IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 30 November 2022, accepted 12 December 2022, date of publication 15 December 2022,
date of current version 21 December 2022.

Digital Object Identifier 10.1109/ACCESS.2022.3229426

RESEARCH ARTICLE

Automatic Component Prediction for Issue Reports Using Fine-Tuned Pretrained Language Models

DAE-SUNG WANG AND CHAN-GUN LEE
Department of Computer Science and Engineering, Chung-Ang University, Seoul 06974, South Korea
Corresponding author: Chan-Gun Lee (cglee@cau.ac.kr)

This work was supported by the National Research Foundation of Korea (NRF) Grant through the Korea Government [Ministry of Science, ICT & Future Planning (MSIP)] under Grant 2022R1F1A1074031.

ABSTRACT Various issues or bugs are reported during the software development. It takes considerable effort, time, and cost for the software developers to triage these issues manually. Many previous studies have proposed various method to automate the triage process by predicting component using word-based language models. However, these methods still suffer from unsatisfactory performance due to their structural limitations and ignorance of the word context. In this paper, we propose a novel technique based on pretrained language models and it aims to predict a component of an issue report. Our approach fine-tunes the pretrained language models to conduct multilabel classifications. The proposed approach outperforms the previous state-of-the-art method by more than 30% with respect to the recall at k on all the datasets considered in our experiment. This improvement suggests that fine-tuned pretrained language models can help us to predict issue components effectively.

INDEX TERMS Component recommendation, machine learning, natural language processing, pretrained language model, software engineering.

I. INTRODUCTION

Numerous software issues are reported daily during the testing and maintenance phases. As the size and complexity of software have recently increased, the number of issues tends to increase, and the need to manage them promptly has become urgent. Typical modern software development processes rely on issue tracking systems, such as Jira[1] or Bugzilla,[2] to manage the issues systematically.

Nevertheless, it is still challenging for the human triagers to handle the issues everyday due to their complexity and volume. An issue report contains various information, such as title/summary, description, reporter, product, component, priority, severity, and other details. Fig. 1 presents an example of an issue report highlighting the title (e.g., "PDF search should default to unlimited white space"), description (e.g., "PDF often include additional spaces..."), and component (e.g., "PDF Viewer").

The component is essential information for the software engineer or developer to determine the locality of the issue or bug. It is not uncommon for a typical software project to have many components, which can grow due to project evolution over time [7]. The manual triager (manager) follows a triage process to assign the issue reports, as illustrated in Fig. 2. In case the triager fails to assign a component properly, the triager should repeat the steps of the process. Therefore, the triager is expected to have deep and extensive knowledge of the projects, software modules, and code bases [7], [19]. However, such an expectation becomes challenging when the scope and size of the project ever grows, which is common for most software.

Moreover, this process is laborious and requires considerable time, cost, and effort, reducing the software quality. In the Eclipse project, an example of a real industrial

The associate editor coordinating the review of this manuscript and approving it for publication was Francisco J. Garcia-Penalvo.

[1]https://www.atlassian.com/software/jira
[2]https://www.bugzilla.org/

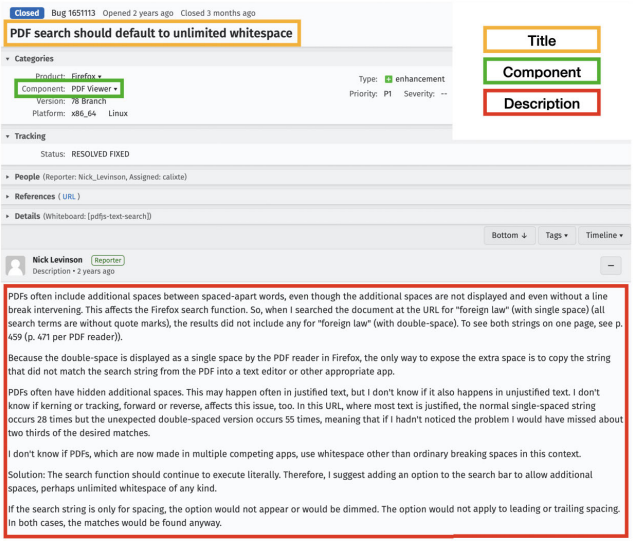

FIGURE 1. The example of an issue report from Mozilla Firefox project in Bugzilla repository.

environment, approximately 25% of the issue reports were reassigned due to incorrect triage, including component mismatch [2]. Incorrect triage is time-consuming and costly at nearly two person-hours per day in triaging issue reports [1]. Thus, from a software engineering perspective, the time-consuming and costly triage process should be improved and requires automation with proper tool support.

Recently, Choetkiertikul et al. [7] proposed a long short-term memory (LSTM)-based architecture [16], a deep learning model for software component recommendation called DeepSoft-C, which learns semantic features from the textual description of issue reports for component recommendation [7]. Cosine similarity [33] was applied for information retrieval to extract sets of textual similarity features from the description in the issue reports. By conducting semantic and similarity features, a single set of features is generated and input into a multilabel neural network to identify the components relevant to the issue report. They also demonstrated the importance of data quality by conducting experiments using a regenerated dataset based on the number of issues for each component.

These approaches suggest the possibility of predicting components using the deep learning method with the title and description data from the issue report. Unfortunately, they still have some problems due to their structural limitations. The word-based LSTM models do not use the full sentence information of the issue report, so these methods fail to exploit information useful for predicting components. In addition, the problem of long-term dependencies in the LSTM method can make the learning inefficient [3]. Issue reports may contain long sequences of words and various information. Thus, these two problems should be addressed to enable better prediction performance.

As suggested in several pieces of literature [7], [41], component information is considered very useful in the bug triage process. However, few studies have assessed component

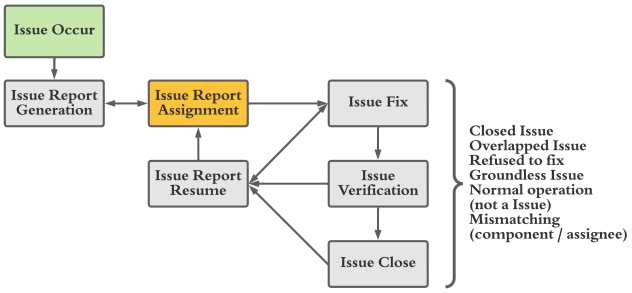

FIGURE 2. Triage process for issue reports.

prediction for issue reports compared to studies on developer prediction. In addition, we argue that recent advances in deep learning approaches have not been actively applied to the component prediction problem. Previous work [7] using the word-based LSTM method demonstrates the possibility that deep learning–based approaches can effectively predict components of an issue report, but the reported performance indicates that further improvement is still needed. This study was motivated by the scarcity of component prediction research based on recent language models and the demand for more accurate prediction performance.

This paper proposes a fine-tuned pretrained model-based component prediction technique that predicts components from the issue report data. In addition, through the fine-tuned pretrained language model, we overcome the structural limitations of the LSTM-based method. Thus, we evaluate the pretrained language models by comparing the performance with the baseline paper [7] to gain confidence in the proposed fine-tuned pretrained language model. We extract the datasets from the Eclipse Foundation, Eclipse Community, and Bugzilla Firefox, a reliable open-source issue repository. Each dataset consists of almost 20,000 issues and 137 components. In addition, we implement a task-based dataset downsampling method to address the biased dataset problem.

Thus, the proposed fine-tuned pretrained language models outperform state-of-the-art methods [7]. Furthermore, the task-based dataset downsampling method demonstrates the importance of the dataset quality in the pretrained language model-based method that Gururangan et al. [14] proposed.

