Duplicate Bug Report Detection by Using Sentence Embedding and Faiss

Lee Sunho[1], Lee Seonah[1,*]

[1] *Gyeongsang National University, Gyeongsangnam-do, Jinju City, Republic of Korea*

Abstract

Duplicate issue reports in the issue management system can cause unnecessary work for users and developers and disrupt the progress of their work. Therefore, researchers have developed duplicate report detection techniques. However, most of those techniques require training pairs of duplicate reports as well as those of non-duplicate bug reports. To speed up the processing of such machine learning-oriented methods, we propose implementing duplicate report detection using the recent technologies Sentence Bert and Faiss. By using the Faiss library, we can quickly detect duplicate issue reports. We also evaluate our proposed approaches with two different experiments and discuss our future work.

Keywords

Bug report, Duplicate report detection, Sentence BERT, SBERT, Faiss, Machine learning

1. Introduction

Many developers collaborate on a single software project, working together with their team members to address various tasks. In the case of open source projects, who interested in the projects can easily participate in development, in addition to the designated team members. When many people are involved in a single project, they create bug reports to share problems of the software systems, information about their situations, the current state of development, and so on. In larger projects, tens of thousands of such bug reports can be daily created by reporters.

However, unnecessarily, duplicate bug reports are also created in this process. These duplicated bug reports can increase unnecessary workload and cause confusion about the progress of the tasks. To resolve these issues, it is necessary to proactively detect and eliminate duplicate bug reports. To identify duplicate bug reports, it is possible to check document similarity among those reports.

Researchers have proposed various methods to detect such duplicate bug reports. Those methods can broadly categorized into ML(Machine Learning) and IR(Information Retrieval). In the IR group, the methods use TF-IDF and cosine similarity [1, 2]. Others employed BM25F to measure the similarity between bug reports to query detecting duplicate bug reports [3, 19]. However, those approaches often fall short in understanding the context. In the ML group, methods used DCCNN (Dual Channel Convolutional Neural Networks) [4], GloVe[10, 11], MLP (Multi Layer Perceptron) [5], and self-attention [6]. Those methods aimed to comprehend the contexts for duplicate bug reports. However, there is a drawback when using those in that those methods require creating bug reports pairs for training and evaluation. Additionally, training deep learning models can be time-consuming.

We propose duplicate report detection using Sentence BERT(SBERT) and Faiss(Facebook AI Similarity Search). Faiss is a library that quickly finds a similar one among several documents. In this case, the approach can detect duplicate bug reports without extensively training on a pair dataset of duplicate bug reports.

In our experiments, we used only unstructured data, specifically the 'title' and 'body' of bug reports without any pairs of bug reports. We employed Sentence BERT to generate

ISE 2023: 2nd International Workshop on Intelligent Software Engineering, December 4, 2023, Seoul

* Corresponding Author

tjsgh5768@gmail.com (Sunho Lee); saleese@gnu.ac.kr (Seonah Lee);

embeddings for this data. Subsequently, we create an indexIDMap in Faiss, where the results of the embeddings are associated with the respective issue IDs. When a new issue is given, our proposed approach utilizes Faiss' search function to find duplicate bug reports in the indexIDMap.

In this paper, we ask three research questions. The first question is about the accuracy of detecting duplicate bug reports. The second question is about predicting “duplicate” labels for bug reports. The third question is the speed of searching and detecting duplicate bug reports. Our experimental results shows that, with regard to duplicate bug report detection, our proposed approach shows reasonable accuracy but not outperforms state-of-the-art approaches. However, as shown in the second question, our approach can be applied to a new problem of predicting “duplicate” labels for bug reports and shows a high prediction accuracy than state-of-the-art approaches. At last, our approach shows its speedy performance in detecting duplicate bug reports and predicting its labels.

Our contributions are as follows. First, we propose a method for quickly identifying duplicate bug reports using the newly developed Faiss library. Second, we collected real-world open-source data and reported the performance of our proposed method. We found that our system can rapidly detect duplicate reports. Our research results could help users and developers to identify duplicate reports and, by doing so, to avoid unnecessary tasks in advance.

