IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Received 9 March 2024, accepted 7 May 2024, date of publication 3 June 2024, date of current version 12 June 2024.

Digital Object Identifier 10.1109/ACCESS.2024.3408688

RESEARCH ARTICLE

A Comparison of Pretrained Models for Classifying Issue Reports

JUEUN HEO[1], GIBEOM KWON[1], CHANGWON KWAK[1], AND SEONAH LEE[1,2], (Member, IEEE)
[1]Department of AI Convergence Engineering, Gyeongsang National University, Jinju 52828, Republic of Korea
[2]Department of Software Engineering, Gyeongsang National University, Jinju 52828, Republic of Korea

Corresponding author: Seonah Lee (saleese@gnu.ac.kr)

This work was supported in part by the Basic Science Research Program through the National Research Foundation of Korea (NRF) funded by the Ministry of Education under Grant NRF-2021R1A2C1094167, in part by the "Leaders in Industry-University Cooperation 3.0" Project, and in part by the Ministry of Education and National Research Foundation of Korea under Grant LINC3.0-2023-11, 1345370630.

ABSTRACT Issues are evolving requirements in software engineering. They are the main factors that increase the cost of software evolution. To help developers manage issues, GitHub provides issue labeling mechanisms in issue management systems. However, manually labeling issue reports still requires considerable developer workload. To ease developers' burden, researchers have proposed automatically classifying issue reports. To improve the classification accuracy, researchers adopted deep learning techniques and pretrained models. However, pretrained models in the general domain such as RoBERTa have limitations in understanding the contexts of software engineering tasks. In this paper, we create a pretrained model, IssueBERT, with issue data to understand whether a domain-specific pretrained model could improve the accuracy of issue report classification. We also adopt and explore several pretrained models in the software engineering domain, namely, CodeBERT, BERTOverflow, and seBERT. We conduct a comparative experiment on these pretrained models to evaluate their performance in classifying issue reports. Our comparison results show that IssueBERT outperforms the other pretrained models. Noticeably, IssueBERT yields an average F1 score that is 1.74% higher than that of seBERT and 3.61% higher than that of RoBERTa, even though IssueBERT was pretrained with much less data than seBERT and RoBERTa.

INDEX TERMS Issue reports, issue classification, BERT, pretrained models, deep learning techniques.

I. INTRODUCTION

Issues are evolving requirements in software engineering. They are the main factors that increase the cost of software evolution. Therefore, managing issues well is a way to reduce the cost of software evolution while systematically evolving software systems. Issue classification is a way to achieve systematic issue management. The GitHub issue management system provides a lightweight labeling mechanism for managing issue reports. Although the mechanism is simple enough for developers to easily report their issues, developers still need to label issue reports by themselves. Manual labeling tasks are repetitive and cumbersome.

Researchers have developed automated issue classification techniques to reduce developers' manual workloads. Initially, researchers proposed automatically classifying issue reports into categories such as *bugs* and *nonbugs* [1], [2], [3], [4], [5]. However, such a classification is far from feasible because developers add multiple labels to issue reports [6]. Another group of researchers proposed automatically classifying issue reports with multiple labels [7], [8], [9], [10], [11]. However, such approaches still suffer from low classification accuracies of up to 64%. Recently, to improve the accuracy of issue report classification, researchers have used pretrained deep learning models [8], [9], [10]. For example, Wang et al. [9] compared the performances of BiLSTM, CNN, RCNN, and BERT and showed that the pretrained bidirectional encoder representations from transformers (BERT) model can improve the performance in multilabel recommendations.

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

However, their accuracies are between 38% and 55%, according to their F1-scores.

We notice a gap between training pretrained models with general domain data and applying those models to software engineering domain contexts, thanks to Mosel et al.'s work [12]. We also observe that there are software engineering-specific pretrained models such as CodeBERT [13], BERTOverflow [14] and seBERT [12]. However, very few researchers have proposed methods based on software engineering-specific pretrained models for issue classification tasks. Among the pretrained models in the software engineering domain, seBERT is the only model that uses issue data for pretraining. seBERT also uses StackOverflow data and commit messages for pretraining. The performance of seBERT has been evaluated experimentally on the binary issue classification task of determining whether issues are bugs or nonbugs. However, such type-based classification is far from practical, as mentioned in the above paragraphs.

Therefore, we develop a pretrained model, IssueBERT, based on only an issue dataset. We then explore the performance of IssueBERT and other pretrained models, namely, RoBERTa [15], CodeBERT [13], BERTOverflow [14], seBERT [12] on issue classification tasks. We investigate whether software engineering-specific pretrained models can outperform general domain pretrained models such as RoBERTa, which are the existing issue classification approaches. We also determine which pretrained model outperforms other pretrained models on issue classification tasks.

Our experiments show that IssueBERT outperforms the other pretrained models and that seBERT performs second-best overall. Noticeably, IssueBERT yields an average F1 score that is 1.74% higher than that of seBERT and 3.61% higher than that of RoBERTa, even though IssueBERT was pretrained with much less data than seBERT and RoBERTa. The performance of CodeBERT is similar to that of RoBERTa. BERTOverflow performs worse than RoBERTa. Our findings contrast with those of an experiment on seBERT in which seBERT and BERTOverflow performed the best on the issue type-based classification task.

Our additional findings are as follows. First, the models pretrained on issue data perform better than do the models pretrained on general domain data for issue classification tasks because IssueBERT and seBERT outperform RoBERTa (see Sections IV. A and IV.B). Second, the homogeneity between the pretraining data and the fine-tuning and application data is still important because BERTOverflow performs the worst when applied to issue classification tasks (see Sections IV-A and IV-B). Third, the structure of a pretrained model and its hyperparameter values can also affect the classification accuracy, considering that RoBERTa performs moderately (see Section IV-B).

Our contributions are as follows.

- First, we develop IssueBERT, which is pretrained on only issue data.
- Second, we perform comparative experiments with various pretrained models, namely, RoBERTa, CodeBERT, BERTOverflow, seBERT and IssueBERT, for binary issue classification tasks and multilabel issue classification tasks.
- Third, we uncover that IssueBERT outperforms other models on issue classification tasks, even though IssueBERT was trained on a much smaller dataset than the other models.
- Fourth, we characterize the models that can improve the issue classification task performance by analyzing the characteristics of the pretrained models.
- Fifth, we report our experimental results, which contrast with the experimental results obtained for seBERT [12].
- Finally, we make IssueBERT publicly available at https://huggingface.co/gbkwon/issueBERT-large.

The remainder of our paper is organized as follows. Section II introduces our related work. Section III explains our experimental setup. Section IV presents and discusses the experimental results. Section V discusses additional experimental results. Section VI explores the implications of our paper. Section VII discusses the threats to the validity of our research. Finally, Section VIII concludes our paper.

II. RELATED WORK

The related work can be divided into three groups. The first group addresses issue classification. The second group proposes pretrained models with software engineering data. The third group compares several different pretrained models for software engineering tasks.

A. ISSUE CLASSIFICATION

1) BINARY CLASSIFICATION

Studies classify issue reports into two groups, *bugs* and *nonbugs*. Sohrawardi et al. [1] proposed a technique for automatically classifying issue reports into *bugs* and *nonbugs*. The authors used machine learning techniques such as Naive Bayes, kNN, Pegasus, Rochio, and Perceptron. They evaluated these techniques on four projects: HttpClient, Jackrabbit, Lucene, and Tomcat5. In their experiment, the perceptron model performed the best, with an average error rate of 23%.

Pandey et al. [2] also proposed a technique for classifying issue reports into *bugs* and *nonbugs*. The authors used machine learning techniques such as Naive Bayes, Latent Dirichlet Allocation (LDA), kNN, Support Vector Machine (SVM), Decision Tree, and Random Forest. Their experimental results showed that the random forest and SVM methods were the most accurate, with accuracies ranging from 75–83%, depending on the project.

Fan et al. [3] focused on distinguishing *bugs* from *nonbugs* among all issues. They proposed a new two-stage classification method. They evaluated their method with other machine learning techniques such as SVM, Naive Bayes, Logistic Regression, and Random Forest on 80 projects. The results showed that the proposed method performed well;

the 1st quartile of the average F-measure was 75%, and the median was 79%.

Pandey et al. [4] also studied how machine learning can be used to automatically classify issue reports as *bugs* or *nonbugs*. They extracted summaries from issue reports and classified them using machine learning techniques such as Naive Bayes, SVM, Logistic Regression, and LDA. The results showed that Naive Bayes and SVM classifiers had the highest accuracy, with accuracies ranging from 61% to 77% and from 64% to 78%, respectively.

Zhu et al. [5] proposed a novel approach for automatically distinguishing between *bugs* and *nonbugs* in an issue tracking system. They used a kNN machine learning algorithm to determine whether the existing labels were accurate and attention-based bidirectional Long Short-Term Memory (LSTM) to classify issue reports. Their approach outperformed state-of-the-art machine learning-based methods, with an F-measure of 85%.

Zhifang, Liao, et al. [16] utilized transfer learning and the personal characteristics of submitters to classify issue reports as *bugs* or *nonbugs*. Their proposed two-stage learning approach involves fine-tuning the BERT language model and utilizing nine characteristics representing the submitters' influence on and familiarity with the projects. The experimental results showed that the proposed method outperformed state-of-the-art classification methods, with an F-measure of 85%.

However, they used the BERT model trained on general domain data; thus, the BERT model was limited in understanding domain-specific issue data. In this paper, we adopt several BERT-based models trained on software engineering domain data and evaluate the issue classification performance of these models.

2) MULTILABEL CLASSIFICATION

Recent studies have classified issue reports with multiple labels as developers do in practice. Xie et al. [17] proposed MULA, a just-in-time multilabeling system, using FastText [18]. MULA automatically assigns multiple labels to issue reports. Xie et al. [17] constructed a dataset with 81,601 issues and 11 labels to build a MULA model. They compared the performance of the MULA model with those of five other models: BR+RandomForest, BR+kNN, TextCNN, TextRNN, and BR+SVM. Although the MULA model showed F1-scores between 45% and 87%, the evaluation was conducted based on the set of labels, not the set of issue reports. Hence, the metrics were different from those used in other studies.