The contributions of this paper are as follows:

- We propose a novel approach of component prediction for issue reports using fine-tuned pretrained language model to overcome the structural limitation of the word-based model.
- We compare the proposed approach with the state-of-the-art method for predicting component using their data. The experimental results reveal that approximately 30% of improvement is achieved by the proposed approach compared to the previous method.
- We propose a task-based downsampling method to address the biased dataset problem. The experimental results shows a significant increase in the recall at k

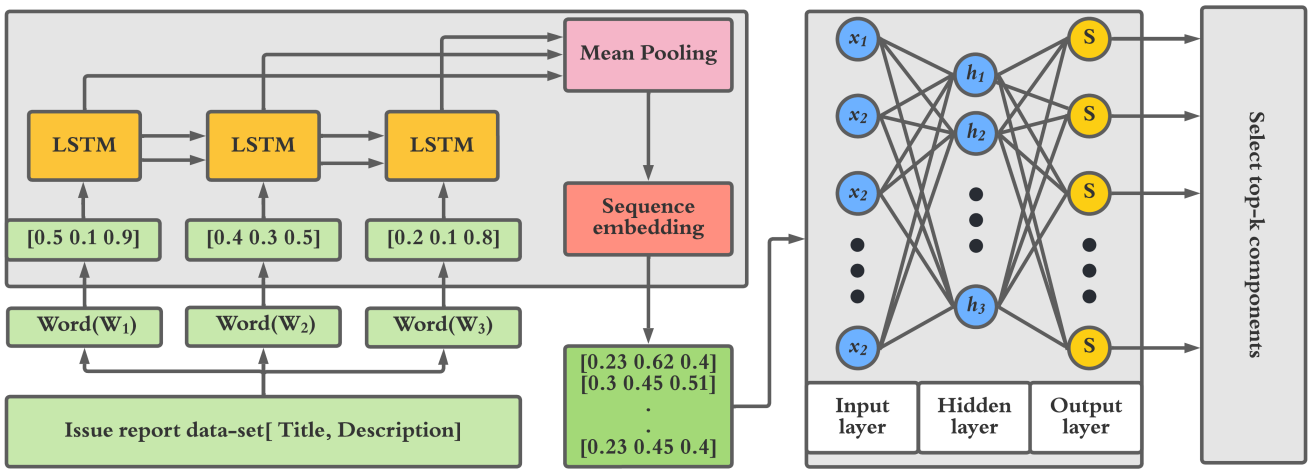

FIGURE 3. Structure of the model used in the previous study [7].

TABLE 1. Dataset statistics for previous work [7].

Repository name	Number of issues	Number of components
HBase	8,103	68
Hive	9,359	65
Infrastructure	10,166	51
Cordova	10,927	64
Hadoop	6,153	37
Fedora CloudSync	1,661	22
Islandora	1,430	67
JIRA software	23,074	142
Confluence	11,019	128
Bamboo	7,803	96
Moodle	52,338	89
Total	142,033	829

and a steady decline in validation loss after applying the proposed method.

- We build a new dataset gathered from various open-issue repositories to validate the performance of the proposed technique's task-based module. Unlike other conventional datasets targeting for bug fixer prediction only, the dataset we build consists of the components, developers, titles, and descriptions to predict the components and bug fixers. The dataset is publicly available on GitHub.[3]

The rest of the paper is organized as follows. Section II describes the related work, and Section III presents the design and implementation of the proposed method. Next, Section IV provides the experimental results and Section V explains the threats to validity of our results. Finally, Section VI explains the conclusions and future work.

II. RELATED WORK

Some researchers have proposed machine and deep learning–based methods to predict components. This section presents previous studies on component prediction and automated bug triage, which exploit the same information in the issue reports: the title and description. The difference between component prediction and bug triage is predicting the component or developer. In addition, we summarize the concepts and recent trends in pretrained language models.

A. AUTOMATIC COMPONENT PREDICTION

Sureka et al. [35] attempted to predict components using a machine learning method consisting of term frequency-inverse document frequency (TF-IDF) [31] and a component reassignment graph. The TF-IDF method can obtain content-based textual features. While experimenting, they found that the reassignment of the component causes a decrease in accuracy. Thus, they also adopted a component reassignment graph based on the changes in the issue report. Therefore, they constructed a predictive model that combines TF-IDF and a component reassignment graph to obtain the top-k results for components. In addition, they gathered the dataset from the Eclipse and Mozilla open-issue repositories consisting of approximately 20,000 reports. As in this paper, they also used the title and description information in training the model.

Yan et al. [38] proposed a discriminative probability latent semantic analysis (DPLSA) model [12] to predict components that concentrate on the topic of the issue report. They focused on which component highly corresponds to terms concerning its function in the issue report. Therefore, they constructed a semantic analysis model with issue reports as a document and the component as a category, as Lu et al. [25] proposed. The experiment achieved the top-k results from the title and description dataset, consisting of 6,000 issue reports for ten components. They achieved improved recall results at k, indicating the top-k results of components. However, their dataset consists of only ten components, fewer than the dataset we used. Therefore, the model by Yan et al. [38] demonstrates the possibility of predicting components using semantic analysis with fewer components than other datasets.

[3]https://github.com/daesungwang/Components-Prediction

TABLE 2. Summary of the selected previous work.

Task	Author	Year	Method	Selected Information	Class	Dataset
Component Prediction	Sureka [35]	2012	TF-IDF	**Title and Description**	Component	Open Issue Repositories
	Yan *et al*. [38]	2016	DPLSA	**Title and Description**	Component	Open Issue Repositories
	Choetkiertikul *et al*. [7]	2021	LSTM	**Title and Description**	Component	Open Issue Repositories
	Kangwanwisit *et al*. [20]	2022	RF	**Title and Description**	Component	Open Issue Repositories
Bug triage	Lee *et al*. [21]	2017	CNN	**Title and Description**	Developer	Industrial Issue Report
	Mani *et al*. [26]	2019	RNN	**Title and Description**	Developer	Open Issue Repositories
	Zaidi *et al*. [39]	2022	BERT	**Title and Description**	Developer	Open Issue Repositories

Choetkiertikul et al. [7] proposed the word-based LSTM [16] deep learning method to predict components. As illustrated in Fig. 3, they implemented the LSTM for semantic feature extraction to predict components using word2vec [8] in the word embedding layer from the title and description information in the issue report. The experiment using the LSTM-based method found that an issue report may be assigned to more than one component, requiring multiple labels. Thus, they adopted a multilabeling classification method to classify the component in a multilabeling situation, as Nam et al. [27] proposed.

Furthermore, Choetkiertikul et al. [7] found that an imbalance of the issue report dataset causes a decrease in accuracy. Thus, they deleted the issue reports with a small number of issues to address the data imbalance problem. As a result, they improved the accuracy using deep learning and deletion. The improvement indicates that adjusting the imbalanced dataset is an integral part of the automatic component prediction field. As presented in Table 1, they collected datasets from 11 open-issue repositories, consisting of 142,083 issues with an average of 12,916 issues each. They used this dataset to evaluate the performance of their word-based LSTM model by the recall at k, which is the top-k results of the components.

Choetkiertikul et al. [7] revealed the importance of dataset quality and component prediction possibilities using a deep learning method with the dataset deletion and word-based LSTM methods. Thus, to gain confidence in the performance of the proposed fine-tuned pretrained language model, we also adopted the dataset from Choetkiertikul et al. [7] to evaluate the proposed model.

Kangwanwisit et al. [20] conducted various word-based experiments to find proper feature extraction and classification techniques for the component prediction. For the feature extraction task, various techniques such as bag of word (BoW) [42], N-gram [5] IDF, and TF-IDF were considered. The experimental results showing the performance of the classification task when using Support Vector Machine (SVM) [9], Gradient Boosting (GB) [15], and Random Forest (RF) [4] were also reported. Their analysis indicated that TF-IDF and RF were the best in extracting the textual features and classifying the issues, respectively. They collected approximately 68,000 reports with title and description information for the experiments.

As described above and in Table 2, some approaches have predicted components using machine learning or deep learning methods. Each study has proposed a different word-based method, but the studies use the same information from the issue report dataset: the title and description. In addition, various approaches toward automatic bug triage [21], [26], [39] using deep learning methods have been actively conducted recently. Thus, we also reviewed automatic bug triage research and found that it uses the title and description information from the issue report repository to predict a developer (assignee). Thus, the bug triage research method can be adopted in the field of component prediction due to the similarity of information and method. As a result, we describe the bug triage research below to investigate this for component prediction.

B. AUTOMATED BUG TRIAGE

Lee et al. [21] proposed the convolutional neural network (CNN)-based [32] deep learning model to predict the assignee (developer) using the title and description information extracted from industrial issue reports. They collected 14,583 issues consisting of 225 assignees.

In a bug triage field, the number of the assignees becomes a class, which is the component in component prediction. Thus, it is difficult to compare the prediction performance between the proposed and Lee et al. [21] method. However, the similarity of the information that the data feature extracts from the title and description can prove that deep learning methods, such as the CNN, and pretrained language models can adopt component prediction.

Mani et al. [26] collected a dataset from Google Chromium, Mozilla Core, and Mozilla Firefox, which are reliable open-issue repositories. They aimed to predict the assignee (developer) using the recurrent neural network (RNN) [16]. However, they found that some developers who contribute less than others decrease the performance of the RNN-based model. Thus, they deleted the issue reports according to the number of contributions of the developer. As a result, they obtained a significant result in top-k accuracy, which refers to the top-k developers, from the contribution-based and RNN-based deep learning methods. In addition, they suggested the importance of the dataset quality by conducting contribution dataset regeneration. Thus, these results are evidence of the proposed second task method, which downsamples the dataset by deleting the components with few issues.

