely relevant to bug fixers' prediction limited by the quality of bug reports. Compared with other classification models, Bi-LSTM and Bi-LSTM with attention can capture more important features including



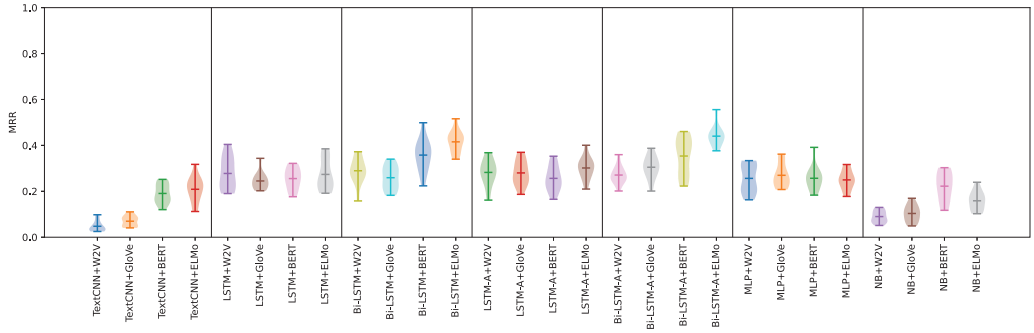
(a) Top-1 accuracy



(b) Top-5 accuracy

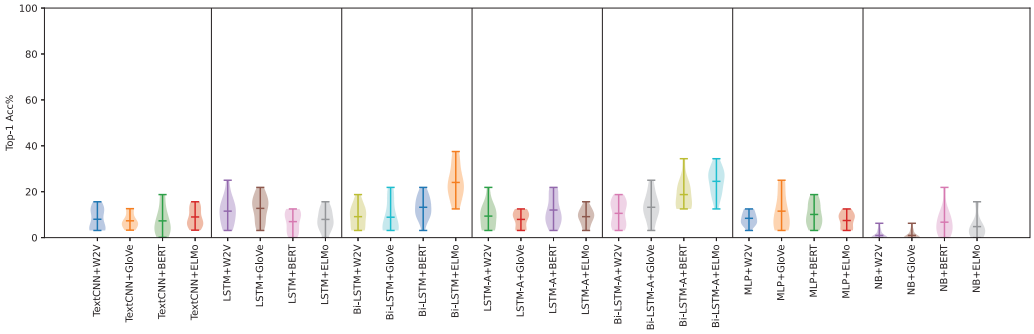


(c) Top-10 accuracy

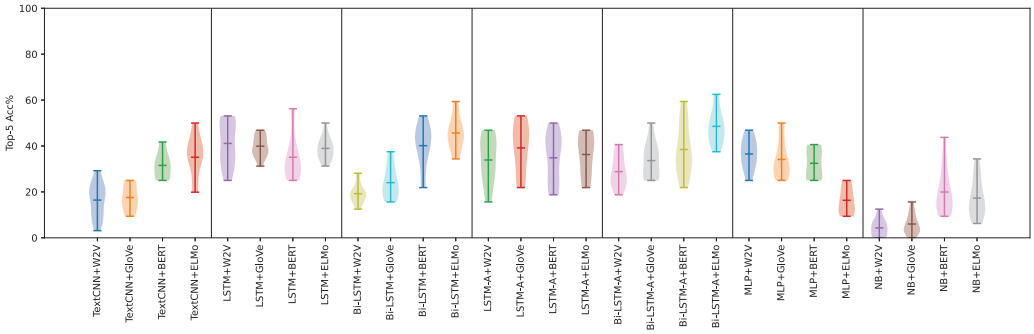


(d) MRR

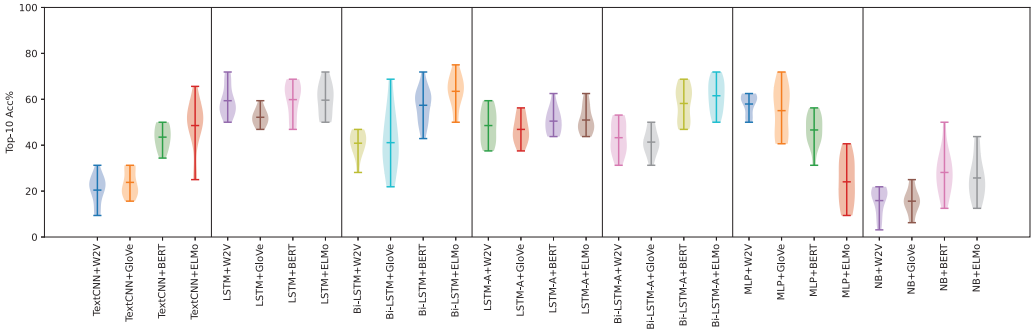
Fig. 4. The Top-k (k = 1, 5, 10) accuracy and MRR over *GCC*.