Park et al. [7] classified issue reports with custom labels that developers defined for their projects. They adopted FastText [18]. In their experiment, they found that the proposed method yielded F1-scores of 57%, 49%, and 64% for the VS Code, Dart-lang, and TypeScript projects, respectively. Similarly, Heo et al. [8] adopted RoBERTa [15] and showed that the replacement model improved the F1-scores by 2% to 6% for three projects: Dart-lang, VS Code, and TypeScript.

To classify issue reports with multiple labels, Wang et al. [9] adopted a pretrained contextual language model, BERT, and other deep learning models, such as Bi-LSTM, CNN, and RCNN. The authors compared the performances of those models. With the best performance, BERT obtained an F1-score of 38%. Wang et al. [10] proposed a new multilabel prediction framework for predicting personalized labels for different projects automatically. They reported an F1-score of 55%.

Heo et al. [11] noted that some labels are classified with high or low accuracy. They reported that the labels *bug*, *enhancement*, *user- submission*, and *new-version* were classified with F1-scores between 67% and 95%, while the labels *duplicate*, *good-first-issue*, and *discussion* had F1-scores between 31% and 39%.

Researchers used a BERT-based model trained on general domain data in those studies. In this paper, we adopt pretrained models trained on software engineering data, such as CodeBERT, BERTOverflow, seBERT and IssueBERT, and compare their performances on binary and multilabel issue classification tasks.

B. DOMAIN-SPECIFIC PRETRAINED MODELS

Recently, researchers have proposed the following pretrained models for software engineering domain data. Feng et al. [13] developed CodeBERT, a pretrained model designed to learn from bimodal data comprising code in multiple programming languages and text in natural languages. They created bimodal training data by incorporating code from different programming languages (Python, Java, and JavaScript) and function-level natural language documents extracted from GitHub. CodeBERT has demonstrated impressive performance on various downstream tasks involving natural language and programming languages, such as natural language code searching and code-to-documentation generation.

Tabassum et al. [14] developed BERTOverflow, a model pretrained on a corpus comprising 152 million sentences sourced from StackOverflow. They also proposed SoftNer, a model specifically designed for named entity recognition. BERTOverflow led to a noteworthy improvement of approximately 10% in the F1-score compared to the baseline BERT model; the SoftNer model demonstrated an impressive F1-score of 79% for the named entity recognition task, thereby enhancing the performance of BERT-based tagging models.

Guo et al. [19] developed GraphCodeBERT, a pretrained model that considers the inherent code structure by leveraging data flow during the pretraining phase. They assessed the effectiveness of their proposed approach by applying it to four downstream tasks: code discovery, duplicate detection, code translation, and code refinement. Their method achieved state-of-the-art performance across all the tasks.

Lin et al. [20] developed T-BERT, a novel framework that utilizes pretrained BERT models (single-BERT, sham-BERT,

triple-BERT). They compared the performance of this method with those of the vector space model (VSM) and the TraceNN (TNN) in open-source software (OSS) projects. The evaluation results demonstrated that T-BERT outperformed the baseline methods in terms of both accuracy and efficiency.

Mosel et al. [12] collected data from four different sources in the software engineering domain and created a pretrained model, seBERT. They subsequently compared pretrained models such as BERT_{base}, BERT_{large}, BERTOverflow, and seBERT. They found that the models pretrained with software engineering data predicted masked words more precisely than did those pretrained with general domain data. For example, in the sentence ''The [MASK] is thrown,'' BERT_{base} predicted ''rule'', while seBERT predicted ''exception''. They concluded that the models pretrained with software engineering data could better understand software engineering contexts than the other models.

Researchers have proposed models pretrained on software engineering domain data and have evaluated the performance of those models on software engineering tasks. However, pretrained models have rarely been applied to issue classification tasks. In this paper, we focus on evaluating the performance of pretrained models for binary and multilabel issue classification tasks.

C. COMPARISON OF PRETRAINED MODELS

Similar to our study, other studies have compared several pretrained models.

Hadi and Fard [21] focused on automatically classifying app reviews. They used six datasets collected by other researchers, which included 16 classification labels, such as *irrelevant*, *praise*, *aspect evaluation*, *feature request*, and *bug report*. They then compared the performances of pretrained models in classifying app reviews on these datasets. ALBERT and RoBERTa performed well in the experiment. However, they did not consider models pretrained on software engineering domain data, such as CodeBERT.

Troshin and Chirkova [22] compared CodeT5, GraphCodeBERT, PLBART, and other pretrained models from the perspective of code comprehension. They designed and performed tasks related to code syntax, semantics, namespaces, dataflow, and algorithm understanding to evaluate how well those models represent information about various code properties. Their work provided insights into the interpretability of the pretrained models' code and its different aspects. This work focused mainly on comparing and evaluating existing models and introducing probing tools.

To the best of our knowledge, few papers have compared models pretrained on software engineering domain data, and none of those studies evaluated the performance of various pretrained models on issue classification tasks.

III. EXPERIMENTAL SETUP

In this section, we develop a pretrained model, IssueBERT, based on only issue data, and we compare various pretrained models on issue classification tasks. We evaluate the performances of IssueBERT and four other pretrained models: RoBERTa, CodeBERT, BERTOverflow, and seBERT.

Fig. 1 shows the overall flow of our work. First, we create IssueBERT by training BERT_{large} with GitHub issue data. Additional details are provided in Section III-B5. Next, we perform binary and multilabel issue classification tasks using the five pretrained models, including our IssueBERT. Afterward, we compare the performances of the pretrained models and analyze the results in detail.

Subsection III-A presents our research questions, Subsection III-B describes the pretrained models as our subjects, Subsection III-C describes our research data, and Subsection III-D explains our experimental procedure, including data preprocessing and the experiments for answering our research questions.

A. RESEARCH QUESTIONS

We asked three research questions to identify a pretrained model that outperforms the other models on issue classification tasks:

RQ1. Which pretrained model performs well for binary classification of issue reports?

RQ2. Which pretrained model performs well for multilabel classification of issue reports?

RQ3. To what extent is the multilabel classification performance of IssueBERT higher than those of the state-of-the-art approaches FastText and RoBERTa?

B. PRETRAINED MODELS

The pretrained models we used in our experiments were RoBERTa, CodeBERT, BERTOverflow, seBERT, and IssueBERT. Table 1 lists the characteristics and hyperparameter values of these pretrained models.

1) RoBERTa

The developers of RoBERTa (robustly optimized BERT approach) [15] focused on the design of a model to improve the performance of BERT, which is undertrained due to overlooked design choices. RoBERTa was trained on the English Wikipedia (16G), CC-News (Common Crawl News corpus, 76G), Open Web Text (38G), and Stories (31G) datasets; together, these datasets include 160 GB of text data, which is almost 10 times greater than the amount of text data used to train traditional BERT. RoBERTa was trained with a larger byte-level byte-pair encoding (BPE) vocabulary containing 50K subword units, without any additional preprocessing or tokenization of the input. RoBERTa was also trained with dynamic masking, sequences of at most $T = 512$ tokens and full-sentences without next sentence prediction (NSP) loss. RoBERTa was finally trained for 100K steps in large mini-batches of 8K size. We chose this model as the comparison model for this experiment since it performed well as a model based on general domain data [23], [24], [25].

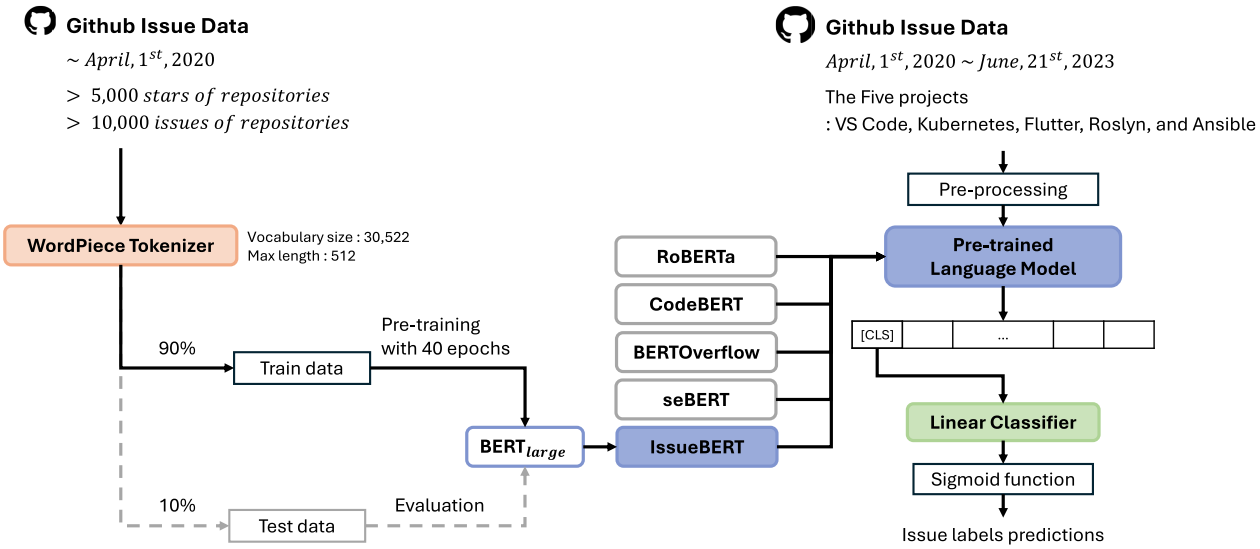


FIGURE 1. Process flow of our work.

TABLE 1. Characteristics and hyperparameter values of the pretrained models.