TABLE 3. Dataset summary for the tasks 1 to 3.

Repository	Product	Collected issues	Task1 components	Task2 components	Task3 issues
Previous Work [7]	Hbase	8,103	76	68	-
	Hive	9,359	67	65	-
	Infrastructure	10,166	52	51	-
	Cordova	10,927	64	64	-
	Hadoop	6,153	39	37	-
	Fedora CloudSync	1,661	32	22	-
	Islandora	1,430	69	67	-
	JIRA software	23,074	169	142	-
	Confluence	11,019	153	128	-
	Bamboo	7,803	101	96	-
	Moodle	52,388	89	89	-
	Total	142,033	911	829	-
	Average	12,916	82	75	-
Ours	Eclipse Foundation	21,644	41	32	10,171
	Eclipse Platform	18,886	19	16	10,429
	Bugzilla Firefox	19,870	51	41	12,527
	Total	60,400	111	89	33,127
	Average	20,133	37	30	11,042

Zaidi et al. [39] adopted the bidirectional encoder representations from transformers (BERT) [11] model, which is a pretrained language model to predict the developer using the title and description information extracted from the work by Mani et al. [26] and data collected from Mozilla[4] and NetBeans[5]. They also adopted the contribution-based downsampling method that Mani et al. [7] used to address dataset problems. Through the experiments, they achieved better top-k accuracy results than Lee et al. [21] and Mani et al. [7] Thus, this finding indicates that the proposed fine-tuned pretrained language model can predict components correctly because the component prediction dataset has fewer classes, which were developers in Zaidi et al. [39] study.

Through the details above, we found important information for automatic component prediction. First, the dataset of every related work should have or share the method to address the data imbalance in an issue report dataset. Due to its generation method, data imbalance is a characteristic of the issue report dataset. Many issues come from users, which can bias the model toward common functions, such as the user interface (UI), general, and debug, which occur easily and more often than other components.

Second, component prediction and bug triage have similarities due to their methods and information. Previous studies do not consider the source code or stack trace because they cannot view all of the information in the issue report. If the stack trace or source code becomes model input, the language-based model trained by common sentence data cannot understand the meaning of the input, reducing the model performance.

The third critical point is the lack of issue report data. Most related studies shared or constructed the dataset to predict the components or developers, making it challenging to compare the performance of each study's proposed method. Thus, we suggest a task-based dataset downsampling method and our dataset for issue report-based research to overcome these problems.

[4]https://bugzilla.mozilla.org/
[5]https://netbeans.apache.org/

C. PRETRAINED LANGUAGE MODEL

Before sentence-based pretrained language models were proposed, word2vec [8], FastText [18], and Glove [28], which are based on word-embedding methods, were primarily used. These word-embedding-based methods initialize the embedding layer to train the models and use the embedding algorithm in word2vec to implement the already-trained embedding layer. These methods have some problems because each word is embedded into the same vector; thus, they cannot understand the information of the entire sentence.

Furthermore, transformer-based [36] language models, such as ELMo [29] and BERT [11], have been proposed to overcome the limitations of the word-based model. Various embedding vectors exist in an issue report dataset, the main source of automatic component prediction. Therefore, we adopted the transformer-based pretrained language model BERT to increase the accuracy of component prediction.

D. BERT AND IMPROVED MODELS

In 2018, Google proposed BERT [11], a pretrained language model based on text data using Wikipedia for 2.5 billion words and a book corpus of 800 million words using transformers. Moreover, BERT uses large-scale data to require relatively considerable computing resources and learning time compared to existing models. However, it has better results than existing models by fine-tuning pretrained models in complex problems. Subsequently, several follow-up studies have been conducted to improve the performance of BERT.

In this paper, we fine-tuned the BERT-based [11] model to overcome the limits of word-based models. We compared the accuracy of automatic component prediction with previous studies by fine-tuning the robustly optimized BERT approach (RoBERTa) [23] and the BERT based on it.

<Original data>
BugID: ####
Assignee: ####
Title: SmartUpdate
Product: Firefox
Component: Menus\n
Status: RESOLVED\n FIXED\n
Description: Feature tracking. Assess smartupdate needs (back end UI etc)
Firebird so we\ncan push updates to users.

BugID: ####
Assignee: ####
Title: Update contribute.md to reflect latest
Product: Firefox
Component: Security\n
Status: RESOLVED\n WORKSFORME\n
Description: Scope of this bug is to update and reflect latest steps\n https://#####/\n

<Pre-processed data>
Title: SmartUpdate
Component: Menus
Description: Feature tracking. Assess smartupdate needs (back end UI etc)
Firebird so we can push updates to users.

Title: Update contribute.md to reflect latest
Component: Security
Description: Scope of this bug is to update and reflect latest steps

FIGURE 4. **Example of dataset pre-processing. Boxes show the original data (above) and the Pre-processed data (below), respectively.**

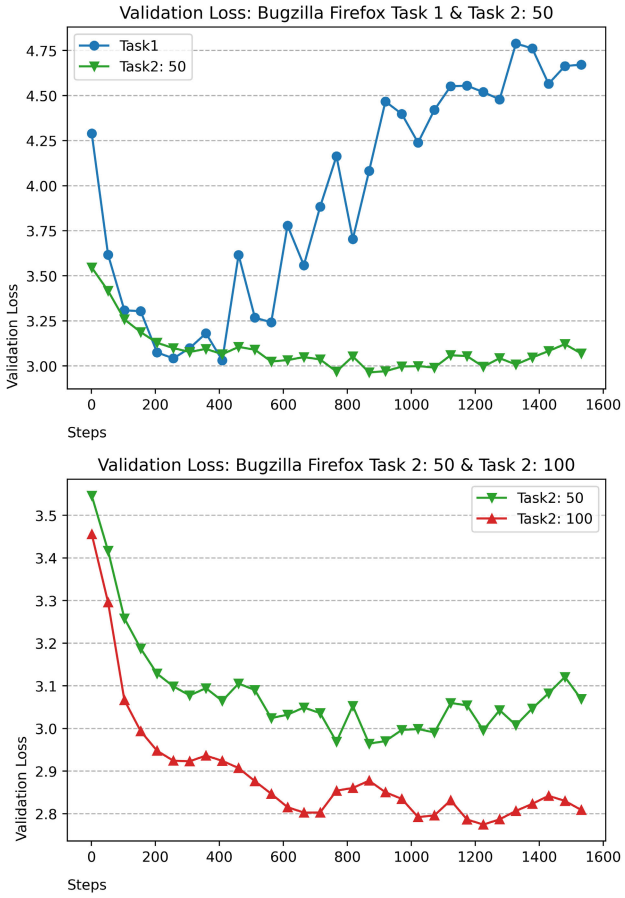

FIGURE 5. **Validation loss graphs for the task-based experiments on Bugzilla Firefox dataset.**

Facebook Artificial Intelligence proposed RoBERTa to improve the performance of the existing BERT model. They trained the BERT model by adding training data, adjusting hyperparameters, and reselecting the training data to conduct replication learning. Through deletion, next sentence prediction improves the maximum sequence length, diversifies the masking pattern, and improves performance compared with BERT.

III. DESIGN AND IMPLEMENTATION

This section discusses the method and experiments of the fine-tuned pretrained language model. Moreover, it presents the task-based downsampling method to improve the performance of the automatic component prediction model.

A. DATASET COLLECTION

Section I and II describe the necessity of adopting an appropriate issue report dataset. Thus, we attempted to determine an appropriate dataset from previous work. However, we could not find a suitable dataset without Choetkiertikul et al. [7] data because most datasets based on the bug triage method to determine the developer do not include component information. Thus, we adopted Choetkiertikul et al. [7] dataset to evaluate the performance of the proposed fine-tuned pretrained language model. By adopting their dataset, we can directly compare performance to demonstrate the robustness and confidence of the proposed method.

We also generated a dataset from reliable open-issue repositories, as presented in Table 3. These data came from Bugzilla Firefox,[6] Eclipse Foundation,[7] and Eclipse Community.[7] To create the proposed dataset, we selected resolved, solved, and closed issues, meaning that the components were already assigned. Thus, the dataset comprised around 60,000 issues consisting of 137 components.