The structure of this paper is as follows. Section 2 explains the basic concepts as the "BACKGROUND" section. Section 3 describes the proposed method and the structure of the model. Section 4 provides an explanation of the experimental approach. Section 5 presents the analysis of the model's results in "EXPERIMENTAL RESULTS AND ANALYSIS". Finally, Section 6 summarizes the findings of this paper and outlines future research directions.

2. Background

We will provide a summary of the terms and concepts used in this paper.

2.1. Duplicate bug reports

Software developers often collaborate with many individuals to enhance project performance and expedite completion via bug management systems. During software development, various bugs arise, and these bug reports are documented to facilitate communication among developers. Most bug reports are described in text form. Given that many people express bug reports in text, even the same bug report can be reported in different bug reports with different textual expressions. These bug reports are referred to as "duplicate bug reports." Duplicate Bug Reports lead to an increase in unnecessary work for developers, which makes it challenging to monitor the progress of tasks. Due to these bug reports, it is necessary to identify duplicate bug reports proactively. However, manually detecting duplicate bug reports requires a significant amount of time and effort.

Figure 1: Example duplicate bug report

2.2. Sentence Embedding

To process text on a computer, it's necessary to convert text into numerical representations. One way to do this is to assign a unique number to each word. The most simple one is One-Hot Encoding, which assigns unique numbers to words. However, such an approach doesn't capture the meaning of words in the context in which the words are used. To create numerical representations that capture contextual meaning, BERT is used. "BERT," introduced in a paper by Jacob Devlin in 2018 [8], is a language model that leverages a self-attention mechanism to transform text into vectors that encapsulate the meaning of context.

In 2019, Nils Reimers introduced "Sentence BERT (SBERT)" in a paper [7], which is a modified Language Model derived from BERT specifically designed for tasks involving the calculation of similarity between sentences. SBERT provides more appropriate sentence-level embeddings using techniques like the "Siamese architecture."

There are three main methods for representing sentences as vectors using SBERT:

- **Mean Pooling**: the sentence is input into BERT, and the average of the resulting vector is used as the representation of the sentence. This approach captures the influence of all words in the sentence.
- **Max Pooling**: Max Pooling uses the maximum value from the vector instead of the average. This approach gives more weight to the most significant words in the sentence.
- **CLS Token**: The BERT model's output contains a [CLS] token as its first value. This token encapsulates the summary embedding of the sentence. Using this [CLS] token as the vector representation effectively summarizes the sentence's meaning.

These different methods allow SBERT to create meaningful sentence embeddings, depending on the specific needs of the task at hand. In our approach, we adopted the Mean Polling method to consider the influence of all words in a sentence.

2.3. Faiss Library

Faiss (Facebook AI Similarity Search)[2] is a library developed by the Facebook AI research team. Faiss quickly finds similar vectors in large amounts of vectorized data or cluster vectorized data. To this end, Faiss uses vector quantization technology and indexing technology. Vector quantization is a technology that maps high-dimensional vectors into low-dimensional vectors. This reduces a memory usage and improves a searching speed. Indexing technology enables to find the most similar cluster in the clusters and compare a vector with the vectors in the matched cluster, rather than comparing it with all vectors.

Faiss makes it possible to quickly calculate the similarity between bug reports. It is particularly useful in detecting duplicate bug reports where two bug reports share the similar textual contents in a speedy way. Therefore, we used the Faiss Library to detect duplicate bug reporters, taking advantage of its capability of quickly finding similar documents.

2.4. Euclidean Distance

There are various metrics used to measure how similar two documents are, such as Cosine Similarity, Inner Product (IP) Similarity, and more. 'Euclidean Distance' is a metric that calculates the distance between two vectors. If the vector values of two documents are denoted as X and Y, the similarity between the two documents can be calculated based on the following formula (1).

$$Euclidean\ Distance = \sqrt{\sum_{i}^{k}(x_i - y_i)^2}\ (where, x_i \in X, y_i \in Y) \quad (1)$$

[2] https://engineering.fb.com/2017/03/29/data-infrastructure/faiss-a-library-for-efficient-similarity-search/

If the metric yields a small value, the distance between the two vectors is short, which indicates that the two documents are similar. Conversely, if the metric yields a large value, the distance between the two vectors is long, which means that the two documents are dissimilar. With the Euclidean Distance , we determined that the two documents were similar if the value was below a specific value(14.5).