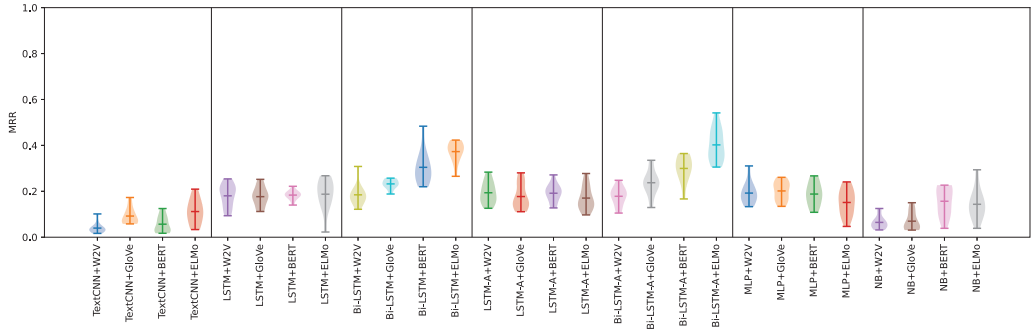
(a) Top-1 accuracy



(b) Top-5 accuracy

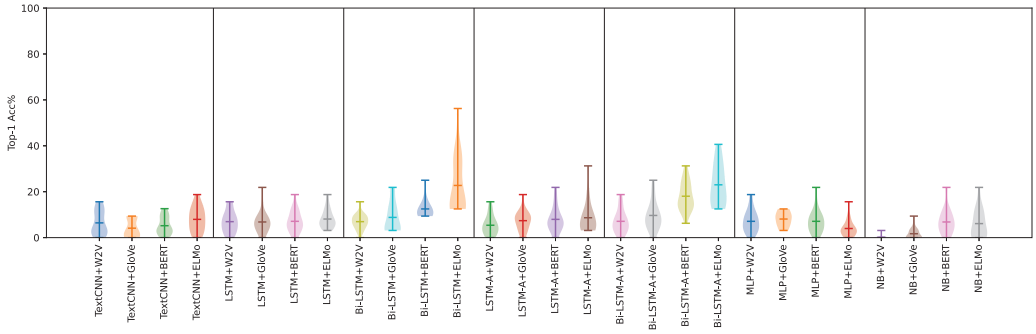


(c) Top-10 accuracy

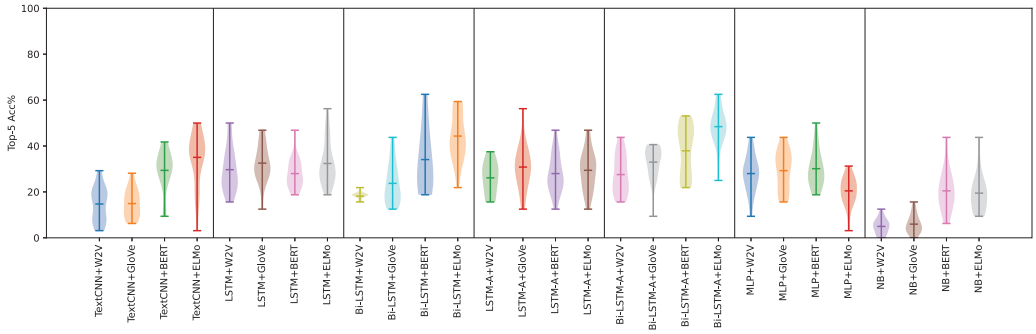


(d) MRR

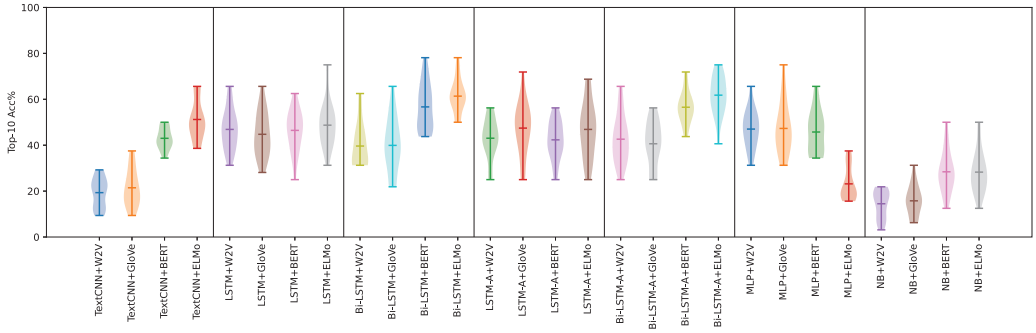
Fig. 5. The Top-k (k = 1, 5, 10) accuracy and MRR over *Firefox*.



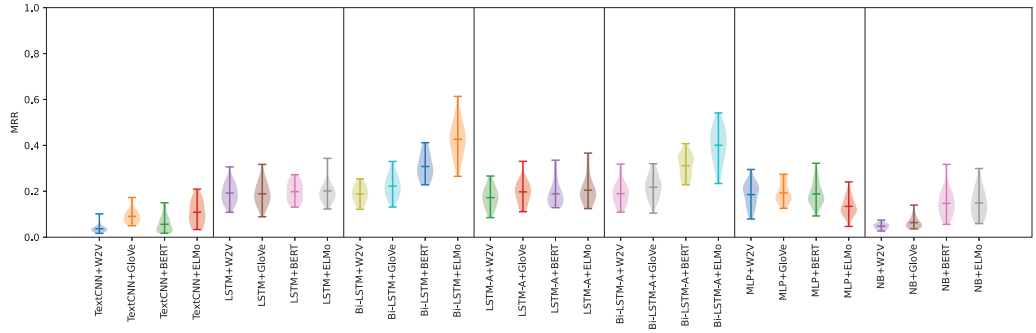
(a) Top-1 accuracy



(b) Top-5 accuracy

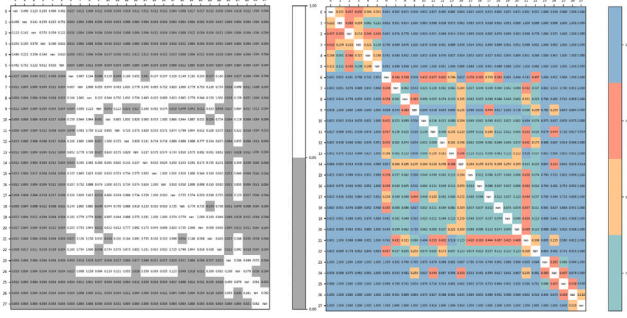


(c) Top-10 accuracy

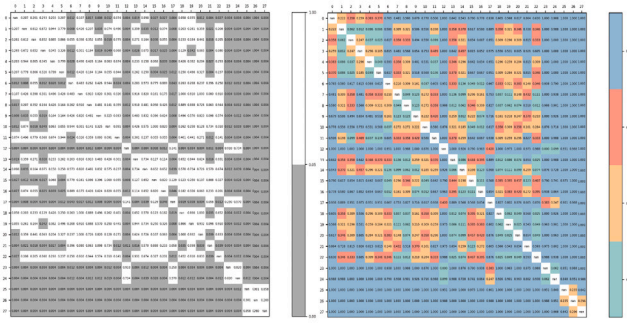


(d) MRR

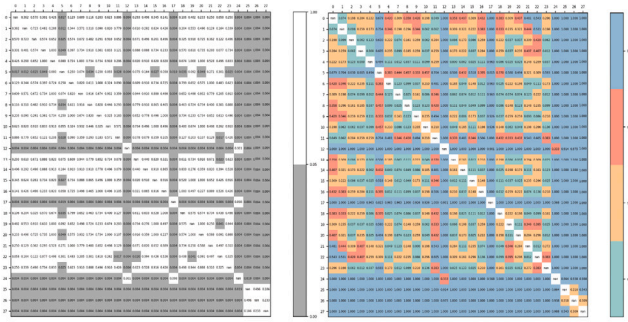
Fig. 6. The Top-k (k = 1, 5, 10) accuracy and MRR over a *cross-project* dataset.