B. DATASET PREPROCESSING

Fig. 1 indicates that issue reports include various information (e.g., title, description, component, product, type, etc.). However, in this paper, we only used the title and description information to predict components, similar to previous work [7], [35], [38] Thus, we arranged the dataset to be suitable for the proposed model, as illustrated in Fig. 4. The issue report dataset consists of natural language that is not standardized, and many people do not follow standards; thus, directly using the dataset as input to a model is challenging without preprocessing. In addition, the dataset consists of various issue report repositories and Choetkiertikul et al. [7] work. Therefore, a suitable preprocessing method must be built for the proposed model.

Based on the issue report datasets, the preprocessing methods are as follows:

[6]https://bugzilla.mozilla.org

[7]https://bugs.eclipse.org/bugs

TABLE 4. The number of issues in the projects for each task based method.

Eclipse Platform				Eclipse Foundation				Bugzilla Firefox			
Method	Task1	Task2	Task3	Method	Task1	Task2	Task3	Method	Task1	Task2	Task3
Name	Quantity	Quantity	Quantity	Name	Quantity	Quantity	Quantity	Name	Quantity	Quantity	Quantity
UI	7,701	7,701	994	CI-Jenkins	3,914	3,914	528	General	2,805	2,805	390
Debug	3,289	3,289	994	Website	2,859	2,859	528	Address Bar	1,863	1,863	390
Ant	1,752	1,752	994	Proposals and Reviews	2,454	2,454	528	Bookmarks & History	1,787	1,787	390
SWT	1,650	1,650	994	Project Management & Portal	1,347	1,347	528	New Tab Page	1,480	1,480	390
Team	929	929	929	Bugzilla	1,173	1,173	528	Messaging System	1,397	1,397	390
Releng	733	733	733	GitHub	1,172	1,172	528	Theme	1,060	1,060	390
Text	630	630	630	Servers	1,145	1,145	528	Sync	1,010	1,010	390
CVS	523	523	523	Git	1,046	1,046	528	Preferences	801	801	390
Compare	356	356	356	Cross-Project	757	757	528	Search	776	776	390
IDE	315	315	315	Gerrit	712	712	528	Tabbed Browser	583	583	390
Resources	273	273	273	Forums and Newsgroups	473	473	473	PDF Viewer	581	581	390
User Assist	248	248	248	Marketplace	465	465	465	Toolbars and Customization	568	568	390
Doc	180	180	180	Architecture Council	396	396	396	Menus	404	404	390
Search	144	144	144	MailingLists	382	382	382	Protections UI	385	385	385
Runtime	93	93	93	Process	365	365	365	Installer	367	367	367
Update (deprecated - use Ecli	57	57	57	Articles	348	348	348	Security	366	366	366
PMC	9	-	-	EclipseCon	323	323	323	Site Identity	359	359	359
Website	3	-	-	Wiki	291	291	291	about:logins	343	343	343
WebDAV	1	-	-	Accounts.eclipse.org	224	224	224	Session Restore	245	245	245
Total(Num of issue)	**18,886**	**18,873**	**8,457**	IPZilla	222	222	222	Untriaged	228	228	228
Average	**994**			CommitterTools	193	193	193	Enterprise Policies	211	211	211
				Dashboard	174	174	174	Downloads Panel	191	191	191
				Nexus	160	160	160	File Handling	179	179	179
				EMO Approvals	127	127	127	WebPayments UI	173	173	173
				Infrastructure	126	126	126	Site Permissions	141	141	141
				Sonar	117	117	117	Migration	139	139	139
				IP Log Tool	113	113	113	Pocket	129	129	129
				vservers	110	110	110	Shell Integration	114	114	114
				Vulnerability Reports	86	86	86	Private Browsing	109	109	109
				Subversion	74	74	74	Type: REG_SZ	106	106	106
				Mail	63	63	63	Nimbus Desktop Client	99	99	99
				API.eclipse.org	59	59	59	Firefox Accounts	91	91	91
				My Account	43	-	-	Extension Compatibility	89	89	89
				License	42	-	-	Tours	84	84	84
				Outside Services	38	-	-	Normandy Client	80	80	80
				FoE Disbursements	17	-	-	Top Sites	76	76	76
				Hosted Services Privacy	10	-	-	Keyboard Navigation	67	67	67
				Mattermost	8	-	-	Offset	55	55	55
				UI Guidelines	8	-	-	Disability Access	53	53	53
				OpenID Connect	7	-	-	Remote Settings Client	52	52	52
				GDPR	1	-	-	Screenshots	50	50	50
				Total(Num of issues)	**21,644**	**21,470**	**10,171**	Pioneer	39	-	-
				Average	**528**			Page Info Window	35	-	-
								Normandy Server	26	-	-
								Foxfooding	24	-	-
								Distributions	15	-	-
								Launcher Process	9	-	-
								Firefox Monitor	8	-	-
								Activity Streams: General	7	-	-
								Headless	6	-	-
								Components	5	-	-
								Total(Num of issues)	**19,870**	**19,696**	**9,651**
								Average	**390**		

1) Delete the new line tags (e.g., \n and \p) from the JSON data format,
2) Delete the URL in a sentence that does not need it for component prediction,
3) Delete the stack trace,
4) Delete the hex code,
5) Separate the dataset into training and testing data,
6) Return a tokenized dataset suitable for the model.

The preprocessing data method in this paper is similar to the method for the existing transformer-based model. Especially in an issue report dataset, there can be stack traces. However, in component prediction, the stack trace is not essential data because it consists of method calls or source code. The BERT-based language model is pretrained using Wikipedia and a data corpus consisting of common sentences that do not have source code information. Thus, the source code information threatens the BERT-based model.

Furthermore, we implemented the human triager method using the title and description information to assign components suitable for component prediction. Thus, we decided to delete the stack trace, which is nonessential in component prediction. Last, we shuffled the dataset to avoid focusing on one class while separating data into 80% for training and 20% for testing.

C. DEALING WITH DATA IMBALANCE

As listed in Table 4, most datasets are biased in specific components. Through the experiment, Fig. 8 and Table 6 reveal that bias issues can be obstacles to training the model. Thus, we propose a task-based method to address data imbalance problems.

In the second task, we deleted the components with fewer than 50 issues, similar to Choetkiertikul et al. [7] method. Choetkiertikul et al. [7] invalidated the number of components that they deleted. Thus, we empirically deleted components with fewer than 50 issues, demonstrating improvements in the dataset. We deleted about 1% of the dataset. To determine the confidence of the second task deletion method, we compared the performance between deleting 50 and 100 issues (Fig. 5). We found that deleting 50 issues