3. Approach

We propose the method we presented in Figure 2. The method consists of the following steps: data preprocessing, sentence embedding, Faiss database creation, and the search process.

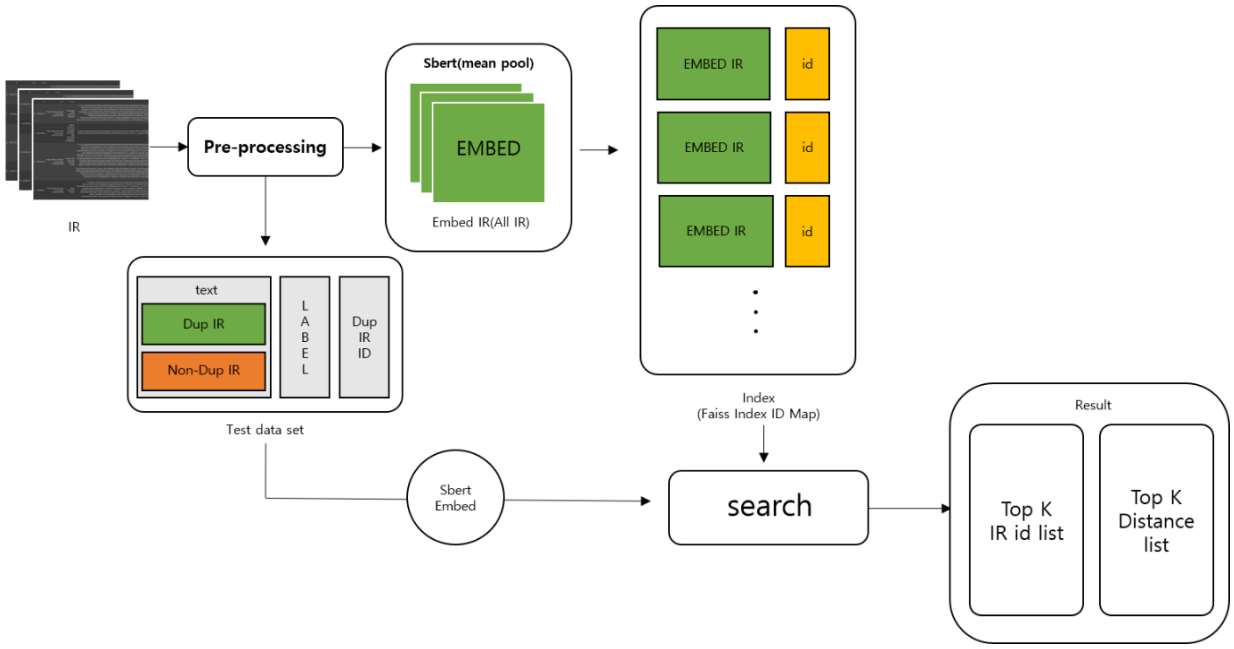


Figure 2 : Duplicate bug reports detection architecture

3.1. Data Preprocessing

In our method, bug reports consist of the title and body attributes. These attributes may contain emojis and special characters. Additionally, the body has no character limit, allowing for lengthy descriptions, sometimes spanning thousands of characters. To mitigate potential hindrances when embedding this content with SBERT and to ensure a more comprehensive representation of contextual meaning, we preprocess the title and body sections. We then combine them into a single text attribute. We apply the following preprocessing steps to the title and body text:

1. Removal of line breaks represented as "/r/n."
2. Elimination of all characters except for letters and numbers.
3. Concatenation of the text from both the title and body.
4. Restriction of the combined text to a maximum of 1024 characters