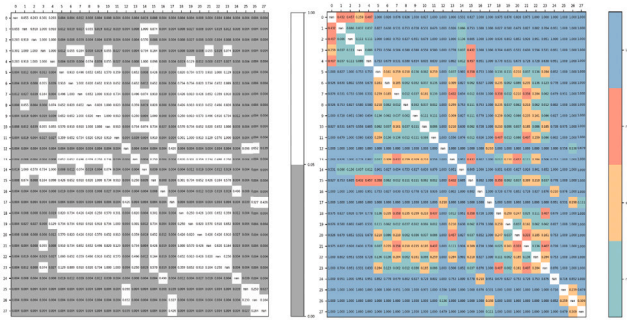
(a) Top-1 accuracy



(b) Top-5 accuracy

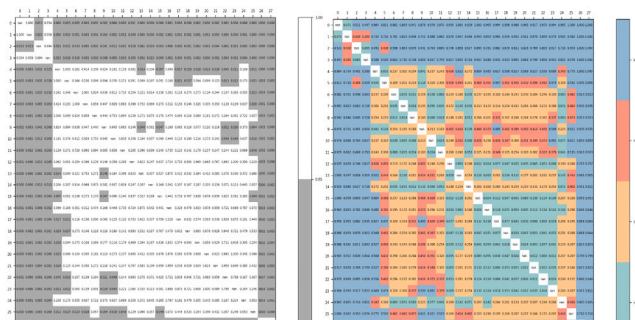


(c) Top-10 accuracy

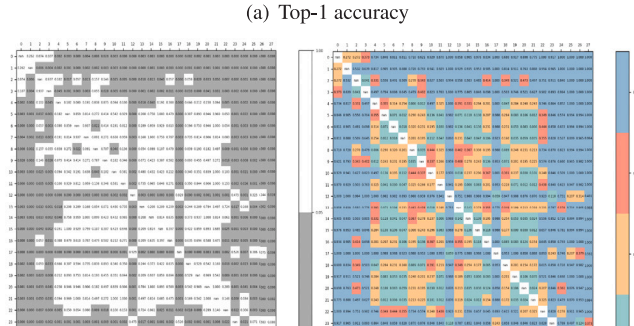


(d) MRR

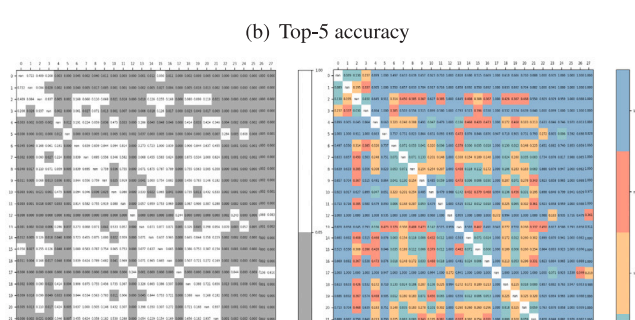
Fig. 7. The statistical and effect size results between all investigated models on *Eclipse JDT*.



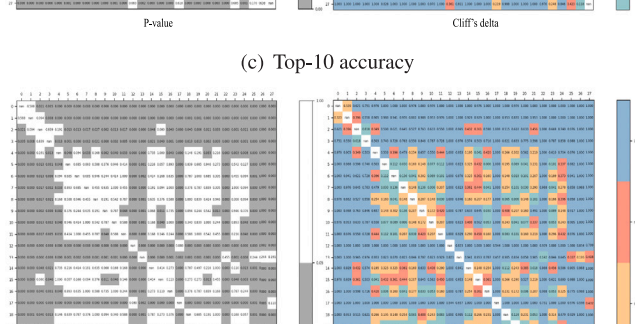
(a) Top-1 accuracy



(b) Top-5 accuracy

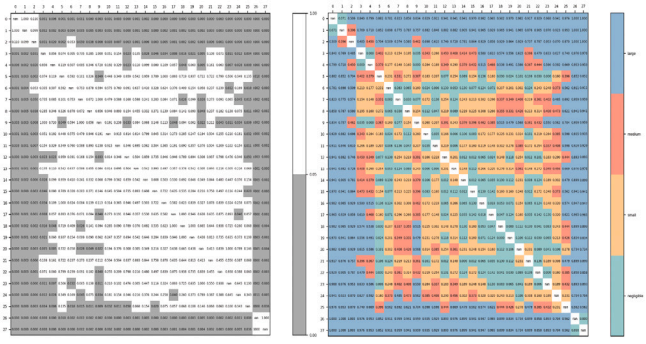


(c) Top-10 accuracy

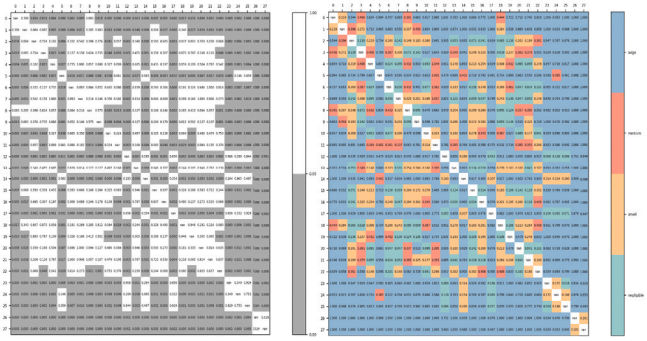


(d) MRR

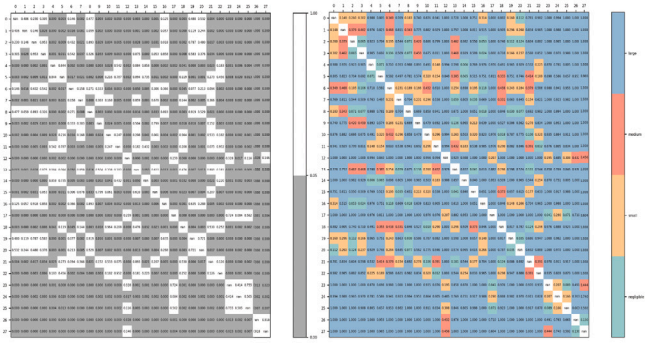
Fig. 8. The statistical and effect size results between all investigated models on *GCC*.



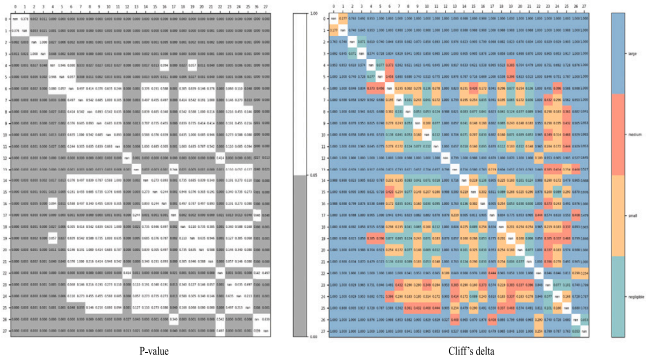
(a) Top-1 accuracy



(b) Top-5 accuracy

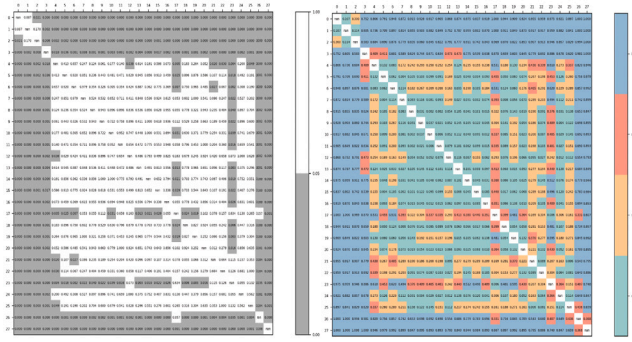


(c) Top-10 accuracy

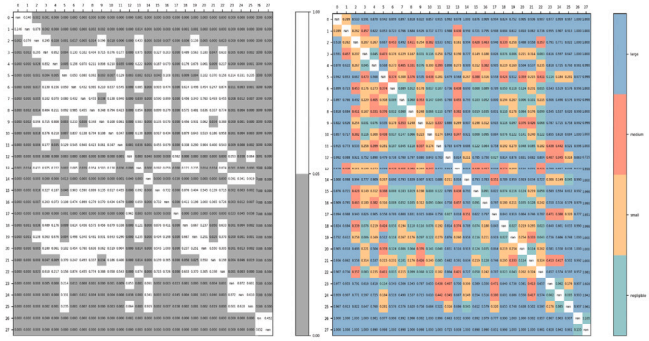


(d) MRR

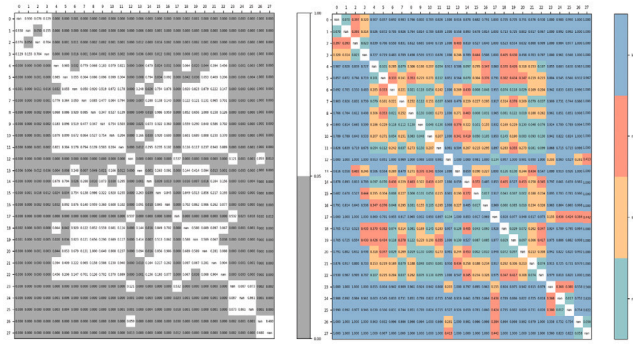
Fig. 9. The statistical and effect size results between all investigated models on *Firefox*.