Model	Based Model	Data Size	Data Type	Method	L[1]	H[2]	A[3]	ML[4]	BS[5]	Steps	P[6]	Task	T[7]	VS[8]
RoBERTa	BERT-base	160 GB	BookCorpus, CC-NEWS, OpenWebText, Stories	Dynamic Masking	12	768	12	512	8K	500K	110M	GLUE, SQuAD, RACE	Byte-Pair Encoding	30,522
CodeBERT	RoBERTa-base	2.1M bimodal data, 6.4M unimodal codes	GitHub bimodal datapoints, unimodal codes across six programming languages	MLM, RTD	12	768	12	512	2,048	100K	125M	Natural language code search, code documentation generation.	Byte-Pair Encoding	50,265
BERTOverflow	BERT-base	152M sentences	StackOverflow	MLM, NSP	12	768	12	512	512	1500K	110M	Code and named entity recognition	WordPiece	64,000
seBERT	BERT-large	119.7 GB	StackOverflow posts, GitHub issues, Jira issues, GitHub commit messages	MLM, NSP	24	1024	16	128	32	100K	340M	Prediction of bug issues, Commit intent classification, Sentiment mining	WordPiece	30,522
IssueBERT	BERT-large	8.97 GB	GitHub issue data	MLM	24	1024	16	512	256	1200K	340M	Issue report binary labeling and multilabeling	WordPiece	30,522

[1] Number of Layers [2] Hidden Layer Size [3] Number of Attention Heads [4] Max Length [5] Batch Size [6] Number of Parameters [7] Tokenizer [8] Vocabulary Size

2) CodeBERT

CodeBERT [13] is a bimodal pretrained bidirectional transformer for natural language and programming languages. CodeBERT was pretrained using a hybrid objective function that combines MLM and RTD tasks. The training data include 2.1M bimodal data pairs of code and the corresponding documentation and 6.4M unimodal data of code. The authors collected the data from GitHub repositories in six programming languages. CodeBERT is based on RoBERTa. The input was set as the concatenation of two segments (NL-PL) with a special separator token. The model used WordPiece tokenization. The maximum length was 512, and the maximum number of training steps was 100K. We chose this model as a comparison model for this experiment because it performs well as a model based on code domain data [26], [27], [28].

3) BERTOverflow

BERTOverflow [14] is a pretrained BERT model specifically designed for the computer programming and software engineering domain and intended for tasks such as code and named entity recognition, using StackOverflow data. It was pretrained with a 64,000 WordPiece vocabulary, utilizing 152 million sentences extracted from StackOverflow. We selected this model as a comparison model for our experiment because it was pretrained on natural language data from the software engineering domain. Since there is a significant similarity in vocabulary between GitHub and StackOverflow, BERTOverflow is expected to perform well for issue classification tasks.

4) seBERT

seBERT [12] is a pretrained model based on BERT_{large} that was specifically designed for the software engineering

domain, similar to BERTOverflow [14]. Unlike BERTOverflow, seBERT was trained with approximately six times more data, and the training data included StackOverflow, GitHub, and Jira issues. In total, 119.7 GB of data was used. seBERT was pretrained based on a 1024-dimensional BERT_{large} model with 30,522 WordPieces and a set sequence length of 128. seBERT was pretrained using the objective function MLM and NSP tasks. seBERT showed vocabulary capture ability related to the SE domain and improved fine-tuning performance in prediction tasks. We selected this model as a comparison model for our experiment since it was pretrained on software engineering domain data and uses the BERT_{large} model, which is known to perform better than BERT_{base}. Trautsch et al. [29] attempted to use seBERT to predict issue types, such as *bugs*, *enhancements*, and *features*.

5) IssueBERT

IssueBERT is a pretrained model that we created from issue data in GitHub. To train the IssueBERT model, we first collected issue data from projects with more than 5,000 stars and more than 10,000 issue reports.[1] For a fair evaluation, we collected the issue data generated by April 1^{st}, 2020, to create IssueBERT. We used 90% of the data as training data and 10% as validation data. We then used the BERT structure proposed by Devlin et al. [30]. These authors achieved successful results in various language understanding tasks. The BERT_{large} model is available through the Hugging Face library and is widely used to develop natural language processing models. The BERT_{large} model uses WordPiece tokenization with a token vocabulary size of 30,522. The maximum length of the tokens is 512, and any sequences exceeding this length are truncated. The masking ratio is 15%, and the other parameters follow the default settings of the BERT_{large} model provided by Hugging Face. IssueBERT[2] was trained for 40 epochs. The required scripts for pretraining IssueBERT are available as part of our replication kit[3]

C. RESEARCH DATA

We selected projects as subjects that met the following criteria as subjects.

We first sorted GitHub projects based on the number of issues. We then manually inspected the top 20 projects. Among these projects, we closely examined the labels attached to issue reports and identified projects with labels corresponding to *bugs* and *features*. Our experiment for RQ1 aims to classify issue reports into these classes. Finally, we selected five projects by reordering the identified projects based on the number of labeled issues. Our experiment for RQ2 aims to assign multiple labels to issue reports. The numbers of issue reports and labels per project are shown in Table 2.

We used issue reports collected from the following five projects to evaluate the classification accuracy of the pretrained models. Because we used the issue data created before April, 1^{st}, 2020 for the pretrained IssueBERT model, we collected the data created after April, 1^{st}, 2020 for our evaluation to avoid the possibility of using the same issue reports and their same linguistic structure in our evaluation.

To conduct the experiments for RQ1 and RQ2, we collected data for the last three years, from 2020-04-01 to 2023-06-21; these data were not used for pretraining. We used the PyGitHub library to extract the issue data for five projects: VS Code, Kubernetes, Flutter, Roslyn, and Ansible.

1) VS CODE

VS Code is a widely used text editor worldwide. The VS Code repository contains numerous issues that have been and continue to be generated. We collected 63,232 issue reports from the VS Code repository.[4] Those issue reports had 454 labels.

2) KUBERNETES

Kubernetes is a container orchestration tool that is widely used and supports various container runtimes. We collected 8,862 issue reports from the Kubernetes repository.[5] These issue reports had 129 labels.

TABLE 2. Numbers of issue reports used in the experiments.

Data for Fine-tuning[1]		VS Code	Kubernetes	Flutter	Roslyn	Ansible
Total	#Labels	454	129	344	219	170
	#Issues	63,232	8,862	41,486	9,394	4,719
Binary Classification	#Labels	2	2	2	2	2
	#Issues	20,972	5,912	14,098	4,204	3,660
Multilabel Classification	#Labels (top 10%)	46	13	35	22	17
	#Issues	61,020	8,613	40,734	9,241	4,714

[1] Date: From 2020-04-01 to 2023-06-21

3) FLUTTER

Flutter is a cross-platform GUI application framework that is used primarily to develop Android, iOS, Windows, Linux, and web applications. Flutter is a relatively recent framework; hence, historical data might be limited. We collected 41,486 issue reports from the Flutter repository.[6] These issue reports contained 344 labels.

4) ROSLYN

Roslyn is the.NET Compiler Platform, an open-source compiler and code analyzer for Microsoft's C# and Visual

[1]Due to our limited computing capacity, we conducted the pretraining process on a computer equipped with a 4x NVIDIA A100 Tensor Core GPU, 256 GB RAM, and a 64-core CPU, which took approximately 618.5 hours to complete.

[2]https://huggingface.co/gbkwon/issueBERT-large

[3]https://github.com/qja1998/pretrain_issue_bert

[4]https://github.com/microsoft/VS Code/issues

[5]https://github.com/kubernetes/kubernetes/issues

[6]https://github.com/flutter/flutter/issues

Basic languages. We collected 9,394 issue reports from the Roslyn repository.[7] Those issue reports contained 219 labels.

5) ANSIBLE

Ansible is an OSS tool that facilitates infrastructure as code, encompassing capabilities for software provisioning, configuration management, and application deployment. We collected 4,719 issue reports from the Ansible repository.[8] Those issue reports contained 170 labels.

D. EXPERIMENTAL PROCEDURE

This subsection describes the data preprocessing and experimental procedures used to answer the three research questions.

1) DATA PREPROCESSING

When we preprocessed the issue data, we focused on the title, body, and comments of the issue reports. We first tokenized the text. We then changed uppercase letters to lowercase letters. We removed stop words, emoticons, and tokens with more than 30 characters. After preprocessing, we also removed issue reports for which the total number of tokens was less than 5. When a single issue report had multiple comments, we concatenated them into a single sequence.

For our evaluation, we preserved the labels attached to each issue report.

2) RQ1. EXPERIMENT ON BINARY CLASSIFICATION

RQ1 measures the accuracy of a pretrained model on the task of classifying issue reports into *bugs* or *features*. To answer RQ1, we identified, selected, and consolidated the labels that corresponded to *bugs* and *features*. For example, in the VS Code project, we mapped the "bug" label to *bug* and the "feature-request" label to *feature*. In Kubernetes, the "kind/bug" label was mapped to *bug*, and the "kind/feature" label was mapped to *feature*.

The "Binary Classification" row of Table 2 shows the number of issue reports and labels for the task. To address the class imbalance problem, we applied downsampling to reduce the sample count of the majority category to that of the minority category. Because the *feature* class typically contains less data than does the *bug* class, we reduced the number of issue reports that belong to the *bug* class by selecting the most recent issue reports from the *bug* class.

Table 3 shows the number of issue reports in the downsampled dataset. As Table 3 shows, the number of issues per label was the same, and we utilized the following numbers of issue reports per project for the RQ1 experiment: VS Code (16,520), Kubernetes (3,338), Flutter (8,294), Roslyn (1,976), and Ansible (1,488).

We divided the dataset into training and evaluation sets at a ratio of 8:2. By using the training data, we fine-tuned each pretrained model. We then applied the pretrained model to the binary classification of issue reports and measured its classification accuracy.

In the experiment, we applied RoBERTa, CodeBERT, BERTOverflow, seBERT, and IssueBERT to the issue data of each project. We evaluated the classification accuracy of each fine-tuned model by referring to previous studies [30], [31]. The hyperparameter values were as follows. First, the loss function was CrossEntropyLoss. The number of epochs for fine-tuning was set to 5. The learning rate was set to 2e-5. The maximum length of an issue report was set to 300. The batch size was set to 32.

TABLE 3. Sampled data distribution per label.