3.2. Sentence Embedding

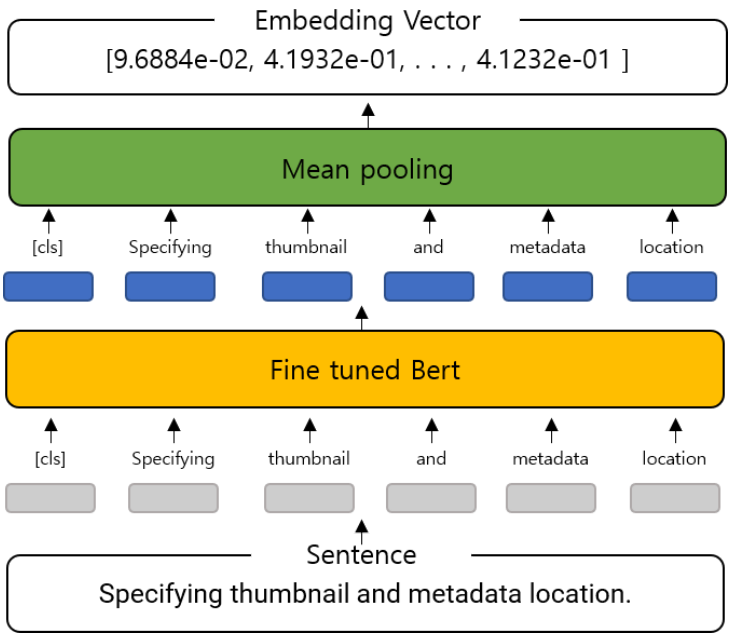


Figure 3 : Sentence BERT Example

The preprocessed bug reports are embedded into vectors so that the similarity of the bug reports can be calculated in a value. When embedding bug reports, it would be good to map semantically similar documents closely. For that, we used SBERT by specifically employing the mean pooling method.

SBERT uses the siamese network and the triplet network to do this with a fine tuned model of Bert, making it possible to map more accurately and to calculate faster according to the meaning of the document.[7] A preprocessed document is provided as input to fine tuned BERT. The output consists of tensors where each word is represented as a vector. The process involves averaging these output vectors to represent the entire sentence as a single tensor.

For example, Figure 3 presents the sentence "Specifying thumbnail and metadata location.". This sentence is tokenized and each word is converted to a vector. Then, the vectors of the words are averaged and the averaged vector becomes the embedding vector for the sentence "Specifying thumbnail and metadata location". If the sentence transformers library is used , the output tensor will typically be a 768- dimensional vector. In this way, all collected bug reports are converted to vectors.This process is applied to all bug reports.

3.3. Faiss IndexIDMap

In large open source projects where many contributors participate, tens of thousands of bug reports are daily generated. Therefore, to detect duplicate bug reports for a single bug report, the tens of thousands of existing bug reports should be compared with the bug report. As the project accumulates more bug reports, the complexity of this process continues to increase. To address this complexity, we used the Faiss library. The Faiss library creates an index map by clustering documents so as to speed up searching similar documents. The Faiss library also reduces memory usage by compressing large amounts of data. . To calculate the similarity of documents by using Faiss, users need to create an IndexIDMap. As shown in Figure 2, this involves creating an IndexIDMap pairing the embedded bug reports with their respective I

3.4. Faiss Search

When a new bug report is created, the bug report is embedded by SBERT. The embedded bug report is then compared to other bug reports in the IndexIDMap.

The K most similar bug reports can be extracted by the Faiss library's Search function. This function returns a list containing the IDs of the bug reports and the distances between the new bug report and each existing bug report. If the distance is less than a certain threshold 14.5, the existing bug reports and the new bug report are determined as duplicates.

4. Experimental Set-up

4.1. Research Questions

In this paper, we ask three research questions.

RQ1. When focusing on duplicate bug report detection for duplicate pairs dataset, what is the accuracy of the proposed method?
RQ2. When considering all bug reports with duplicate Labels, what is the accuracy of the proposed method for duplicate bug report detection tasks?
RQ3. How fast is the search process using the Faiss library?

4.2. Target Data

To conduct our experiments, we initially identified and sorted projects in the GitHub dataset that had a significant number of "duplicate" Labels. We then manually investigated these projects to determine if they were practical open-source projects. As a result of this process, we selected the YouTube-dl project on GitHub for our evaluation.