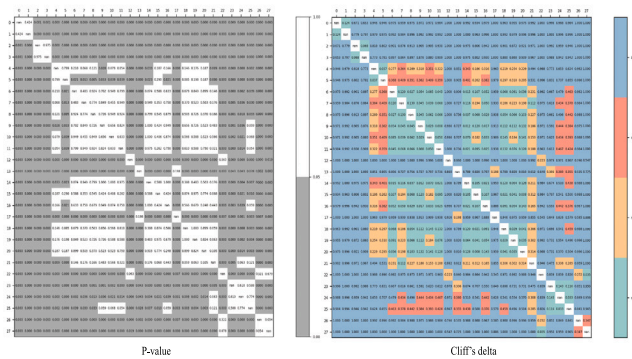
(a) Top-1 accuracy



(b) Top-5 accuracy



(c) Top-10 accuracy



(d) MRR

Fig. 10. The statistical and effect size results between all investigated models on a *cross-project* dataset.

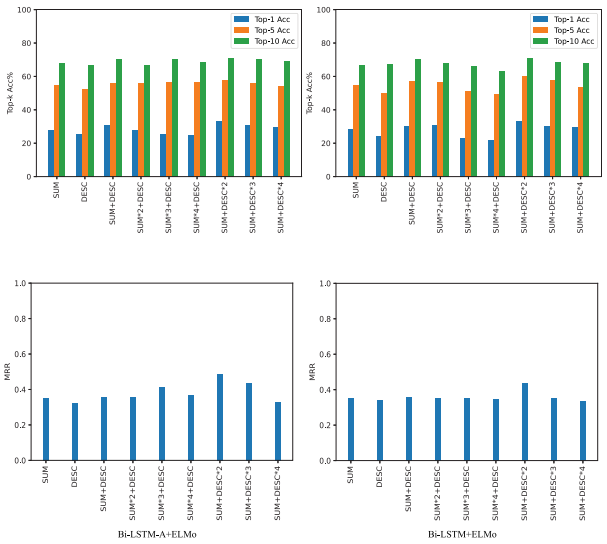


Fig. 11. The effect of the training text composed of different proportions of the summary and description on Bi-LSTM-A+ELMo and Bi-LSTM+ELMo.

the preceding and succeeding context information from bug reports. This could be why they can get better performance.

4.2. RQ2: Is the description useful for bug assignment? what is the optimal weight between the summary and description?

Motivation: Zhou et al. (2016) claimed that the bug report's description is considered valueless due to its lengthy and scattered nature. Therefore, they only utilized the summary because it is concise and exhibits strong semantic connections between words (Ko et al., 2006; Lamkanfi et al., 2010). In comparison with the summary, the description contains additional valuable information related to the bug. Several existing studies (Matter et al., 2009; Zhou et al., 2016) only used the description or summary. Although most studies (Murphy and Cubranic, 2004; Bhattacharya et al., 2012; Jeong et al., 2009) considered both the summary and description, they were treated equally. The summary and description in the bug assignment are different. Thus, it could be unfair to treat them equally. This prompted us to further explore RQ2.

Method: We designed nine strategies representing the summaries and descriptions with different weights. SUM and DESC represent the use of only summary and description, respectively. SUM+DESC denotes the use of the summary and description. SUM+DESC*n represents the use of the summary and n replications of the description. On the contrary, SUM*n+DESC expresses the use of the description and n replications of the summary. According to the nine strategies, we conducted an empirical investigation of their influences on bug assignment. Given that Bi-LSTM-A+ELMo and Bi-LSTM+ELMo outperform other models in terms of top-k accuracy, we only focused on the effects of the nine strategies on the two models. We excluded the *Eclipse JDT* and *Firefox* as their bug reports only contain the descriptions without summaries. The results are shown in Fig. 11, where the x-axis represents different strategies and the y-axis represents the top-k accuracy or MRR values.

Findings: Both the summary and description of the bug report are useful for bug assignment, but the description is more valuable than the summary. The optimal weight between them is 1:2.

From Fig. 11, it can be seen that the SUM+DESC strategy has higher top-k (k = 1, 5, 10) accuracy and MRR values than the SUM and DESC. This means that both the summary and description of the bug report are useful for bug assignments. The SUM+DESC strategy provides more comprehensive and richer information to the model, thereby enhancing top-k accuracy and MRR values. Furthermore, the SUM+DESC*2 strategy always has higher top-k (k = 1, 5, 10) and MRR values than the other eight strategies. In other words, the optimal proportion between the summary and the description is 1:2. This means that the summary and description cannot be treated with the same weights.

Additionally, we found a very interesting phenomenon, *i.e.*, with the increase of the number of description copies, the top-k accuracy and MRR values do not rise, but fall. There are two likely reasons. One is that as the number of description copies increases, the features extracted from the summary become less effective. The other is that the increase of description copies generates more redundant training data, leading to the accumulation of noisy data. This could make the model more susceptible to the influence of noise data, resulting in a decrease in top-k accuracy and MRR. Additionally, the increase of description copies can improve the model's top-k accuracy on the training set but paradoxically lead to lower performance on the test set. Therefore, the increase of description copies may cause the over-fitting of the model, ultimately reducing its generalization ability.

4.3. RQ3: To what extent does the training corpus influence the accuracy of the representation learning model for bug assignment tasks?

Motivation: Various pre-trained models trained on specific domains, such as SciBERT (Beltagy et al., 2019) and BioBERT (Lee et al., 2020), have emerged since the introduction of the generative pre-training (GPT). SciBERT was trained on over one million documents, 82% of which originated from the biology domain and the rest from the computer science domain. Similarly, BioBERT is also a pre-trained model specifically trained on biomedical domain data. A prior study (Von der Mosel et al., 2022) has demonstrated that a domain-specific pre-trained model can lead to performance improvement compared to a model trained on a general domain corpus. However, the impact of the training corpus difference between the two different domains on the improvements is relatively small. This raises the question of whether such a conclusion still holds in the context of bug assignment. Furthermore, if the conclusion is still valid, to what extent does the training corpus influence the accuracy of representation learning models? This drives us to set up RQ3.

Method: To answer RQ3, we selected a pre-trained model, *i.e.*, NextBug (Du et al., 2022) trained based on a dataset containing four hundred thousand bug reports. It is the first model trained based on a bug domain corpus. The bug assignment task aims at extracting information and predicting bug fixers from bug reports. In this sense, the NextBug model is suitable for bug assignment tasks. Since Bi-LSTM+ELMo and Bi-LSTM-A+ELMo are better than other models, we only compared the performance of ELMo and NextBug using Bi-LSTM and Bi-LSTM-A on four datasets. The experimental results are shown in Fig. 12, where the x-axis and y-axis of each violin chart represent a deep learning model and the top-k accuracy or MRR, respectively.