Project	Label	Original Data	Downsampled Data
VS Code	Bug	12,712 (61%)	8,260 (50%)
	Feature	8,260 (39%)	8,260 (50%)
	Total	20,972	**16,520**
Kubernetes	Bug	4,243 (72%)	1,669 (50%)
	Feature	1,669 (28%)	1,669 (50%)
	Total	5,912	**3,338**
Flutter	Bug	9,951 (71%)	4,147 (50%)
	Feature	4,147 (29%)	4,147 (50%)
	Total	14,098	**8,294**
Roslyn	Bug	3,216 (76%)	988 (50%)
	Feature	988 (24%)	988 (50%)
	Total	4,204	**1,976**
Ansible	Bug	2,916 (80%)	744 (50%)
	Feature	744 (20%)	744 (50%)
	Total	3,660	**1,488**

3) RQ2. EXPERIMENT ON MULTILABEL CLASSIFICATION

RQ2 measures the accuracy of a pretrained model in classifying issue reports with multiple labels.

As Table 2 shows, the total number of labels in each project ranges from 129 to 454. If we were to consider all the labels, there would be too many classes, and the accuracy of predicting labels would be significantly low. Park et al. [7] reported that issue reports with labels in the top 10% in terms of usage yielded the highest F1-score for multilabel issue classification. Therefore, we decided to use issue reports with labels in the top 10% in terms of usage in each project. As a result, the 'Multilabel Classification' row of Table 2 shows the number of issue reports and labels for the RQ2 experiment.

If an issue report contained not only glabels belongingg to the top 10% in terms of usage but also other labels, we removed the labels that were not applicable. Table 4 shows the number of issue reports according to the number of labels per project after processing the data.

We divided the dataset into training and evaluation sets at a ratio of 8:2. We first fine-tuned each pretrained model using the training data. To implement the multilabeling classification task, we input the contextual information [CLS] token in the last layer of the pretrained model into a linear classifier. The shape of the output of the linear classifier was determined by the size of the hidden layers and the number

[7]https://github.com/dotnet/roslyn/issues
[8]https://github.com/ansible/ansible/labels

TABLE 4. Number of issue reports according to the number of labels.

Project \ #of label	1	2	3	4	5	6	7	≥ 8
VS Code	34,437	12,662	8,118	4,326	1,187	247	34	9
Kubernetes	931	2,606	3,854	1,142	77	2	1	-
Flutter	16,119	6,869	5,776	6,069	3,820	1,532	436	113
Roslyn	1,456	4,513	2,448	751	71	2	-	-
Ansible	582	397	1,267	1,305	763	318	70	12

of labels. A sigmoid function was used to represent the value of each label between 0 and 1. We applied a threshold from 0.1 to 0.9 to the value of each label and identified the labels attached to each issue report.

Second, we evaluated the multilabel classification accuracy with each threshold using the evaluation data. In the experiment, we fine-tuned RoBERTa, CodeBERT, BERTOverflow, seBERT, and IssueBERT on the issue data of each project. We evaluated the classification accuracy of each fine-tuned model.

The hyperparameters used to fine-tune the pretrained models were as follows: The loss function was BCEWithLogitsLoss. The number of epochs was set to 5. The learning rate was set to 2e-5. The maximum length of an issue report was set to 514. The batch size was set to 32.

4) RQ3. COMPARISON OF IssueBERT WITH STATE-OF-THE-ART APPROACHES

In RQ3, we compared the accuracy of IssueBERT with those of FastText and RoBERTa in multilabel issue classification tasks. FastText and RoBERTa achieved state-of-the-art performance in previous work [7], [8], [11]. For comparison, we considered to the performance of RoBERTa and IssueBERT at a threshold of 0.2, which yielded the best performance among the RQ2 experimental results. We additionally experimented with the FastText model on the same data used in the experiment for RQ2.

We set the hyperparameter K to -1, which indicates multiple labels, according to previous studies [7]. We applied it to a multilabel classification task and measured its accuracy.

E. EVALUATION METRICS

The metrics used for our evaluation were the precision, recall, F1-score, and MCC. These metrics were calculated with the respective equations.

For that, we first defined TP, TN, FP, and FN as follows:

- TP (number of true positives): The number of positive samples that are correctly predicted as positive.
- TN (number of true negatives): The number of negatives samples that are correctly predicted as negative.
- FP (number of false positives): The number of negative samples that are incorrectly predicted as positive.
- FN (number of false negatives): The number of positive samples that are incorrectly predicted as negative.

Based on these definitions, we defined the precision, recall, and F1-score as follows:

$$Precision = \frac{1}{n}\sum_{i=1}^{n}\frac{TP}{TP+FP} \tag{1}$$

$$Recall = \frac{1}{n}\sum_{i=1}^{n}\frac{TP}{TP+FN} \tag{2}$$

$$F1-score = \frac{2\cdot Precision\cdot Recall}{Precision+Recall} \tag{3}$$

For RQ1 (binary classification), we defined n as the number of classes predicted by the model. Additionally, researchers [32], [33] reported that the Matthews correlation coefficient (MCC) achieves higher prediction accuracy than does the F1-score or other metrics, and they recommended using the MCC metric for binary classification tasks. As a result, we adopted the MCC as the primary metric for our RQ1 experiment. We defined the MCC metrics as follows:

$$MCC = \frac{TP\cdot TN - FP\cdot FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}} \tag{4}$$

For RQ2 (multilabel classification), we defined n as the number of issue reports used in the evaluation. Previous studies [7], [8], [11] measured multilabel performance on a per-sample (issue report) basis, so we also measured performance on a per-sample basis in our experiment. In the case of multilabel classification, the precision (Equation 1) and recall (Equation 2) values can be calculated for each issue report. We adopted the F1-score as the primary metric for the RQ2 and RQ3 experiments. In this case, the F1-score was calculated as follows:

$$F1-score = \frac{1}{n}\sum_{i=1}^{n}\frac{2\cdot Precision_i\cdot Recall_i}{Precision_i+Recall_i} \tag{5}$$

$$F1-score_{micro} = \frac{2\cdot\sum_{i=1}^{n}Precision_i\cdot\sum_{i=1}^{n}Recall_i}{\sum_{i=1}^{n}Precision_i+\sum_{i=1}^{n}Recall_i} \tag{6}$$

We also calculated another accuracy metric for multilabel issue classification tasks, the micro average F1-score (F1-score$_{micro}$).[9] F1-score$_{micro}$ (Equation 6) calculates the precision and recall simultaneously by accumulating TP, FN, and TN for the entire test dataset. We considered the F1-score$_{micro}$ because it is not affected by label imbalance and is often used for multilabel classification tasks.

IV. EXPERIMENTAL RESULTS

This section discusses the experimental results for the three research questions.

A. BINARY CLASSIFICATION

In this section, we present the experimental results for Research Question 1 (RQ1), as shown in Figure 2.

[9]https://scikit-learn.org/stable/modules/generated/sklearn.metrics.f1_score.html

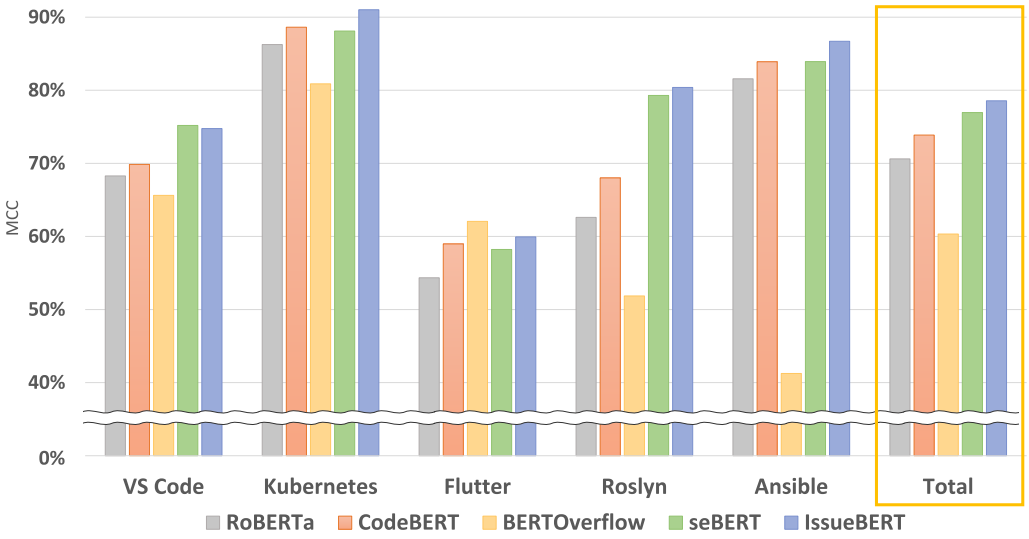


FIGURE 2. MCC of the models for binary classification.

TABLE 5. Results on the binary classification task.

Project	Model	Precision	Recall	F1-score	MCC
VS Code	RoBERTa	84.14	84.14	84.14	68.28
	CodeBERT	84.93	84.93	84.93	69.86
	BERTOverflow	84.17	81.51	81.14	65.62
	seBERT	**87.66**	**87.53**	**87.52**	**75.19**
	IssueBERT	87.50	87.26	87.24	74.76
Kubernetes	RoBERTa	93.14	93.11	93.11	86.25
	CodeBERT	94.32	94.31	94.31	88.63
	BERTOverflow	90.76	90.12	90.08	80.88
	seBERT	94.09	94.01	94.01	88.1
	IssueBERT	**95.66**	**95.36**	**95.35**	**91.02**
Flutter	RoBERTa	77.17	77.17	77.17	54.34
	CodeBERT	79.52	79.46	79.45	58.97
	BERTOverflow	**81.28**	**80.78**	**80.71**	**62.06**
	seBERT	79.12	79.1	79.09	58.22
	IssueBERT	80.06	79.88	79.85	59.94
Roslyn	RoBERTa	82.07	80.56	80.32	62.61
	CodeBERT	84.43	83.59	83.48	68.01
	BERTOverflow	80.25	72.22	70.25	51.85
	seBERT	89.65	89.65	89.65	79.29
	IssueBERT	**90.24**	**90.15**	**90.15**	**80.39**
Ansible	RoBERTa	90.97	90.6	90.58	81.57
	CodeBERT	91.95	91.95	91.95	83.9
	BERTOverflow	76.97	65.77	61.81	41.25
	seBERT	91.98	91.95	91.94	83.92
	IssueBERT	**93.41**	**93.29**	**93.28**	**86.70**
Total	RoBERTa	85.50	85.12	85.07	70.61
	CodeBERT	87.03	86.85	86.82	73.87
	BERTOverflow	82.69	78.08	76.80	60.33
	seBERT	88.50	88.45	88.44	76.95
	IssueBERT	**89.37**	**89.19**	**89.17**	**78.56**

Figure 2 shows that IssueBERT outperformed the other models across all projects, followed by seBERT. The IssueBERT model outperformed the other three models on three projects, and it performed second best on the other two projects. Both IssueBERT and seBERT were trained on issue data. The difference between the two models in terms of data is that seBERT was trained on data from a wider range of sources, including not only GitHub issues but also Stack Overflow posts, Jira issues, and GitHub commit messages.