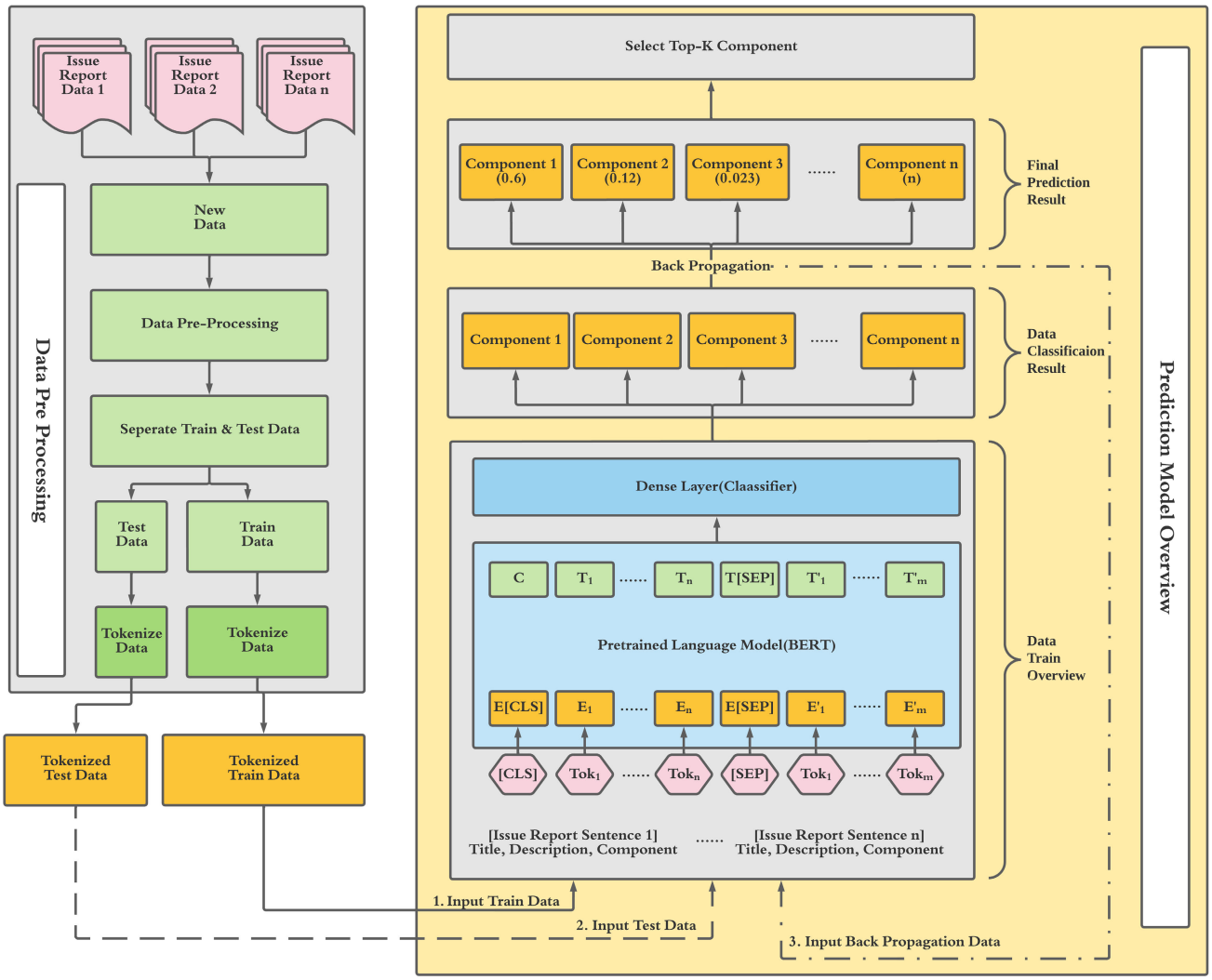

FIGURE 6. The structure of our learning model for automatic component prediction.

exhibited improvement compared with the original data, but no difference was found between deleting 50 and 100 issues. Therefore, we deleted 50 issues for the second task.

The validation loss had not yet stabilized in the first and second tasks (Fig. 5). To stabilize it, we downsampled the dataset with more issues than average. We obtained a better result through the third task, as presented in Fig. 8, without compromising the issue report dataset characteristics.

By comparing each task, we determined the proper dataset normalization method suitable for automatic component prediction when most datasets are imbalanced.

The data handling method presented in Table 4 addresses the following tasks:

Task 1: Keeping the original dataset without removing anything.

Task 2: Deleting the issue reports with the components appearing in the dataset fewer than 50 times.

Task 3: Downsampling the issue reports by limiting the number of issues per component to the average of the total issues.

D. FINE-TUNED PRETRAINED LANGUAGE MODELS

The pretrained language models implemented in this paper are BERT and RoBERTa, which are based on the transformer [36]. Thus, we must tokenize the data using a BERT-based tokenizer, which is suitable for the proposed model that is unlike the word-based methods. As illustrated in Fig. 6, we can obtain tokenized testing and training data through data preprocessing. The model trains using the training data while conducting backpropagation to update the model. After training the model, the model is reverified to classify the recall at k, which indicates the top-k components.

The existing BERT-based models exhibit a decrease in accuracy while conducting multilabel classification [22], which is difficult to classify for many classes, such as

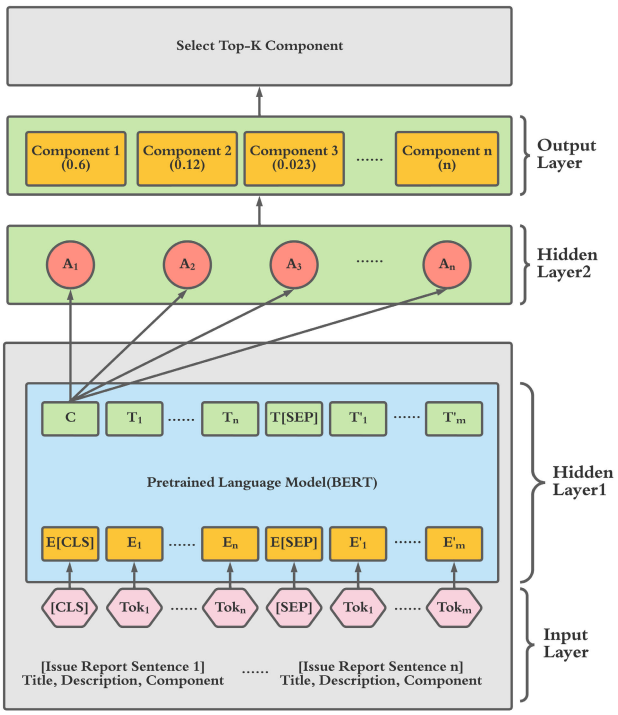

FIGURE 7. The overview of fine-tuned hidden layers.

component prediction or bug triage. Thus, it is necessary to fine-tune the models to conduct multilabel classification. Due to its limitations, it is impossible to predict components that consist of many classes, as presented in Table 3. Thus, as depicted in Fig. 7, we fine-tuned the model to classify many classes to predict components.

We fine-tuned the pretrained language models, BERT and RoBERTa, for the classification task. Note that the number of components (classes) that need to be classified varies depending on the project (e.g., Eclipse Platform: 19, Eclipse Foundation: 41, Bugzilla Firefox: 51). Thus, the classifier layer needs to be fine-tuned to handle different number of classes. Furthermore, we collected the best parameters, such as weight decay to 5e-6 and learning rate to 5e-6 through the experiment, which are important metrics in BERT-based models [34]. Those best parameters are described in Section III-F.

E. EXPERIMENT ENVIRONMENT

During the evaluation, we ran the previous study's experiments to compare the performance of the proposed fine-tuned model. The task-based experiment was conducted in the same environment: Intel(R) Xeon(R) Silver 4214R CPU @ 2.40 GHz and two Nvidia GeForce RTX 3090 with 24 GB.

F. EVALUATION OPTIMIZATION

In this paper, we used the dataset from a previous study to evaluate the performance of the proposed fine-tuned model. We also employed this dataset to evaluate the performance of the downsampling task-based method. We set the parameter values for the weight decay to 5e-6, the learning rate to 5e-6, and the batch size to 300. To avoid overfitting problems, we conducted a set on various weight decay and learning rate values (e.g., from 1e-6 to 5e-6). We found that the training with the weight decay and learning rate at values other than 5e-6 makes the model vulnerable to underfitting or overfitting. We set the dropout value to 0.1.

We trained the proposed model for 1,500 steps during the experiment and applied early stopping when the validation loss stabilized. We obtained a steady decrease in validation loss and an increase in validation accuracy throughout the experiment. Thus, this result suggests that applying the downsampling method proposed in Task 3 can effectively handle imbalanced issue report data. We adopted the AdamW [24] algorithm for the optimization method and set the β_1 to 0.9 and β_2 to 0.999, respectively. In addition, we adopted a deepspeed [30] scheduler to schedule the learning rate during the training. Thus, we set the minimum learning rate to 0 and the maximum learning rate to 5e-6 during the 150 steps. We adopt the recall@k metric to evaluate the performance of the proposed method.

IV. RESULTS

The baseline models [7], [20], [35], [38] consist of a word-based method with its own vocabulary, which might not consist of suitable information for predicting components in new issues. Thus, the proposed fine-tuned pretrained language models predict components using the sequence information from the sentence, which can respond to the new issues correctly. It is much more effective, time-consuming, and cost-effective in the modern software field to manage new issues.

Furthermore, we constructed the model with the title, description, and component data that the triager uses to triage the component in the real world. Thus, the proposed method is more effective in any triage system that manually triages the components.

We evaluated the proposed component prediction method and addressed the following research questions:

- **RQ 1)** Are the proposed fine-tuned pretrained language models more effective than the method from the previous work in automatic component prediction?
- **RQ 2)** Is the proposed method significantly better than the baseline method to predict the top-k components?
- **RQ 3)** Are the proposed task-based methods appropriate in an imbalanced issue report dataset?

A. RESEARCH QUESTION 1

Table 5 presents the validation recall score of the pretrained BERT-based component prediction model, pretrained RoBERTa-based prediction model, and DeepSoft-C method. The RoBERTa and BERT-based models achieved better recall scores than DeepSoft-C for all datasets. The RoBERTa-based model had better recall for the top five components than BERT for every dataset except the Hbase and Fedora CloudSync datasets.