Findings: The training corpus of the representation learning model has a significant impact on bug assignment tasks. The model using the specialized-domain corpus has larger or medium effect sizes over the model using the general-domain corpus in bug assignments with respect to top-1 accuracy.

From Fig. 12, it can be seen that the classification models using NextBug have higher median top-k and MRR values than the same classification models using ELMo on the four datasets. The top-k and MRR values on the cross-project dataset are closer to the ones on Firefox because the project contains many more samples than the other two projects. To qualitatively analyze the difference between the two models, we conducted a Wilcoxon signed-rank test. The Cliff's Delta test was also executed for quantitative analysis of the magnitude of the difference. The results are shown in Fig. 13, where the corresponding relation between the flags from 0 to 3 and the models is shown in Table 4.

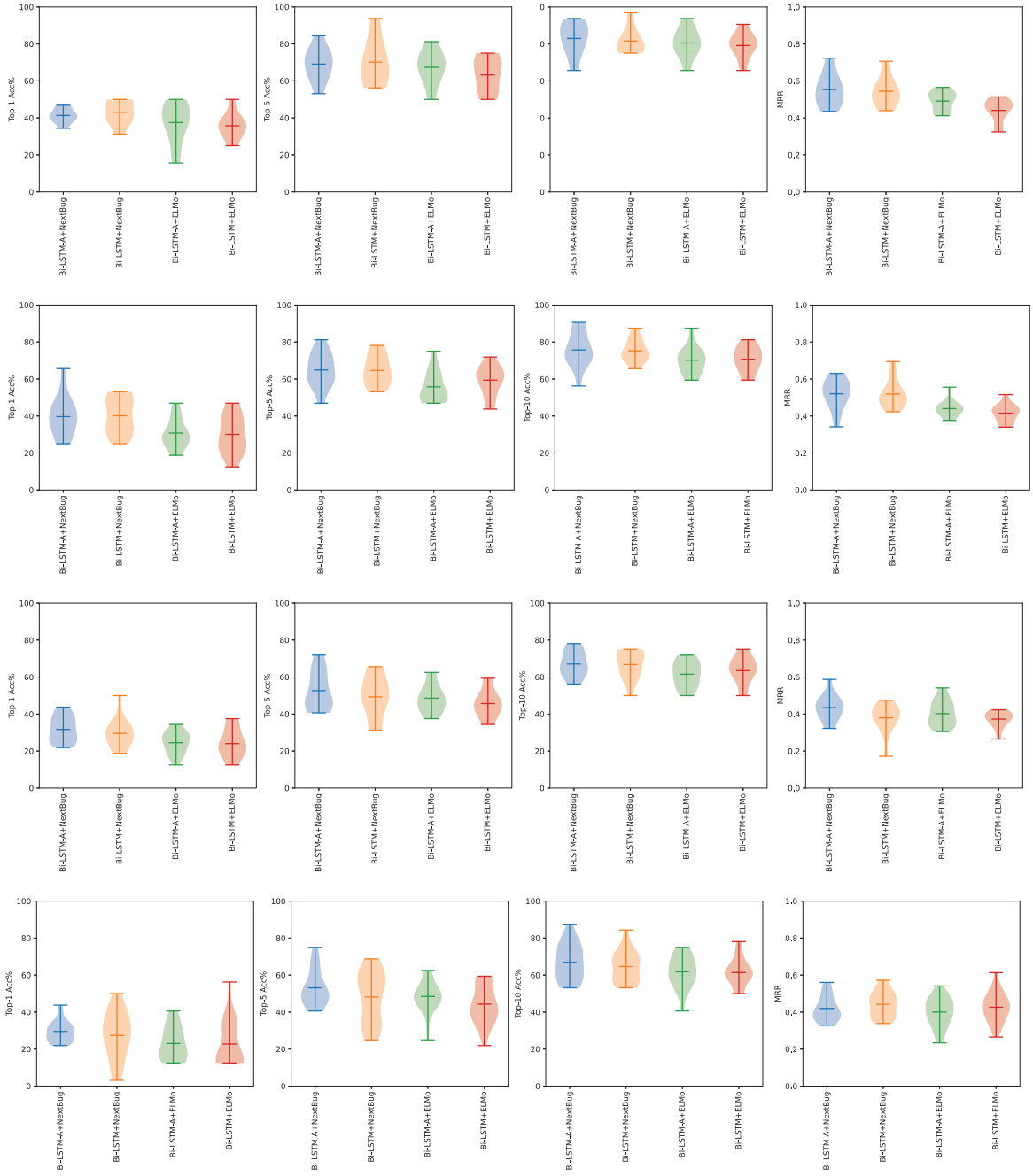
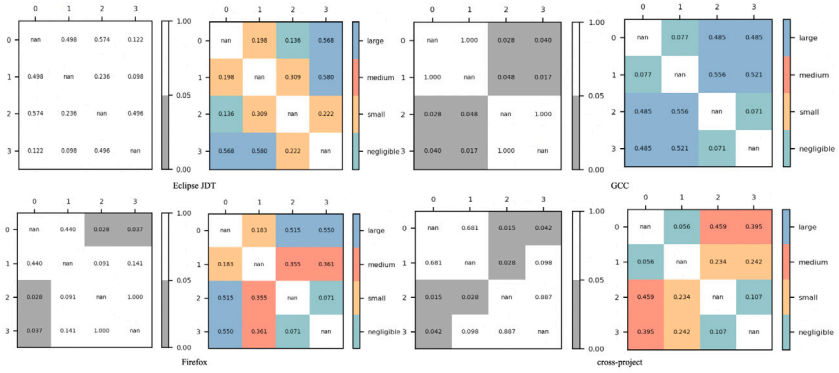


Fig. 12. The top-k(k = 1, 5, 10) accuracy and MRR of ELMo and NextBug with Bi-LSTM and Bi-LSTM with attention over *Eclipse JDT*, *GCC*, *Firefox*, and *cross- project* datasets .

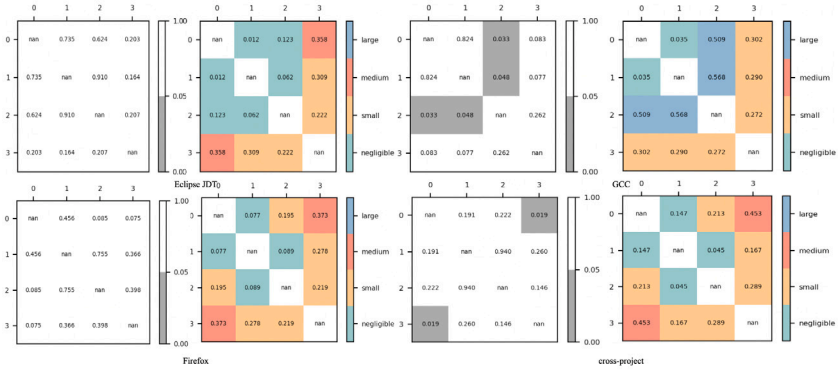
Table 4
Corresponding relation of the studied models and the flags in Fig. 13.