In contrast, BERTOverflow had the lowest accuracy on all projects except for one. BERTOverflow achieved the lowest accuracy on four projects, namely, VS Code, Kubernetes, Roslyn, and Ansible, with the highest accuracy on the Flutter project. BERTOverflow was trained only on StackOverflow

data. Different training data could affect the issue report classification accuracy.

Moreover, RoBERTa and CodeBERT performed at the middle level. Notably, these two models were trained on much larger quantities of data than the other models. Interestingly, RoBERTa performed moderately well despite not being trained on specific software engineering data. CodeBERT was trained on both code data and natural language descriptions, as described earlier. Training on dual data sources might have improved the accuracy of CodeBERT.

Table 5 shows detailed information corresponding to Figure 2. The ''Project'' column in Table 5 represents the specific projects targeted for the experiments. The ''Model'' column indicates the kinds of pretrained models employed for each project. The accuracy evaluation metrics, including precision, recall, F1-score, and MCC, are presented in separate columns, with all the values expressed as percentages (%) and rounded to two decimal places. Additionally, we highlighted the highest scores for each project in bold and underlined the second-best scores.

Overall, Table 5 shows that the precision, recall, and F1-score trends are similar to the trend in the MCC. For example, in the whole row, IssueBERT yielded the highest accuracy with 89.37%, 89.19%, 89.17%, and 78.56% for the precision, recall, F1-score, and MCC, respectively. seBERT yielded the second-highest accuracy, with 88.50%, 88.45%, 88.44%, and 76.95% for precision, recall, F1-score, and MCC, respectively. CodeBERT and RoBERTa obtained the 3rd and 4th highest classification accuracies, respectively. BERTOverflow achieved the lowest classification accuracy.

IssueBERT achieved the highest precision, recall, F1-score, and MCC on three projects: Kubernetes, Roslyn, and Ansible. In the case of VS Code, seBERT achieved the highest accuracy across all the metrics, while IssueBERT achieved the second-highest accuracy. In the case of Flutter, BERTOverflow achieved the highest accuracy across all the metrics, while IssueBERT achieved the second-highest accuracy. Nevertheless, BERTOverflow was ranked the lowest across all the projects except Flutter.

Overall, IssueBERT performed the best, followed by seBERT. However, in one case, seBERT outperformed IssueBERT. Additionally, in this experiment, the recall, precision, and F1-score followed trends similar to that of the MCC, suggesting that no model performed particularly well in terms of any specific metric.

Therefore, the results measured by the four different metrics in Table 5 lead to the same conclusions as the results in Figure 2. With respect to the data used by the pretrained models, the best-performing IssueBERT was trained on issue-related data. In this regard, models trained with issue domain-specific data may perform better because they better understand the issue domain.

For the binary classification task, the issue classification task for each pretrained model yields only one value for each metric. Each label is classified as a 'bug' or 'feature.' TP, FP, TN, and FN across the entire dataset for each project are used to calculate the MCC and other metrics. Therefore, we could not apply statistical validation to this experiment.

Summary: On all the projects, IssueBERT outperformed the other pretrained models. Since these pretrained models learn from issue domain-specific data, they may better understand the issue classification task.

B. MULTILABEL CLASSIFICATION

In this section, we present the experimental results for Research Question 2 (RQ2), as shown in Figure 3. When we averaged the F1-scores of the five projects, IssueBERT performed best, with seBERT, RoBERTa, and CodeBERT performing in the middle and BERT Overflow performing the worst.

IssueBERT performed well even though it learned from less data than did the other pretrained models. One of the reasons is because IssueBERT was pretrained on issue data and because the vocabularies and contexts of the training data correspond to the tasks of classifying issue reports with multiple labels.

Additionally, IssueBERT and seBERT were trained on a BERT_{large} model with more layers and parameters, while the other models were trained on BERT-based models. Therefore, IssueBERT and seBERT learned richer representations than did CodeBERT and BERTOverflow. Additionally, both models learned from issue data, which could also improve their accuracy on issue classification tasks.

Overall, the accuracy of RoBERTa was similar to that of CodeBERT. This similarity is interesting because RoBERTa was not trained with software engineering domain-specific data. It could be conjectured that RoBERTa was trained with an enormous amount of data, which might have positively affected the issue classification task accuracy.

BERTOverflow still achieved the lowest accuracy across projects, even for the multilabel issue classification task. BERTOverflow was trained on the StackOverflow data, which are data from a community that aims to provide questions and answers related to programming. In contrast, the GitHub issue management system maintains bug, feature, enhancement, and question reports that arise when using and maintaining open-source projects. Therefore, we conjecture that the characteristics of the StackOverflow data differ from those of the issue data. The different data characteristics might have affected the accuracy on the issue classification task.

Table 6 shows detailed information corresponding to Figure 3. Figure 3 shows the model accuracy for threshold values between 0.1 and 0.9, where the best model accuracy is obtained when the threshold is 0.2. Table 6 presents the precision, recall, F1-score, and F1-score_{micro} with a threshold value of 0.2. The precision, recall, F1-score, and F1-score_{micro} are all expressed as percentages (%) and rounded to two decimal places. For example, for the Kubernetes project, IssueBERT yielded the highest accuracy with 79.7%, 89.2%,

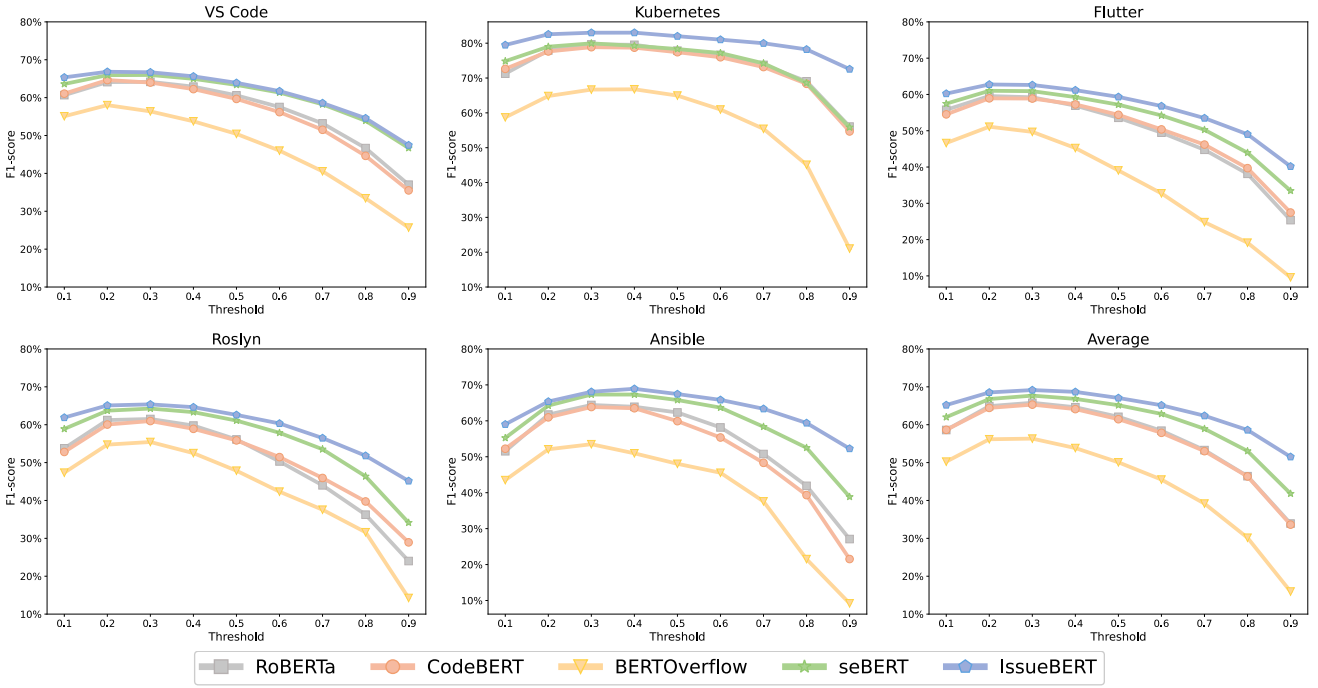


FIGURE 3. F1-scores obtained with various model thresholds for multilabel classification.

TABLE 6. Results on the multilabel classification task with a threshold of 0.2.

Project	Model	Precision	Recall	F1-score	F1-score$_{micro}$
VS Code	RoBERTa	62.34	72.94	64.1	61.68
	CodeBERT	63.78	72.22	64.64	62.06
	BERTOverflow	59.01	63.14	58.02	55.49
	seBERT	64.63	**73.77**	65.93	63.46
	IssueBERT	**66.29**	73.62	**66.83**	**64.67**
Kubernetes	RoBERTa	72.58	88.8	77.91	79.01
	CodeBERT	73.34	86.67	77.63	78.91
	BERTOverflow	56.73	81.78	64.83	66.0
	seBERT	74.5	88.35	79.0	79.95
	IssueBERT	**79.7**	**89.2**	**82.56**	**83.26**
Flutter	RoBERTa	56.04	69.96	59.56	58.76
	CodeBERT	53.88	72.55	58.93	57.56
	BERTOverflow	46.71	64.24	51.11	50.29
	seBERT	56.7	**72.73**	61.01	60.3
	IssueBERT	**59.87**	72.24	**62.75**	**61.75**
Roslyn	RoBERTa	57.28	71.86	61.22	61.28
	CodeBERT	55.43	71.5	60.04	60.22
	BERTOverflow	51.13	65.05	54.74	55.34
	seBERT	60.59	72.97	63.71	64.06
	IssueBERT	**61.64**	**74.98**	**65.09**	**65.26**
Ansible	RoBERTa	55.28	77.72	61.74	63.16
	CodeBERT	54.44	77.67	60.97	62.91
	BERTOverflow	44.8	68.06	56.02	54.29
	seBERT	56.57	82.12	**63.88**	65.58
	IssueBERT	**58.09**	**82.42**	**63.88**	**66.57**
Total	RoBERTa	60.7	76.26	64.91	64.78
	CodeBERT	60.17	76.12	64.44	64.33
	BERTOverflow	51.68	68.45	56.16	56.28
	seBERT	62.6	77.99	66.78	66.67
	IssueBERT	**65.12**	**78.49**	**68.52**	**68.3**

82.56%, and 83.26% for the precision, recall, F1-score, and F1-score$_{micro}$, respectively.