TABLE 5. Evaluation result of R@*k* (recall at *k*) compared with the baseline paper [7].

Measurement Metric	R@5			R@10			R@15		
Repository / Method	DeepSoft-C	BERT	RoBERTa	DeepSoft-C	BERT	RoBERTa	DeepSoft-C	BERT	RoBERTa
Bamboo	0.46	0.74	**0.89**	0.57	0.78	**0.91**	0.63	0.85	**0.93**
Cordova	0.43	0.82	**0.88**	0.57	0.86	**0.91**	0.65	0.88	**0.94**
Confluence	0.62	0.74	**0.77**	0.71	0.80	**0.81**	0.76	0.82	**0.84**
Fedora CloudSync	0.52	**0.58**	0.55	**0.77**	0.70	0.64	0.85	**0.94**	0.87
Hadoop	0.56	0.76	**0.83**	0.72	0.81	**0.87**	0.79	0.86	**0.91**
Hbase	0.49	**0.64**	0.62	0.61	0.71	**0.73**	0.68	**0.78**	0.77
Hive	0.36	0.70	**0.73**	0.46	0.75	**0.79**	0.55	0.80	**0.82**
Infrastructure	0.72	0.73	**0.83**	0.79	0.85	**0.90**	0.83	0.88	**0.92**
Islandora	0.47	0.60	**0.61**	0.57	0.63	**0.66**	0.64	0.67	**0.68**
JIRA software	0.50	0.76	**0.80**	0.61	0.82	**0.90**	0.65	0.88	**0.96**
Moodle	0.45	0.62	**0.71**	0.56	0.69	**0.76**	0.62	0.72	**0.79**

TABLE 6. Evaluation result of R@*k* for the task based methods.

Repository Name	**Method**	**Task1**			**Task2**			**Task3**			**Imp(Task3 - Task1)**		
		R@5	**R@10**	**R@15**	**R@5**	**R@10**	**R@15**	**R@5**	**R@10**	**R@15**	**R@5**	**R@10**	**R@15**
Bugzilla Firefox	BERT	0.20	0.27	0.36	0.21	0.29	0.39	0.24	0.48	0.55	**0.04**	**0.21**	**0.19**
	RoBERTa	0.15	0.29	0.37	0.18	0.27	0.38	0.40	0.48	0.55	**0.25**	**0.19**	**0.18**
Eclipse Platform	BERT	0.57	0.56	0.63	0.65	0.85	0.87	0.91	0.93	0.94	**0.34**	**0.37**	**0.31**
	RoBERTa	0.63	0.65	0.68	0.71	0.80	0.88	0.90	0.93	0.95	**0.27**	**0.28**	**0.27**
Eclipse Foundation	BERT	0.35	0.36	0.36	0.34	0.37	0.37	0.66	0.82	0.90	**0.31**	**0.46**	**0.54**
	RoBERTa	0.37	0.36	0.36	0.33	0.36	0.37	0.81	0.88	0.90	**0.44**	**0.52**	**0.54**

The Infrastructure data comprise 10,166 issue reports with 51 components. It turns out that the dataset is highly imbalanced. Note that the BERT-based model shows only 1% higher accuracy than DeepSoft-C. In contrast, RoBERTa achieves 11% higher accuracy than DeepSoft-C. Similarly, the Fedora CloudSync dataset is imbalanced, too, and it has only 1,617 issues and 22 components. We suspect that the data imbalance contributed to the overfitting and rendered the accuracy difference of RoBERTa and DeepSoft-C to only 3%.

Issue reports may contain a homonym which cannot be distinguished by a word-based model and this might lead to decreased accuracy. In addition, component prediction typically requires large number of classes, and the learning ability tends to decrease when the number of classes increases. Our experimental results show that using the fine-tuned pretrained language model proposed in this paper can yield good results in such environments.

A similar trend was found for the top-10 component prediction. The BERT and RoBERTa models outperformed DeepSoft-C. However, the RoBERTa-based model outperformed the BERT-based model for the Hbase dataset, which differs from the observation for the top five components. Recall scores at 15 components also presented a trend similar to the top five components, where the RoBERTa-based model had better results than the BERT-based model for all datasets except the Fedora CloudSync and Hbase datasets.

In an automatic component prediction evaluation, the recall at five is a strict evaluation for the models. Thus, the higher recall scores at five suggest that the proposed fine-tuning method is suitable for component prediction using pretrained language models. In summary, the proposed pretrained RoBERTa-based fine-tuned model is a better than the proposed pretrained BERT-based fine-tuned model and baseline DeepSoft-C method for component prediction.

It is common to fine-tune the BERT-based language models to a specific task as shown in various previous approaches such as [40] and [37]. Our experimental results reveal that our fine-tuned pretrained language models also achieve better performance through the fine-tuning methods described in Section III and Fig.7. To the best of our knowledge, this is the first work that uses a sentence-based model to predict components for issue reports.

B. RESEARCH QUESTION 2

Throughout the experiments, the proposed methods showed better recall at k compared to the baseline method as presented in Table 5. Specifically, to evaluate the performance of the proposed method, we adopted 11 datasets, which came from the baseline paper [7]. In addition to the comparison of the experimental results, we performed statistical tests to check the significance of the results by comparing the recalls at the top 5, 10, and 15 components.

The Friedman test [13] was performed to check the overall significance of the results with a 95% confidence interval. The test had a p-value of less than 0.05, confirming the significance of the results. Then, a Wilcoxon–Holm posthoc test [17] was performed to identify the significantly better model for top-k component prediction among the proposed and baseline models. The pairwise testing had a p-value of less than 0.05 for all pairs, confirming the significance of the methods.

Furthermore, the Demšar diagram [10] in Fig. 9 was built to rank the comparative methods for predicting the top 5, 10, and 15 components. The RoBERTa-based method was

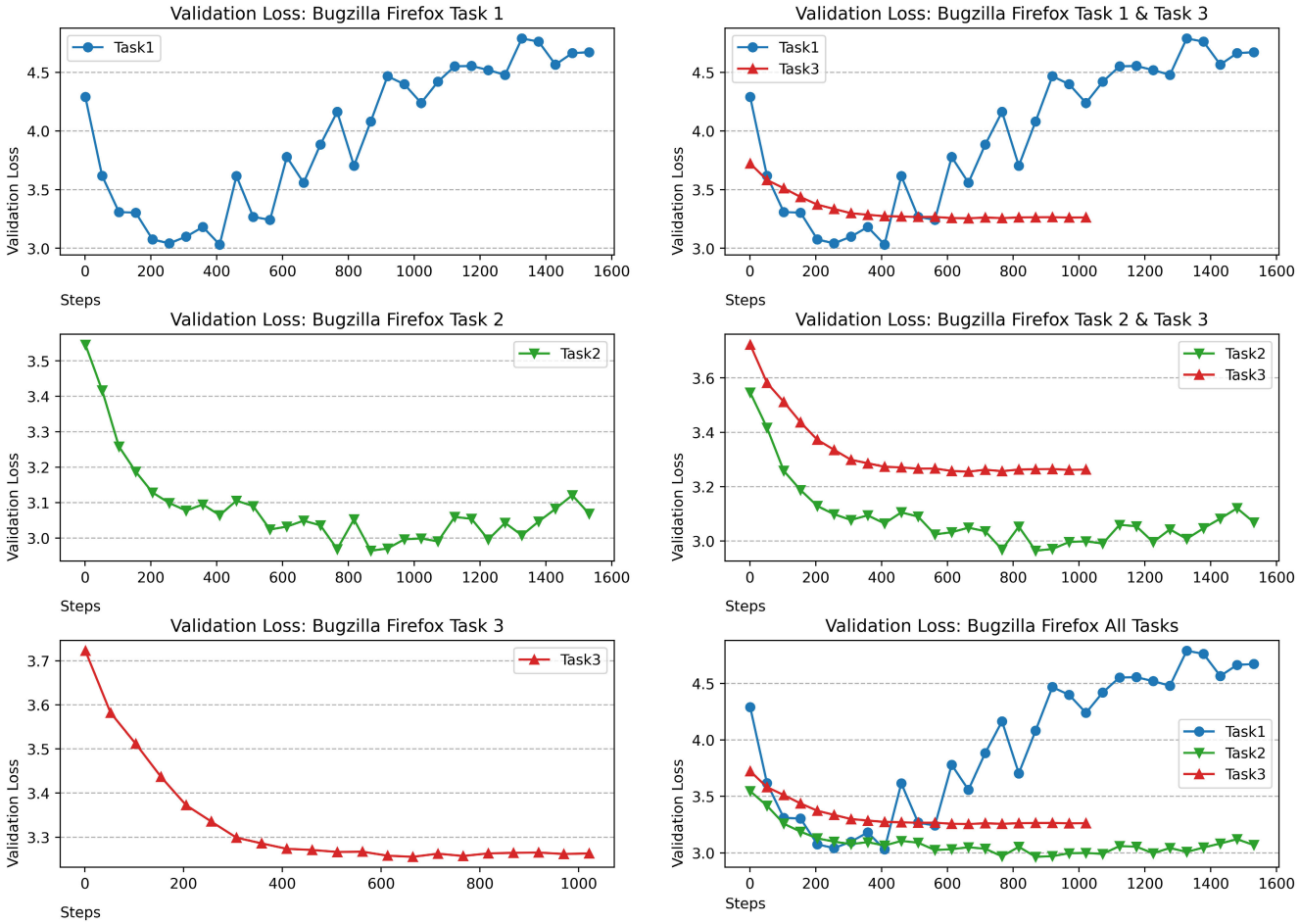

FIGURE 8. Validation loss for the tasks 1-3 on Bugzilla Firefox dataset3. Task 2 stopped at the step 1,000 due to its early stabilization.