No.	Model
0	Bi-LSTM-A+NextBug
1	Bi-LSTM+NextBug
2	Bi-LSTM-A+ELMo
3	Bi-LSTM+ELMo

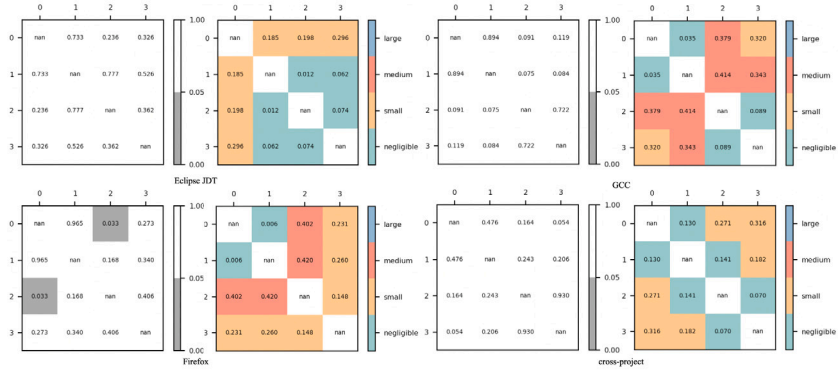
From Fig. 13, it can be seen that the classifications using NextBug are significantly better than the models with the same classification models using ELMo with respect to top-1 in most cases. Moreover, the former ones have large or medium effect sizes over the latter ones. With the exception of the top-1 accuracy, there is no significant difference between the NextBug and ELMo in most cases. Moreover, the former only has a negligible or small effect size over the latter. In comparison with other metrics, the top-1 metric is more useful. In this sense, compared with the general domain training corpus, the domain-specific training corpus related to the solving task can significantly improve the accuracy of bug assignments. Although Bi-LSTM-A+NextBug uses the attention mechanism, there is no significant difference between Bi-LSTM-A+NextBug and Bi-LSTM+NextBug in terms of top-k (k = 1, 5, 10) accuracy and MRR. Moreover, the former only has a negligible or small effect size over the latter.



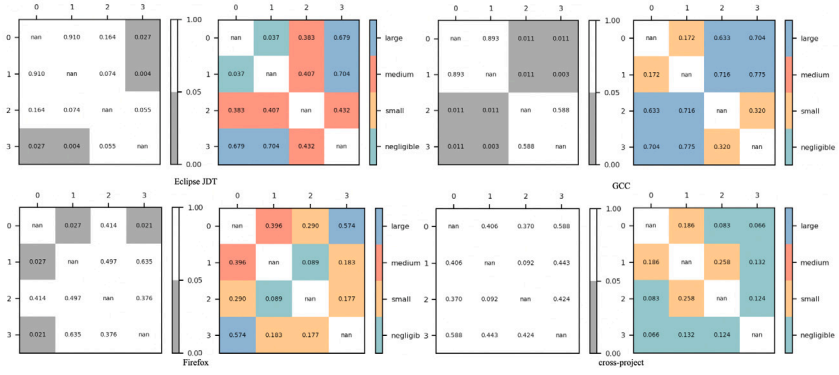
(a) top-1 accuracy



(b) top-5 accuracy



(c) top-10 accuracy



(d) MRR

Fig. 13. The statistical and effect size results between NextBug and ELMo on *Eclipse JDT*, *GCC*, *Firefox*, and *cross-project* datasets.

5. Threats to validity

Despite careful experiment design, it is important to acknowledge the potential threats to the validity of this study. These threats can be summarized as follows.

5.1. Internal validity

The random split of the datasets can cause fluctuations in the performance of a model, leading to generating different results. This is because when the training set contains the class with few samples, the model cannot well learn the features corresponding to the class, resulting in inaccurate predictions. To alleviate the impact of random data set partitioning, we employed ten-fold cross-validation, which involves training and validating the model on all data subsets in a rotating way.

Another internal validity arises from the influence of parameters in all studied models. The performance of different models can vary depending on the chosen parameters. To ensure a fair and objective comparison, on the one hand, we conducted parameter experiments to obtain the optimal parameters. On the other hand, we also referred to the related works for selecting appropriate parameters.

5.2. External validity

Deep learning-based bug assignment methods depend on bug reports. Limited by the expertise of bug submitter, the quality of bug reports can vary significantly. Low-quality bug reports can cause the performance degeneration of a model. To mitigate the impact of bug report quality, we conducted the empirical study using three widely used datasets in the field of bug assignment, which are extracted from four popular and mature open-source software projects, and a cross-project dataset. We will conduct an empirical study based on more open-source and closed-source projects to address this threat in future work.

Additionally, the data collected from the three projects may contain noise. Noisy data can affect the performance of the bug assignment method. Consequently, we preprocessed the data by removing missing data, URLs, non-English bug reports, and stop words to mitigate the effect of noisy data.

5.3. Construct validity

The construct validity may be affected by the use of the metric, *i.e.*, top-k accuracy. As previous studies, we also employed this metric to evaluate the impacts of different representation learning models and deep learning classifiers on bug assignment. It may not to be enough provide a comprehensive evaluation of the performance of the studied models. We will seek other appropriate metrics for the comprehensive evaluation of all studied models to alleviate this threat. Since top-k accuracy only focuses on the top-k recommended bug fixers and does not report the specific position of the true assignee in the recommendation list. The MRR was also employed to address this limitation.

6. Conclusions

Deep learning-based bug assignment methods have achieved promising performance. Several deep learning models have been increasingly proposed. No previous studies empirically evaluate the effects of different deep learning models on bug assignment. In this context, we conducted an empirical study of evaluating the impacts of 35 deep learning models based on five representation models, namely Word2Vec, GloVe, NextBug, BERT, and ELMo, and seven classification models, *i.e.*, LSTM, TextCNN, LSTM with attention mechanism, Bi-LSTM, Bi-LSTM with attention mechanism, MLP, and NB on bug assignment. Three commonly used datasets, *i.e.*, *Eclipse JDT*, *GCC*, *Firefox*, and a *cross-project* dataset were used for experimental comparisons.

According to the experimental results, the following three findings can be drawn: (1) Bi-LSTM-A+ELMo and Bi-LSTM+ELMo are significantly superior to other deep learning models on bug assignment tasks in terms of top-k (k = 1, 5, 10) accuracy and MRR; (2) Both the summary and description of bug reports are useful for bug assignment, but the description should be given higher weight than the summary; (3) The choice of training corpus for representation learning models has a significant impact on bug assignment task. The model that utilizes a bug-specific training corpus has large or medium effect sizes over the model using a general domain corpus with respect to top-1 accuracy. This study not only aids in understanding the critical components of bug assignment based on deep learning, but also provides practice guidance for picking the optimal deep learning technique for addressing bug assignment tasks. Additionally, it helps identify the boundaries of problem-solving abilities for different deep learning models.

All experimental subjects come from the same bug tracking system, *i.e.*, Bugzilla. In addition, there are several popular bug tracking systems (*e.g.* JIRA and BugNet). The structure of bug reports from different bug-tracking systems could be different. For future work, we will conduct a large-scale empirical study on more projects from other bug-tracking systems. The fixed bug reports can reflect the skills or expertise of the developers. We are planning to measure the similarities of developers based on the similarities of their fixed historical bug reports. Contrastive learning will be employed to improve the bug assignment accuracy based on the measured similarity information. The Transformer model has achieved promising results in natural language processing tasks. Moreover, most studies only utilize bug reports. Unlike bug reports, the fixed code can directly reflect the expertise of bug fixers. Therefore, we will extract more features from source code and bug reports via the Transformer model to improve the performance of bug assignment further.