Here, we note that there are no significant differences in the performances of the metrics F1-score and F1-score$_{micro}$ in measuring the issue report classification accuracy, as shown in Table 6. Therefore, we discuss the F1-score$_{micro}$ for the experimental results in Table 6, and we discuss our result based on the F1-score in other places in this paper.

In total, IssueBERT yielded the highest accuracy with 65.12%, 78.49%, 68.52%, and 68.3% for precision, recall, F1-score, and F1-score$_{micro}$, respectively. In addition, IssueBERT achieved the best F1-score and F1-score$_{micro}$ for all five projects. seBERT yielded the second-highest accuracy with 62.6%, 77.99%, 66.78%, and 66.67% for precision, recall, F1-score, and F1-score$_{micro}$, respectively. RoBERTa and CodeBERT showed the 3rd and 4th highest classification accuracies, respectively. BERTOverflow achieved the lowest classification accuracy.

We conducted a statistical test to determine whether there were significant performance differences between IssueBERT and the other models. We first collected the F1-scores for the issue reports of the five projects. We then applied the Wilcoxon signed rank test to the data. Table 7 shows the results. We found that the performance of IssueBERT differed significantly from those of the other models, with p values $< 2.2^{-16}$ for RoBERTa, CodeBERT, and BERTOverflow, and p values $< 6.5^{-13}$ for seBERT.

In the case of IssueBERT, issue domain data were used to pretraining, fine-tune, and evaluate the model. It differs from other models pretrained on software engineering domain data (seBERT and BERTOverflow) or code domain data (CodeBERT). Conducting pretraining and fine-tuning with the same domain data seems to be the reason that IssueBERT had the best accuracy.

TABLE 7. Result of statistical tests.

	Model	p value
IssueBERT	RoBERTa, CodeBERT, BERTOverflow	2.2^{-16}(<0.05)
	seBERT	6.5^{-13}(<0.05)

Summary: Overall, IssueBERT outperformed the other pretrained models, with seBERT exhibiting good accuracy as the second-best model, indicating that the similarity between the pretrained and target domains has a greater impact on the multilabel issue classification accuracy than does the quantity of data used for model pretraining.

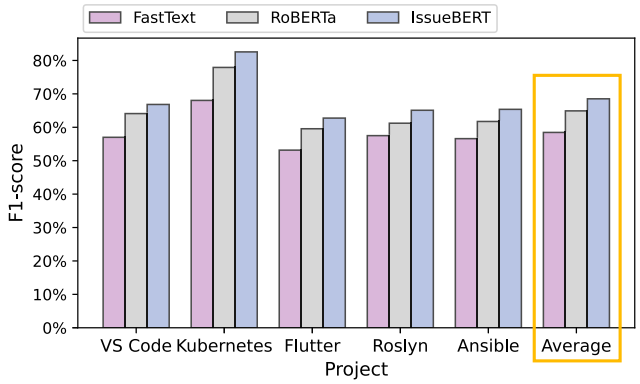


FIGURE 4. F1-scores of FastText, RoBERTa, and IssueBERT at a threshold of 0.2.

TABLE 8. Comparison of the state-of-the-art models and IssueBERT in terms of multilabel classification accuracy at a threshold of 0.2.

Project	FastText[1]	RoBERTa[2]	IssueBERT[3]	difference of 1 and 3	difference of 2 and 3
VS Code	57.0	64.1	66.83	+9.83%	+2.73%
Kubernetes	68.04	77.91	82.56	+14.52%	+4.65%
Flutter	53.16	59.56	62.75	+9.59%	+3.19%
Roslyn	57.5	61.22	65.09	+7.59%	+3.87%
Ansible	56.6	61.74	65.37	+8.77%	+3.63%
Total	58.46	64.91	68.52	+10.06%	+3.61%

C. COMPARISON WITH THE STATE-OF-THE-ART METHODS

In this section, we present the experimental results for Research Question 3 (RQ3), as shown in Figure 4 and Table 8. Here, we compare the accuracy of our method (IssueBERT) with those of the FastText and RoBERTa methods used in previous studies [7], [8], [11] related to issue label classification tasks.

As shown by the trend in Figure 4, IssueBERT performed the best among all the tested models, followed by RoBERTa and FastText. For VS Code, the F1-score of IssueBERT was 10.06% greater than that of FastText and 3.61% greater than those of RoBERTa. Similarly, the F1-score of IssueBERT was consistently greater than those of FastText and RoBERTa across other projects, such as Kubernetes, Flutter, Roslyn, and Ansible.

Overall, IssueBERT outperformed FastText by a maximum of 14.52% and a minimum of 7.59% and outperformed RoBERTa by a maximum of 4.65% and a minimum of 2.73%. In terms of the total average, the accuracy of IssueBERT was 10.06% better than that of FastText and 3.61% better than that of RoBERTa.

In addition, we conducted a statistical test to evaluate the significant differences between the IssueBERT and RoBERTa models. We used the Wilcoxon signed rank test. We used all the test data for each project and compared the models' F1-scores. The p value of the statistical test was less than 2.2^{-16}(<0.05) as shown in Table 7.

Summary: Overall, IssueBERT realized higher issue classification task accuracy than the state-of-the-art approaches. This improvement could have been because IssueBERT uses contextual word embeddings and was pretrained on issue domain data.

V. ADDITIONAL EXPERIMENTS

In this section, we report two additional experiments that were conducted to determine the characteristics of IssueBERT and other pretrained models.

A. PREDICTION OF MISSING WORDS IN SENTENCES

We qualitatively investigated the ability of IssueBERT and seBERT to predict missing words in sentences, as Mosel et al. [12] did to investigate the capability of

TABLE 9. Word predictions by IssueBERT and seBERT [12] for [MASK] tokens.

	Sentence	Expectation	IssueBERT		seBERT	
			Prediction	Prob.	Prediction	Prob.
1	Does this issue occur when all extensions are [MASK]?	disabled	**disabled**	0.8804	**disabled**	0.9999
			uninstalled	0.0251	closed	0.00002
			enabled	0.0243	inactive	0.000008
2	Steps to [MASK].	reproduce	**reproduce**	0.4123	**reproduce**	0.9981
			test	0.0446	replicate	0.0006
			readme	0.0400	recreate	0.0005
3	Process doesn't terminate after [MASK] app when run in release mode on Windows.	closing	**closing**	0.2830	**closing**	0.3894
			exit	0.1125	packages	0.1401
			exiting	0.0947	dependencies	0.0888
4	Please review and approve my [MASK] request related to this issue.	pull	**pull**	0.9902	**pull**	0.9591
			merge	0.0039	merge	0.0215
			change	0.0009	change	0.0043
5	I've [MASK] an issue to track this bug.	opened	created	0.6091	created	0.4078
			opened	0.1614	**opened**	0.3035
			added	0.0819	added	0.0974
6	The [MASK] is a default label that indicates a need for improvements or additions to documentation.	Documentation	**documentation**	0.4245	issue	0.1950
			issue	0.0442	label	0.1113
			following	0.0427	pr	0.0752
7	The [MASK] wanted is a default label that indicates that a maintainer wants help on an issue or pull request.	help	**help**	0.9940	**help**	0.9999
			maintainers	0.0017	assistance	0.00005
			maintainer	0.0016	maintainer	0.00003
8	[MASK] is a proprietary issue tracking product developed by Atlassian that allows bug tracking and agile project management.	Jira	**jira**	0.3974	**jira**	0.4954
			sentry	0.1387	there	0.0837
			this	0.1312	zenhub	0.0763
9	[MASK] is an internet hosting provider for software development and version control using Git.	Github, Gitlab	**github**	0.4832	**github**	0.4233
			this	0.1054	it	0.1490
			gitlab	0.0412	**gitlab**	0.0332
10	In object-oriented programming, a [MASK] is an extensible program-codetemplate for creating objects.	class	**class**	0.5593	**class**	0.4126
			module	0.0592	constructor	0.1765
			program	0.0382	factory	0.1066

seBERT. Table 9 shows the results. In the table, the indices from 1 to 7 indicate the sentences that we extracted from issue reports in GitHub. The indices 8 to 10 indicate the sentences that we obtained from Mosel et al.'s work [12]. The predicted answers that matched the expected answers are marked in bold.

We first reviewed sentences 1 to 8 in Table 9. For the masked word in sentence 1, IssueBERT correctly predicted the word *disabled*, as did seBERT. Similarly, IssueBERT and seBERT demonstrated comparable accuracies for sentences 2, 4, 5, and 7. Moreover, for the masked words in sentences 3 and 6, IssueBERT correctly predicted closing and documentation, respectively, while seBERT did not.

IssueBERT, our pretrained model specialized in the issue-related domain, performed well in filling in the masks of sentences commonly used in GitHub issue reports. Its accuracy was comparable to that of seBERT and, in some cases, even better. Interestingly, IssueBERT was trained on a limited dataset (approximately thirteen times smaller than that on which seBERT was trained). We can infer that pretraining with specialized issue data helps predict masked words in issue-relevant sentences.

Additionally, IssueBERT showed comparable accuracy to seBERT in most cases, as presented in Mosel et al.'s work [12]. In sentences 8 to 10, IssueBERT predicted the correct word in the same order as seBERT did. Therefore,

we conclude that a pretrained model based on issue data performs exceptionally well in the software domain.