the best method out of all three methods. The average ranks for RoBERTa, BERT, and DeepSoft-C are 1.1818, 2, and 3, respectively. The Demšar diagram confirms the significant difference in the results of all three methods because there is no connection between the methods in the Demšar diagram for the top 5, 10, and 15. In the Demšar diagram, a connection between the two methods indicates an insignificant difference in the results. In summary, the statistical tests suggest that the RoBERTa-based method and the BERT-based methods are significantly better than the BERT-based component prediction method.

C. RESEARCH QUESTION 3

We evaluated the task-based method by comparing the difference in the validation loss graph described in Fig. 8. We experimented with the Bugzilla Firefox dataset using the fine-tuned BERT model.

Task 1 was evaluated with the original dataset, which did not apply any methods. We found that the validation loss does not decrease evenly. Thus, we adjusted the learning rate, the standard method to address the problem. Despite our efforts, the validation loss and accuracy do not evenly decrease or increase. Therefore, the collected dataset also has a data imbalance problem, like most previous studies.

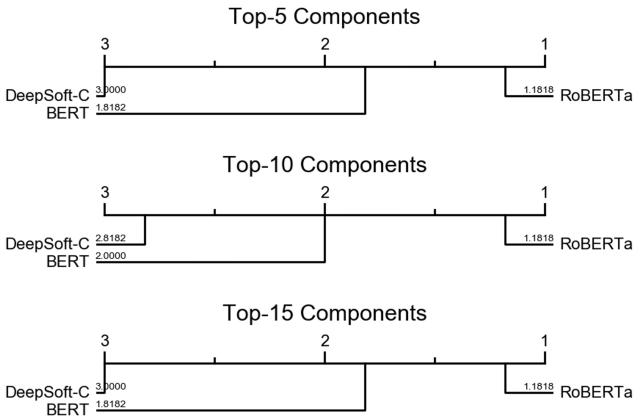

FIGURE 9. Demšar diagram for the experimental results.

Choetkiertikul et al. [7] deleted components with insufficient issues on their standard to address the Task 1 problem. Thus, we deleted the components that consisted of fewer than 50 issues. As we described in Fig 5 and Section III, Choetkiertikul et al. [7] did not explain the states of the component deletion. Therefore, we deleted 50 issues, empirically comparing the validation loss when deleting components with

0 or fewer than 50 or 100 issues. In Task 2, we obtained good results, demonstrating a more even decrease in the validation loss than in Task 1. However, Fig. 8 reveals that the decrease of validation loss still fluctuates beginning at 400 steps. Therefore, Task 2 suggests that only deleting the components is not always the proper method to address imbalanced data.

We proposed Task 3 to resolve the limitations of Task 2, which is downsampling the dataset. Table 4 indicates that most issue report data are easily biased due to their characteristics. During the triage cycle, common issues are triaged into common components (e.g., Bugzilla Firefox: General, Eclipse Foundation: CI-Jenkins, Eclipse Platform: UI), which frequently occurs during the software life cycle. Thus, only deleting components with a few issues is insufficient to address the biased data. These bias issues are considered characteristic of the issue report repository.

Therefore, we downsampled the components to obtain a higher-than-average number of issues for each repository. Through Task 3, based on the downsampling method, we obtained an even decrease in validation loss (Fig. 8). Furthermore, Table 6 presents a 54% increase in the recall at 15 in the Eclipse Foundation dataset, the most imbalanced data. Thus, the proposed task-based method that downsampled the dataset without compromising the characteristics of the issue report repository is useful in component prediction.

V. THREATS TO VALIDITY

The lack of issue report datasets with component information may threaten the robustness of this research. Our study uses datasets imported from several open issue repositories, and each dataset contains almost 20,000 issues per repository. Compared to the datasets of the previous work [6] targeting for the component prediction which contain an average of 12,916 issues per repository, the size of our datasets seems comparable. In contrast, a recent study [26] targeting for the developer prediction uses a dataset containing nearly 200,000 issue reports. Note that the component field can be omitted when writing a issue report. Therefore, there are many issue reports without the component information, which makes it difficult for researchers to obtain enough datasets for the component prediction.

Another potential threat is that we utilize only the title and description information in an issue report when predicting its component information, which is similarly done in the previous work [7]. As mentioned in Section I and Fig. 1 each issue report may contain various information other than the tile and description. For example, the information such as issue reporter and product name could be also utilized for the prediction. Furthermore, the component information can be used to predict the developer and vice versa. We are planning to explore this in our next study.

We use a task-based method to address the dataset imbalance problem in this study. During the experiments, we observed improvements in the recall at k and a steady decrease in validation loss. However, deleting the biased component with more than the average number of issues is not a fundamental solution for an imbalanced dataset. Especially, deleting issues might lose important information for training the model even if we shuffle the dataset.

VI. CONCLUSION

In this paper, we proposed an approach to improve the performance of automatic component prediction by using the pretrained language models which are sentence-based deep learning. We argued that the previous approaches relying on word-based deep learning models did not provide sufficient performance. Specifically, we fine-tuned the BERT and RoBERTa to fit the issue report dataset consisting of various classes. We believe that because the sentence-based pretrained models use information from the whole sequence of information, it also contributed to the performance of the proposed approach, providing better experimental results than those from the study by Choetkiertikul et al. [7]

We collected the issue reports from various open-source issue repositories to evaluate the performance of the proposed model. Using the task-based methods, we effectively addressed the imbalance problem in the dataset. We obtained a 45% improvement compared with the method from the previous work. The method for Task 3 exhibits a 54% improvement in the recall at 15 in the Eclipse Foundation dataset in Table 6. Furthermore, the task-based method also improves the performance of the training model and obtains high accuracy on all datasets. This method improves the performance of component prediction and demonstrates the importance of dataset quality.

Finally, the experimental results for the proposed method suggest that fine-tuned pretrained language models are effective for automatic component prediction. Our proposed method showed decent accuracy even in a small dataset environment, indicating that if sufficient datasets with components are prepared in the future, we can expect better performance in component prediction. Thus, we plan to employ data augmentation methods in issue reports in future work. Retraining a pretrained language model suitable for an issue report dataset can improve automatic component prediction. Especially in industrial environments, triaging issues manually can be very inefficient due to frequent software updates and expanded scope of the project and stakeholder. Thus, we expect to apply our proposed approach to industrial projects and evaluate its effectiveness.

REFERENCES

[1] J. Anvik, L. Hiew, and G. C. Murphy, "Who should fix this bug?" in *Proc. 28th Int. Conf. Softw. Eng.*, May 2006, pp. 361–370.
[2] J. Anvik and G. C. Murphy, "Reducing the effort of bug report triage: Recommenders for development-oriented decisions," *ACM Trans. Softw. Eng. Methodol.*, vol. 20, no. 3, pp. 1–35, Aug. 2011.
[3] Y. Bengio, P. Simard, and P. Frasconi, "Learning long-term dependencies with gradient descent is difficult," *IEEE Trans. Neural Netw.*, vol. 5, no. 2, pp. 157–166, Mar. 1994.
[4] L. Breiman, "Random forests," *Mach. Learn.*, vol. 45, no. 1, pp. 5–32, 2001.
[5] A. Z. Broder, S. C. Glassman, M. S. Manasse, and G. Zweig, "Syntactic clustering of the web," *Comput. Netw. ISDN Syst.*, vol. 29, nos. 8–13, pp. 1157–1166, Sep. 1997.