CRediT authorship contribution statement

Rongcun Wang: Conceptualization, Methodology, Experimental design, Writing – original draft, Supervision. **Xingyu Ji:** Writing – original draft, Experimental implementation, Data analysis, Data validation, Data visualization. **Senlei Xu:** Experimental implementation, Data analysis, Data validation. **Yuan Tian:** Investigation, Writing – review & editing. **Shujuan Jiang:** Supervision, Writing – review & editing. **Rubing Huang:** Writing – review & editing.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

Data availability

https://github.com/AI4BA/dl4ba.

Acknowledgments

The authors would like to thank the anonymous reviewers for their valuable comments and helpful suggestions. This work is partially supported by the National Natural Science Foundation of China under grant NO. 61673384, No. 61872167 and No. 61502205, partially supported by the Science and Technology Development Fund of Macau, Macau SAR under grant 0046/2021/A and 0021/2023/R1A1, and partially supported by a Faculty Research Grant of Macau University of Science and Technology under grant FRG-22-103-FIE.

References

Ahsan, S.N., Ferzund, J., Wotawa, F., 2009. Automatic software bug triage system (BTS) based on latent semantic indexing and support vector machine. In: Proceedings of the 4th International Conference on Software Engineering Advances. pp. 216–221.

Alazzam, I., Aleroud, A., Al Latifah, Z., Karabatis, G., 2020. Automatic bug triage in software systems using graph neighborhood relations for feature augmentation. IEEE Trans. Comput. Soc. Syst. 7 (5), 1288–1303.

Anvik, J., Hiew, L., Murphy, G.C., 2006. Who should fix this bug? In: Proceedings of the 28th International Conference on Software Engineering. ICSE '06, pp. 361–370.

Aung, T.W.W., Wan, Y., Huo, H., Sui, Y., 2022. Multi-triage: A multi-task learning framework for bug triage. J. Syst. Softw. 184, 111133.

Beltagy, I., Lo, K., Cohan, A., 2019. SciBERT: A pretrained language model for scientific text. In: Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing and the 9th International Joint Conference on Natural Language Processing. EMNLP-IJCNLP, pp. 3615–3620.

Bhattacharya, P., Neamtiu, I., Shelton, C.R., 2012. Automated, highly-accurate, bug assignment using machine learning and tossing graphs. J. Syst. Softw. 85 (10), 2275–2292.

Bird, S., Klein, E., Loper, E., 2009. Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit. O'Reilly Media, Inc..

Chakraborty, S., Krishna, R., Ding, Y., Ray, B., 2022. Deep learning based vulnerability detection: Are we there yet? IEEE Trans. Softw. Eng. 48 (9), 3280–3296.

Choquette-Choo, C.A., Sheldon, D., Proppe, J., Alphonso-Gibbs, J., Gupta, H., 2019. A multi-label, dual-output deep neural network for automated bug triaging. In: Proceedings of the 18th IEEE International Conference on Machine Learning and Applications. ICMLA, pp. 937–944.

Cliff, N., 1993. Dominance statistics: Ordinal analyses to answer ordinal questions. Psychol. Bull. 114 (3), 494.

Dai, W., Xue, G.-R., Yang, Q., Yu, Y., 2007. Transferring Naive Bayes classifiers for text classification. In: AAAI, Vol. 7. pp. 540–545.

Dedík, V., Rossi, B., 2016. Automated bug triaging in an industrial context. In: Proceedings of the 42th Euromicro Conference on Software Engineering and Advanced Applications. SEAA, pp. 363–367.

Du, X., Zheng, Z., Xiao, G., Zhou, Z., Trivedi, K.S., 2022. DeepSIM: Deep semantic information-based automatic mandelbug classification. IEEE Trans. Reliab. 71 (4), 1540–1554.

Frank, E., Bouckaert, R.R., 2006. Naive Bayes for text classification with unbalanced classes. In: Knowledge Discovery in Databases: PKDD 2006: 10th European Conference on Principles and Practice of Knowledge Discovery in Databases Berlin, Germany, September 18-22, 2006 Proceedings 10. Springer, pp. 503–510.

Giray, G., Bennin, K.E., Köksal, Ö., Babur, Ö., Tekinerdogan, B., 2023. On the use of deep learning in software defect prediction. J. Syst. Softw. 195, 111537.

Graves, A., 2012. Long short-term memory. In: Supervised Sequence Labelling with Recurrent Neural Networks. Springer Berlin Heidelberg, Berlin, Heidelberg, pp. 37–45.

Guo, S., Zhang, X., Yang, X., Chen, R., Guo, C., Li, H., Li, T., 2020. Developer activity motivated bug triaging: Via convolutional neural network. Neural Process. Lett. 51 (3), 2589–2606.

Hu, D., Chen, M., Wang, T., Chang, J., Yin, G., Yu, Y., Zhang, Y., 2018. Recommending similar bug reports: A novel approach using document embedding model. In: Proceedings of the 25th Asia-Pacific Software Engineering Conference. APSEC, pp. 725–726.

Hu, X., Li, G., Xia, X., Lo, D., Jin, Z., 2020. Deep code comment generation with hybrid lexical and syntactical information. Empir. Softw. Eng. 25 (3), 2179–2217.

Jahanshahi, H., Cevik, M., 2022. S-DABT: Schedule and dependency-aware bug triage in open-source bug tracking systems. Inf. Softw. Technol. 151, 107025.

Jahanshahi, H., Chhabra, K., Cevik, M., Basar, A., 2021. DABT: A dependency-aware bug triaging method. In: International Conference on Evaluation and Assessment in Software Engineering. pp. 221–230.

Jeong, G., Kim, S., Zimmermann, T., 2009. Improving bug triage with bug tossing graphs. In: Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on the Foundations of Software Engineering. pp. 111–120.

Kim, Y., 2014. Convolutional neural networks for sentence classification. In: Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing. EMNLP, Doha, Qatar, pp. 1746–1751.

Ko, A.J., Myers, B.A., Chau, D.H., 2006. A linguistic analysis of how people describe software problems. In: Visual Languages and Human-Centric Computing. VL/HCC'06, IEEE, pp. 127–134.

Lamkanfi, A., Demeyer, S., Giger, E., Goethals, B., 2010. Predicting the severity of a reported bug. In: 2010 7th IEEE Working Conference on Mining Software Repositories. MSR 2010, IEEE, pp. 1–10.

Lee, S.-R., Heo, M.-J., Lee, C.-G., Kim, M., Jeong, G., 2017. Applying deep learning based automatic bug triager to industrial projects. In: Proceedings of the 11th Joint Meeting on Foundations of Software Engineering. pp. 926–931.

Lee, D.-G., Seo, Y.-S., 2019. Systematic review of bug report processing techniques to improve software management performance. J. Inf. Process. Syst. 15 (4), 967–985.

Lee, J., Yoon, W., Kim, S., Kim, D., Kim, S., So, C.H., Kang, J., 2020. BioBERT: A pre-trained biomedical language representation model for biomedical text mining. Bioinformatics 36 (4), 1234–1240.