B. CROSS-PROJECT CAPABILITIES FOR PREDICTING ISSUE LABELS

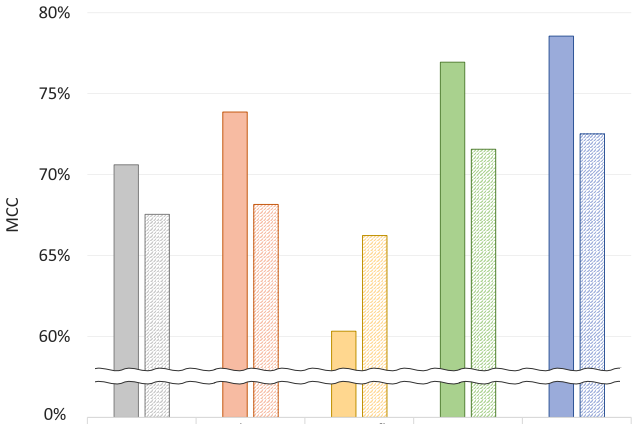
Readers might wonder about the cross-project capabilities of pretrained models. However, as our main problem is predicting custom labels, which differ among projects, this question does not align well with our problem. To clarify, we present the experimental results for binary classification with the same labels across projects and multilabel classification with different labels across projects as follows.

1) BINARY CLASSIFICATION

We first evaluated the cross-project capabilities of IssueBERT by training a binary classifier with cross-project datasets. For that purpose, we combined the training datasets of five projects and used the datasets to fine-tune a pretrained model. There were 25,293 issue reports in the datasets. Afterward, we evaluated the accuracy of the cross-project finetuned model on the test dataset for the five projects.

Figure 5 shows the evaluation results. As shown in Figure 5, the per-project finetuned models of RoBERTa, CodeBERT, seBERT, and IssueBERT outperformed cross-project finetuned models, while BERTOverflow did not. Despite predicting the same labels across projects, the per-project fine-tuned models outperformed the cross-project finetuned model by a margin of 3.06 - 6.04% in terms of F1-score.

One exception was BERTOverflow, where the cross-project fine-tuned model outperformed the per-project fine-tuned model.

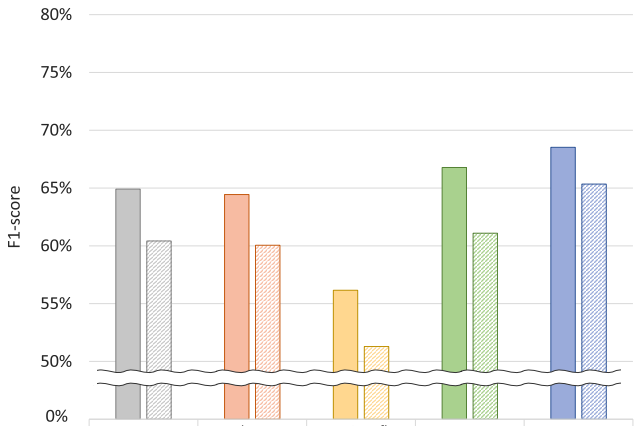


	RoBERTa	CodeBERT	BERTOverflow	seBERT	IssueBERT
Average	70.61	73.87	60.33	76.95	78.56
CrossProj	67.55	68.16	66.24	71.57	72.52

FIGURE 5. Comparison of the binary classification results of IssueBERT and IssueBERT_CP (CP: Cross-Projects).

2) MULTILABEL CLASSIFICATION

Second, we evaluated the cross-project capabilities of IssueBERT by training a multilabel classifier with cross-project



	RoBERTa	CodeBERT	BERTOverflow	seBERT	IssueBERT
Average	64.91	64.44	56.16	66.78	68.52
CrossProj	60.42	60.05	51.29	61.09	65.34

FIGURE 6. Comparison of the multilabel classification results of IssueBERT and IssueBERT_CP (CP: Cross-Projects).

datasets. For that purpose, we collected the training datasets of the five projects and used the datasets to fine-tune the cross-project model. There were 124,322 issue reports in the datasets, and 133 labels were used. Afterward, we evaluated the accuracy of the cross-project fine-tuned model with the test dataset for each project.

Figure 6 shows the evaluation results. All the per-project finetuned models outperformed the cross-project fine-tuned models by a margin of 3.18-5.69%. When we compared the results in Figure 6 with the binary classification results in Figure 5, we found that the trends were the same, except for that of BERTOverflow. In BERTOverflow, the per-project fine-tuned model outperformed the cross-project fine-tuned model.

The reason for the poor accuracy of the cross-project fine-tuned model in multilabel issue classification tasks could be that developers use different custom labels for each project. Therefore, if a model is trained across projects, the number of labels to be predicted increases, and the accuracy on multilabel classification tasks decreases accordingly. The issue here is that the same trend also appeared in our binary issue classification tasks. Thus, we propose that project issue data can be quite different, which might impact the issue classification task accuracy.

VI. DISCUSSION

Researchers have proposed many methods for classifying issue reports, including deep learning methods and pretrained models. However, the proposed methods have not achieved sufficient issue classification accuracy for practical use. Therefore, we explored several software engineering-specific pretrained models through experiments to improve the classification accuracy. As a result, we found that IssueBERT and seBERT, which were trained on issue data, performed better than did CodeBERT and BERTOverflow, which were trained on code or StackOverflow data, and RoBERTa, which

was trained on general data. Our results show that a model pretrained on issue data is suitable for issue classification tasks.

As large-scale data can be used to increase the reliability of experimental evaluations, there could be a question as to why we conducted our experiments on a project-by-project basis. There are two reasons for this. The first reason is that developers classify issue reports relevant to a project on which they work. The second reason is that developers create custom labels for each project on which they work. These custom labels vary by project. For these reasons, we conducted our experiments on a project-by-project basis.

The implications of this study are as follows. First, the accuracy on automated issue classification tasks should be sufficiently high so that developers can adopt automatic classification methods and tools in practice.

In this paper, we investigated whether and to what extent using a software engineering-specific pretrained model could improve the issue classification task accuracy. We found that the best-performing models were IssueBERT and seBERT, which were trained on issue data. Therefore, we expect that developers can achieve higher accuracy on issue classification tasks by adopting these specialized pretrained models.

Second, our research identified a pretrained model that performs well on issue classification tasks, and we reported that the model can lead to an improvement in accuracy of 3.61%. Based on our results, researchers can adapt the pretrained model to achieve improved accuracy. For example, researchers could consider what features should be added to the pretrained model or how they should fine-tune and further train these models. Additionally, it is possible to build a better-performing model for issue categorization tasks by increasing the data size. Moreover, researchers could use a good-performing pretrained model for other issue-relevant tasks, such as summarizing or predicting issues.

Third, we developed IssueBERT for companies, which may be useful for issue classification tasks. Our issue categorization model, IssueBERT, is available at https://huggingface.co/gbkwon/issueBERT-large. Companies can use our issue categorization model to classify issues in their projects. By automatically classifying issues, companies can reduce developers' workload in classifying issue reports while maintaining issues more systematically.

VII. THREATS TO VALIDITY

We report threats to internal and external validity.

A. THREATS TO INTERNAL VALIDITY

Several factors, such as the pretrained models, data, and hyperparameters, might have affected the experimental results. First, to create IssueBERT, we selected the masked language model task (MLM) with a BERT-large model. By training the model on a larger dataset, the accuracy of IssueBERT increased.

Second, when pretraining IssueBERT, we constructed the training data by concatenating the title and body of each issue report. However, when classifying issue reports, we additionally used the comments of the issue reports after tokenization with 512 tokens. Differences in the data might have negatively affected the accuracy of IssueBERT.

Third, for the hyperparameters, we used the values recommended in the BERT-based pretraining study [30]. To minimize the impact of the hyperparameters, we did our best to set parameter values similar to those of the pretrained models used in our experiments. Nevertheless, we acknowledge that we did not perform hyperparameter tuning. We experimented with several different settings but found that these different settings did not reverse the overall trend in the experimental results during our experiment.

B. THREATS TO EXTERNAL VALIDITY

There are threats to the validity of our experimental results. First, we report the experimental results on five open-source projects, which limits the generalizability of the experimental results. In this respect, to select representative projects, we selected projects with high numbers of stars, forks, and issue reports.

Second, we selected open-source projects from GitHub, which limits the generalizability of the experimental results as well. Using different datasets from different issue management systems could lead to different experimental results. However, GitHub is a very popular and common platform for open-source projects, so it is reasonable in terms of availability and accessibility to use GitHub issue data.

Third, as we focused on issue classification tasks when evaluating IssueBERT, there might be limitations in generalizing the model's performance. To comprehensively assess the performance of the proposed IssueBERT model, tasks such as issue summarization, response generation, similarity analysis, and keyword extraction can be considered. Future research could evaluate the performance of IssueBERT across various types of tasks related to issue reports.

Fourth, additional pretrained models are available. For example, UniXcoder [34], CodeT5 [35], and Codex [36] are models that are pretrained on code data. We used CodeBERT as the representative model pretrained on code data, but different code-based pretrained models could yield different accuracies. However, we did not include these models because we were interested in the accuracy of models pretrained on issue data. According to our experimental results, code-based pretrained models trained on data from different sources are unlikely to produce higher accuracy than IssueBERT.

VIII. CONCLUSION

In this paper, we developed IssueBERT, which was pretrained on only issue data, and we conducted comparative

experiments with RoBERTa, CodeBERT, BERTOverflow, seBERT and IssueBERT. We investigated which pretrained model yields the highest accuracy for issue classification tasks. Our experimental results showed that IssueBERT, which we developed, yielded the highest accuracy in both binary and multilabel issue classification tasks. seBERT performed second best. RoBERTa and CodeBERT performed moderately well. BERTOverflow performed poorly. We found two interesting points, here. First, IssueBERT was pretrained on much less data than were the other pretrained models. Second, BERTOverflow, reported as an outperforming model in Mosel et al.'s work [12], performed poorly in our experiments. Based on our observations, we attribute the outstanding performance of IssueBERT to the homogeneity between the pretraining data and the downstream task.