[6] M. Choetkiertikul, H. K. Dam, T. Tran, T. Pham, and A. Ghose, "Predicting components for issue reports using deep learning with information retrieval," in *Proc. 40th Int. Conf. Softw. Eng., Companion Proceeedings*, May 2018, pp. 244–245.
[7] M. Choetkiertikul, H. K. Dam, T. Tran, T. Pham, C. Ragkhitwetsagul, and A. Ghose, "Automatically recommending components for issue reports using deep learning," *Empirical Softw. Eng.*, vol. 26, no. 2, pp. 1–39, Mar. 2021.
[8] K. W. Church, "Word2Vec," *Nat. Lang. Eng.*, vol. 23, no. 1, pp. 155–162, 2017.
[9] C. Cortes and V. Vapnik, "Support-vector networks," *Mach. Learn.*, vol. 20, no. 3, pp. 273–297, 1995.
[10] J. Demšar, "Statistical comparisons of classifiers over multiple data sets," *J. Mach. Learn. Res.*, vol. 7, pp. 1–30, Dec. 2006.
[11] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," 2018, *arXiv:1810.04805*.
[12] S. T. Dumais, "Latent semantic analysis," *Annu. Rev. Inf. Sci. Technol.*, vol. 38, no. 1, pp. 188–230, 2004.
[13] M. Friedman, "The use of ranks to avoid the assumption of normality implicit in the analysis of variance," *J. Amer. Statist. Assoc.*, vol. 32, no. 200, pp. 675–701, Dec. 1937.
[14] S. Gururangan, A. Marasović, S. Swayamdipta, K. Lo, I. Beltagy, D. Downey, and N. A. Smith, "Don't stop pretraining: Adapt language models to domains and tasks," 2020, *arXiv:2004.10964*.
[15] T. Hastie, R. Tibshirani, and J. Friedman, "Boosting and additive trees," in *The Elements of Statistical Learning*. New York, NY, USA: Springer, 2009, pp. 337–387.
[16] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Comput.*, vol. 9, no. 8, pp. 1735–1780, 1997.
[17] S. Holm, "A simple sequentially rejective multiple test procedure," *Scandin. J. Statist.*, vol. 6, no. 2, pp. 65–70, 1979.
[18] A. Joulin, E. Grave, P. Bojanowski, M. Douze, H. Jégou, and T. Mikolov, "FastText.Zip: Compressing text classification models," 2016, *arXiv:1612.03651*.
[19] G. Kakarontzas, I. Stamelos, S. Skalistis, and A. Naskos, "Extracting components from open source: The component adaptation environment (COPE) approach," in *Proc. 38th Euromicro Conf. Softw. Eng. Adv. Appl.*, Sep. 2012, pp. 192–199.
[20] P. Kangwanwisit, M. Choetkiertikul, C. Ragkhitwetsagul, T. Sunetnanta, R. Maipradit, H. Hata, and K. Matsumoto, "A component recommendation model for issues in software projects," in *Proc. 19th Int. Joint Conf. Comput. Sci. Softw. Eng. (JCSSE)*, Jun. 2022, pp. 1–6.
[21] S.-R. Lee, M.-J. Heo, C.-G. Lee, M. Kim, and G. Jeong, "Applying deep learning based automatic bug triager to industrial projects," in *Proc. 11th Joint Meeting Found. Softw. Eng.*, Aug. 2017, pp. 926–931.
[22] W. Liu, P. Zhou, Z. Zhao, Z. Wang, H. Deng, and Q. Ju, "FastBERT: A self-distilling BERT with adaptive inference time," 2020, *arXiv:2004.02178*.
[23] Y. Liu, M. Ott, N. Goyal, J. Du, M. Joshi, D. Chen, O. Levy, M. Lewis, L. Zettlemoyer, and V. Stoyanov, "RoBERTa: A robustly optimized BERT pretraining approach," 2019, *arXiv:1907.11692*.
[24] I. Loshchilov and F. Hutter, "Decoupled weight decay regularization," 2017, *arXiv:1711.05101*.
[25] Y. Lu, Q. Mei, and C. Zhai, "Investigating task performance of probabilistic topic models: An empirical study of PLSA and LDA," *Inf. Retr.*, vol. 14, no. 2, pp. 178–203, 2011.
[26] S. Mani, A. Sankaran, and R. Aralikatte, "DeepTriage: Exploring the effectiveness of deep learning for bug triaging," in *Proc. ACM India Joint Int. Conf. Data Sci. Manage. Data*, Jan. 2019, pp. 171–179.
[27] J. Nam, J. Kim, E. L. Mencía, I. Gurevych, and J. Fürnkranz, "Large-scale multi-label text classification—Revisiting neural networks," in *Proc. Joint Eur. Conf. Mach. Learn. Knowl. Discovery Databases*. Nancy, France: Springer, 2014, pp. 437–452.
[28] J. Pennington, R. Socher, and C. Manning, "Glove: Global vectors for word representation," in *Proc. Conf. Empirical Methods Natural Lang. Process. (EMNLP)*, 2014, pp. 1532–1543.
[29] E. Matthew Peters, M. Neumann, M. Iyyer, M. Gardner, C. Clark, K. Lee, and L. Zettlemoyer, "Deep contextualized word representations," 2018, *arXiv:1802.05365*.
[30] J. Rasley, S. Rajbhandari, O. Ruwase, and Y. He, "DeepSpeed: System optimizations enable training deep learning models with over 100 billion parameters," in *Proc. 26th ACM SIGKDD Int. Conf. Knowl. Discovery Data Mining*, Aug. 2020, pp. 3505–3506.
[31] S. Robertson, "Understanding inverse document frequency: On theoretical arguments for IDF," *J. Documentation*, vol. 60, no. 5, pp. 503–520, Oct. 2004.
[32] P. Y. Simard, D. Steinkraus, and J. C. Platt, "Best practices for convolutional neural networks applied to visual document analysis," in *Proc. 7th Int. Conf. Document Anal. Recognit.*, Edinburgh, U.K., vol. 3, 2003, pp. 958–963.
[33] A. Singhal, "Modern information retrieval: A brief overview," *IEEE Comput. Soc. Tech. Committee Data Eng.*, vol. 24, no. 4, pp. 35–43, Dec. 2001.
[34] C. Sun, X. Qiu, Y. Xu, and X. Huang, "How to fine-tune bert for text classification?" in *Proc. China Nat. Conf. Chin. Comput. Linguistics*. Kunming, China: Springer, pp. 194–206, 2019.
[35] A. Sureka, "Learning to classify bug reports into components," in *Proc. Int. Conf. Model. Techn. Tools Comput. Perform. Eval.* Prague, Czech Republic: Springer, 2012, pp. 288–303.
[36] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, Ł. Kaiser, and I. Polosukhin, "Attention is all you need," in *Proc. Adv. Neural Inf. Process. Syst.*, vol. 30, 2017, pp. 1–11.
[37] D. Wadden, U. Wennberg, Y. Luan, and H. Hajishirzi, "Entity, relation, and event extraction with contextualized span representations," 2019, *arXiv:1909.03546*.
[38] M. Yan, X. Zhang, D. Yang, L. Xu, and J. D. Kymer, "A component recommender for bug reports using discriminative probability latent semantic analysis," *Inf. Softw. Technol.*, vol. 73, pp. 37–51, May 2016.
[39] S. F. A. Zaidi, H. Woo, and C.-G. Lee, "Toward an effective bug triage system using transformers to add new developers," *J. Sensors*, vol. 2022, pp. 1–19, Apr. 2022.
[40] T. Zhang, F. Wu, A. Katiyar, K. Q. Weinberger, and Y. Artzi, "Revisiting few-sample BERT fine-tuning," 2020, *arXiv:2006.05987*.
[41] W. Zhang, "Efficient bug triage for industrial environments," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol. (ICSME)*, Sep. 2020, pp. 727–735.
[42] Y. Zhang, R. Jin, and Z. Zhou, "Understanding bag-of-words model: A statistical framework," *Int. J. Mach. Learn. Cybern.*, vol. 1, nos. 1–4, pp. 43–52, Dec. 2010.

DAE-SUNG WANG was born in Jeonju, South Korea. He received the B.S. degree in software engineering from Jeonbuk National University, Jeonju. He is currently pursuing the M.S. degree in computer science and engineering with Chung-Ang University, Seoul, South Korea. His research interests include software engineering, requirement engineering, and natural language processing.

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