Mani, S., Sankaran, A., Aralikatte, R., 2019. 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.

Matter, D., Kuhn, A., Nierstrasz, O., 2009. Assigning bug reports using a vocabulary-based expertise model of developers. In: Proceedings of the 6th International Working Conference on Mining Software Repositories. pp. 131–140.

Metsis, V., Androutsopoulos, I., Paliouras, G., 2006. Spam filtering with naive Bayes-which Naive Bayes? In: CEAS, Vol. 17. Mountain View, CA, pp. 28–69.

Mikolov, T., Chen, K., Corrado, G., Dean, J., 2013. Efficient estimation of word representations in vector space. arXiv preprint arXiv:1301.3781.

Murphy, G., Cubranic, D., 2004. Automatic bug triage using text categorization. In: Proceedings of the 6th International Conference on Software Engineering & Knowledge Engineering. Citeseer, pp. 1–6.

Naguib, H., Narayan, N., Brügge, B., Helal, D., 2013. Bug report assignee recommendation using activity profiles. In: Proceedings of the 10th Working Conference on Mining Software Repositories. MSR, pp. 22–30.

Nowak, J., Taspinar, A., Scherer, R., 2017. LSTM recurrent neural networks for short text and sentiment classification. In: Artificial Intelligence and Soft Computing. Springer International Publishing, Cham, pp. 553–562.

Romano, J., Kromrey, J.D., Coraggio, J., Skowronek, J., 2006. Appropriate statistics for ordinal level data: Should we really be using t-test and Cohen'sd for evaluating group differences on the NSSE and other surveys. In: The Annual Meeting of the Florida Association of Institutional Research. pp. 1–31.

Rusland, N.F., Wahid, N., Kasim, S., Hafit, H., 2017. Analysis of Naïve Bayes algorithm for email spam filtering across multiple datasets. In: IOP Conference Series: Materials Science and Engineering, vol. 226, (no. 1), IOP Publishing, 012091.

Sajedi-Badashian, A., Stroulia, E., 2020. Vocabulary and time based bug-assignment: A recommender system for open-source projects. Softw. - Pract. Exp. 50 (8), 1539–1564.

Sarkar, A., Rigby, P.C., Bartalos, B., 2019. Improving bug triaging with high confidence predictions at ericsson. In: 2019 IEEE International Conference on Software Maintenance and Evolution. ICSME, pp. 81–91.

Sawarkar, R., Nagwani, N.K., Kumar, S., 2019. Predicting available expert developer for newly reported bugs using machine learning algorithms. In: Proceedings of the 5th International Conference for Convergence in Technology. I2CT, pp. 1–4.

Sbih, A., Akour, M., 2018. Towards efficient ensemble method for bug triaging. J. Mult.-Valued Logic Soft Comput. 31, 567–590.

Sun, X., Yang, H., Xia, X., Li, B., 2017. Enhancing developer recommendation with supplementary information via mining historical commits. J. Syst. Softw. 134, 355–368.

Tan, S., Cheng, X., Wang, Y., Xu, H., 2009. Adapting Naive Bayes to domain adaptation for sentiment analysis. In: Advances in Information Retrieval: 31th European Conference on IR Research, ECIR 2009, Toulouse, France, April 6-9, 2009. Proceedings 31. Springer, pp. 337–349.

Taud, H., Mas, J., 2018. Multilayer perceptron (MLP). In: Camacho Olmedo, M.T., Paegelow, M., Mas, J.-F., Escobar, F. (Eds.), Geomatic Approaches for Modeling Land Change Scenarios. Springer International Publishing, Cham, pp. 451–455.

Von der Mosel, J., Trautsch, A., Herbold, S., 2022. On the validity of pre-trained transformers for natural language processing in the software engineering domain. IEEE Trans. Softw. Eng. 49 (4), 1487–1507.

Voorhees, E.M., et al., 1999. The trec-8 question answering track report. In: Trec, Vol. 99. pp. 77–82.

Wilcoxon, F., 1946. Individual comparisons by ranking methods. Biometrics 1 (6), 80–83.

Wongkar, M., Angdresey, A., 2019. Sentiment analysis using Naive Bayes algorithm of the data crawler: Twitter. In: 2019 Fourth International Conference on Informatics and Computing. ICIC, IEEE, pp. 1–5.

Wu, W., Zhang, W., Yang, Y., Wang, Q., 2011. DREX: Developer recommendation with K-nearest-neighbor search and expertise ranking. In: Proceedings of the 18th Asia-Pacific Software Engineering Conference. pp. 389–396.

Xi, S., Yao, Y., Xiao, X., Xu, F., Lu, J., 2018. An effective approach for routing the bug reports to the right fixers. In: Proceedings of the 10th Asia-Pacific Symposium on Internetware. Internetware '18.

Xia, X., Lo, D., Ding, Y., Al-Kofahi, J., Nguyen, T., Wang, X., 2017. Improving automated bug triaging with specialized topic model. IEEE Trans. Softw. Eng. 43 (3), 272–297.

Xia, X., Lo, D., Wang, X., Zhou, B., 2015. Dual analysis for recommending developers to resolve bugs. J. Softw.: Evol. Process 27 (3), 195–220.

Xu, S., Li, Y., Wang, Z., 2017. Bayesian multinomial naïve Bayes classifier to text classification. In: Advanced Multimedia and Ubiquitous Engineering: MUE/FutureTech 2017 11. Springer, pp. 347–352.

Xuan, J., Jiang, H., Hu, Y., Ren, Z., Zou, W., Luo, Z., Wu, X., 2015. Towards effective bug triage with software data reduction techniques. IEEE Trans. Knowl. Data Eng. 27 (1), 264–280.

Yadav, A., Singh, S.K., 2020. A novel and improved developer rank algorithm for bug assignment. Int. J. Intell. Syst. Technol. Appl. 19 (1), 78–101.

Yang, G., Zhang, T., Lee, B., 2014. Towards semi-automatic bug triage and severity prediction based on topic model and multi-feature of bug reports. In: Proceedings of the 38th Annual Computer Software and Applications Conference. pp. 97–106.

Yin, Y., Dong, X., Xu, T., 2018. Rapid and efficient bug assignment using ELM for IOT software. IEEE Access 6, 52713–52724.

Zaidi, S.F.A., Awan, F.M., Lee, M., Woo, H., Lee, C.-G., 2020. Applying convolutional neural networks with different word representation techniques to recommend bug fixers. IEEE Access 8, 213729–213747.

Zhang, T., Chen, J., Jiang, H., Luo, X., Xia, X., 2017. Bug report enrichment with application of automated fixer recommendation. In: 2017 IEEE/ACM 25th International Conference on Program Comprehension. ICPC, pp. 230–240.

Zhang, Y., Wallace, B.C., 2017. A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification. In: Proceedings of the 8th International Joint Conference on Natural Language Processing (Volume 1: Long Papers). pp. 253–263.

Zhang, J., Wang, X., Hao, D., Xie, B., Zhang, L., Mei, H., 2015. A survey on bug-report analysis. Sci. China Inf. Sci. 58 (2), 1–24.

Zhou, Y., Tong, Y., Gu, R., Gall, H., 2016. Combining text mining and data mining for bug report classification. J. Softw.: Evol. Process 28 (3), 150–176.