In our future work, based on our findings, we will continue to construct pretrained models that perform best in issue classification tasks. In this paper, we used issue data collected from projects with 5,000 stars and 10,000 issues. However, additional issue data are available. Therefore, we can construct pretrained models with more issue data. We could also consider training a large language model (LLM) with more parameters or constructing a different structure of the pretrained model that performs best for issue classification tasks. However, given computing resources, it may be more practical to lightweight an LLM and to create a model specific to this issue classification domain.

REFERENCES

[1] S. J. Sohrawardi, I. Azam, and S. Hosain, "A comparative study of text classification algorithms on user submitted bug reports," in *Proc. 9th Int. Conf. Digit. Inf. Manage. (ICDIM)*, Sep. 2014, pp. 242–247.

[2] N. Pandey, D. K. Sanyal, A. Hudait, and A. Sen, "Automated classification of software issue reports using machine learning techniques: An empirical study," *Innov. Syst. Softw. Eng.*, vol. 13, no. 4, pp. 279–297, Dec. 2017.

[3] Q. Fan, Y. Yu, G. Yin, T. Wang, and H. Wang, "Where is the road for issue reports classification based on text mining?" in *Proc. ACM/IEEE Int. Symp. Empirical Softw. Eng. Meas. (ESEM)*, Nov. 2017, pp. 121–130.

[4] N. Pandey, A. Hudait, D. K. Sanyal, and A. Sen, "Automated classification of issue reports from a software issue tracker," in *Progress in Intelligent Computing Techniques: Theory, Practice, and Applications*, vol. 1. Cham, Switzerland: Springer, 2016, pp. 423–430.

[5] Y. Zhu, M. Pan, Y. Pei, and T. Zhang, "A bug or a suggestion? An automatic way to label issues," 2019, *arXiv:1909.00934*.

[6] J. Kim and S. Lee, "An empirical study on using multi-labels for issues in GitHub," *IEEE Access*, vol. 9, pp. 134984–134997, 2021.

[7] D. Park, H. Cho, and S. Lee, "Classifying issues into custom labels in GitBot," in *Proc. 4th Int. Workshop Bots Softw. Eng.*, May 2022, pp. 28–32.

[8] J. Heo, D. Park, and S. Lee, "Comparison of multi-label classification performance using roberta fine-tuning for GitHub issue report management," in *Proc. Korea Comput. Congr. (KCC)*. Jeju Island, South Korea: KIISE, Jun. 2022.

[9] J. Wang, X. Zhang, and L. Chen, "How well do pre-trained contextual language representations recommend labels for GitHub issues?" *Knowl.-Based Syst.*, vol. 232, Nov. 2021, Art. no. 107476.

[10] J. Wang, X. Zhang, L. Chen, and X. Xie, "Personalizing label prediction for GitHub issues," *Inf. Softw. Technol.*, vol. 145, May 2022, Art. no. 106845.

[11] J. Heo and S. Lee, "An empirical study on the performance of individual issue label prediction," in *Proc. IEEE/ACM 20th Int. Conf. Mining Softw. Repositories (MSR)*, May 2023, pp. 228–233.

[12] J. von der Mosel, A. Trautsch, and S. Herbold, "On the validity of pre-trained transformers for natural language processing in the software engineering domain," *IEEE Trans. Softw. Eng.*, vol. 49, no. 4, pp. 1487–1507, Apr. 2023.

[13] Z. Feng, D. Guo, D. Tang, N. Duan, X. Feng, M. Gong, L. Shou, B. Qin, T. Liu, D. Jiang, and M. Zhou, "CodeBERT: A pre-trained model for programming and natural languages," 2020, *arXiv:2002.08155*.

[14] J. Tabassum, M. Maddela, W. Xu, and A. Ritter, "Code and named entity recognition in StackOverflow," in *Proc. 58th Annu. Meeting Assoc. Comput. Linguistics*, 2020, pp. 4913–4926.

[15] 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*.

[16] L. Zhifang, W. Kun, Z. Qi, L. Shengzong, Z. Yan, and H. Jianbiao, "Classification of open source software bug report based on transfer learning," *Expert Syst.*, vol. 41, no. 5, May 2024, Art. no. e13184.

[17] X. Xie, Y. Su, S. Chen, L. Chen, J. Xuan, and B. Xu, "MULA: A just-in-time multi-labeling system for issue reports," *IEEE Trans. Rel.*, vol. 71, no. 1, pp. 250–263, Mar. 2022.

[18] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *Trans. Assoc. Comput. Linguistics*, vol. 5, pp. 135–146, Dec. 2017.

[19] D. Guo, S. Ren, S. Lu, Z. Feng, D. Tang, S. Liu, L. Zhou, N. Duan, A. Svyatkovskiy, S. Fu, M. Tufano, S. Kun Deng, C. Clement, D. Drain, N. Sundaresan, J. Yin, D. Jiang, and M. Zhou, "Graph-CodeBERT: Pre-training code representations with data flow," 2020, *arXiv:2009.08366*.

[20] J. Lin, Y. Liu, Q. Zeng, M. Jiang, and J. Cleland-Huang, "Traceability transformed: Generating more accurate links with pre-trained BERT models," in *Proc. IEEE/ACM 43rd Int. Conf. Softw. Eng. (ICSE)*, May 2021, pp. 324–335.

[21] M. A. Hadi and F. H. Fard, "Evaluating pre-trained models for user feedback analysis in software engineering: A study on classification of app-reviews," *Empirical Softw. Eng.*, vol. 28, no. 4, p. 88, Jul. 2023.

[22] S. Troshin and N. Chirkova, "Probing pretrained models of source codes," in *Proc. 5th BlackboxNLP Workshop Analyzing Interpreting Neural Netw. (NLP)*, 2022, pp. 371–383.

[23] Z. Shaheen, G. Wohlgenannt, and E. Filtz, "Large scale legal text classification using transformer models," 2020, *arXiv:2010.12871*.

[24] Z. Guo, L. Zhu, and L. Han, "Research on short text classification based on RoBERTa-TextRCNN," in *Proc. Int. Conf. Comput. Inf. Sci. Artif. Intell. (CISAI)*, Sep. 2021, pp. 845–849.

[25] S. Minaee, N. Kalchbrenner, E. Cambria, N. Nikzad, M. Chenaghlu, and J. Gao, "Deep learning-based text classification: A comprehensive review," *ACM Comput. Surv.*, vol. 54, no. 3, pp. 1–40, Apr. 2022.

[26] X. Zhou, D. Han, and D. Lo, "Assessing generalizability of CodeBERT," in *Proc. IEEE Int. Conf. Softw. Maintenance Evol. (ICSME)*, Sep. 2021, pp. 425–436.

[27] C. Pan, M. Lu, and B. Xu, "An empirical study on software defect prediction using CodeBERT model," *Appl. Sci.*, vol. 11, no. 11, p. 4793, May 2021.

[28] E. Mashhadi and H. Hemmati, "Applying CodeBERT for automated program repair of Java simple bugs," in *Proc. IEEE/ACM 18th Int. Conf. Mining Softw. Repositories (MSR)*, May 2021, pp. 505–509.

[29] A. Trautsch and S. Herbold, "Predicting issue types with seBERT," in *Proc. IEEE/ACM 1st Int. Workshop Natural Lang.-Based Softw. Eng. (NLBSE)*, May 2022, pp. 37–39.

[30] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," 2018, *arXiv:1810.04805*.

[31] Z. Xinxi, "Single task fine-tune BERT for text classification," in *Proc. 2nd Int. Conf. Comput. Vis., Image, Deep Learn.*, Kunming, China, Oct. 2021, pp. 194–206.

[32] M. Gan, Z. Yücel, and A. Monden, "Neg/pos-normalized accuracy measures for software defect prediction," *IEEE Access*, vol. 10, pp. 134580–134591, 2022.

[33] D. Chicco and G. Jurman, "The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation," *BMC Genomics*, vol. 21, no. 1, pp. 1–13, Dec. 2020.

[34] D. Guo, S. Lu, N. Duan, Y. Wang, M. Zhou, and J. Yin, "UniXcoder: Unified cross-modal pre-training for code representation," 2022, *arXiv:2203.03850*.

[35] Y. Wang, W. Wang, S. Joty, and S. C. H. Hoi, "CodeT5: Identifier-aware unified pre-trained encoder–decoder models for code understanding and generation," 2021, *arXiv:2109.00859*.
[36] M. Chen, "Evaluating large language models trained on code," 2021, *arXiv:2107.03374*.

JUEUN HEO received the B.S. degree from the Department of Aerospace and Software Engineering, Gyeongsang National University, in 2022, and the M.S. degree from the Department of AI Convergence Engineering, Gyeongsang National University, in 2024, where she is currently pursuing the Ph.D. degree with the Department of AI Convergence Engineering. Her research interests include software engineering, artificial intelligence (AI), natural language processing, and classification.

GIBEOM KWON received the B.S. degree from the Department of Aerospace and Software Engineering, Gyeongsang National University, in 2024. His research interests include software engineering, artificial intelligence (AI), natural language processing, time series forecasting, and data science.

CHANGWON KWAK received the B.S. degree from the Department of Aerospace and Software Engineering, Gyeongsang National University, in 2022, and the M.S. degree from the Department of AI Convergence Engineering, Gyeongsang National University, in 2024. His research interests include software engineering, artificial intelligence (AI), natural language processing, multimodal deep learning, and data science.

SEONAH LEE (Member, IEEE) received the B.S. and M.S. degrees in computer science and engineering from Ewha Womans University, in 1997 and 1999, respectively, the M.S.E. degree from the School of Computer Science, Carnegie Mellon University, in 2005, and the Ph.D. degree from the School of Computer Science, KAIST, in 2013. She was a Software Engineer with Samsung Electronics, from 1999 to 2006. She was also a Research Professor with KAIST, from 2014 to 2015. Currently, she is an Associate Professor with the Department of Software Engineering and the Department of AI Convergence Engineering, Gyeongsang National University. Her research interests include software evolution, documentation updates, requirement traceability, software architecture, and data mining. She is a member of KIISE and KIPS.