We then collected bug reports from the YouTube-dl project by using web crawling. From the various attributes of GitHub bug reports, we collected only the title, body, label, and ID attributes. Figure 4 represents the analysis of the bug report data from the YouTube-dl project. The statistics are visualized as a bar graph. In total, there were 19,855 issues in the dataset.

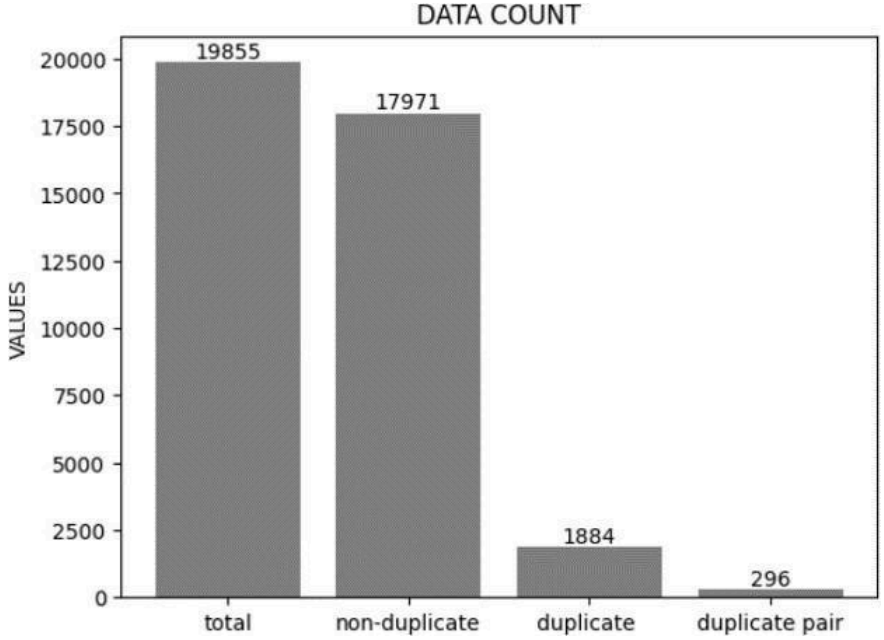


Figure 4 : Youtube-dl project data analysis

The issues were categorized into three main groups based on the following criteria:

1. **Non-duplicate**: The number of bug reports without “duplicate” labels.
2. **Duplicate**: The number of bug reports with “duplicate” labels, assigned by authorized developers.
3. **Duplicate pair**: The number of bug reports within the duplicate data set where the pairs of duplicate reports can be found.

In practice, authorized developers manually assign labels to bug reports. In projects with a high volume of developer participation, thousands of new bug reports can be generated daily. It becomes practically impossible for authorized developers to individually verify and label all of them. As a result, there are often non-duplicate bug reports that are effectively considered as duplicates. Conversely, there are situations where bug reports receive “duplicate” labels but actually they are not duplicate bug reports.

Bug reports with Duplicate labels must have an originating source bug report which have same content. When authorized developers assign a "duplicate” label, they may or may not include the ID or link of the source bug report in the bug report's Comment. Most duplicate issues do not include the ID or link to the source issue. Ignoring the mistakes made by authorized developers in such cases is difficult. Therefore, to minimize the impact of such errors, we collected only duplicate bug reports for which the source bug report could be identified, creating a dataset named ‘Duplicate pair’.

Table 1
Test Data Set Count

Test Data set	Non duplicate	Duplicate	Total
Duplicate	1884	1884	3768
Duplicate pair	296	296	592

To avoid bias in the test data set and ensure that the model does not perform exceptionally well due to such bias, we constructed the test data set with an equal number of duplicate and non-duplicate issues. Table 1 shows a table with the number of issues for two different datasets.

The first test dataset, named ‘Duplicate’, is created by randomly selecting the same number of Duplicate issues as the Non-duplicate data and then combining them. The second test dataset, named ‘Duplicate pair’, is generated by randomly extracting as many issues from the Non-duplicate issues as there are in the Duplicate pair dataset and then combining them with the Duplicate pair dataset.

5. Experimental Results & Analysis

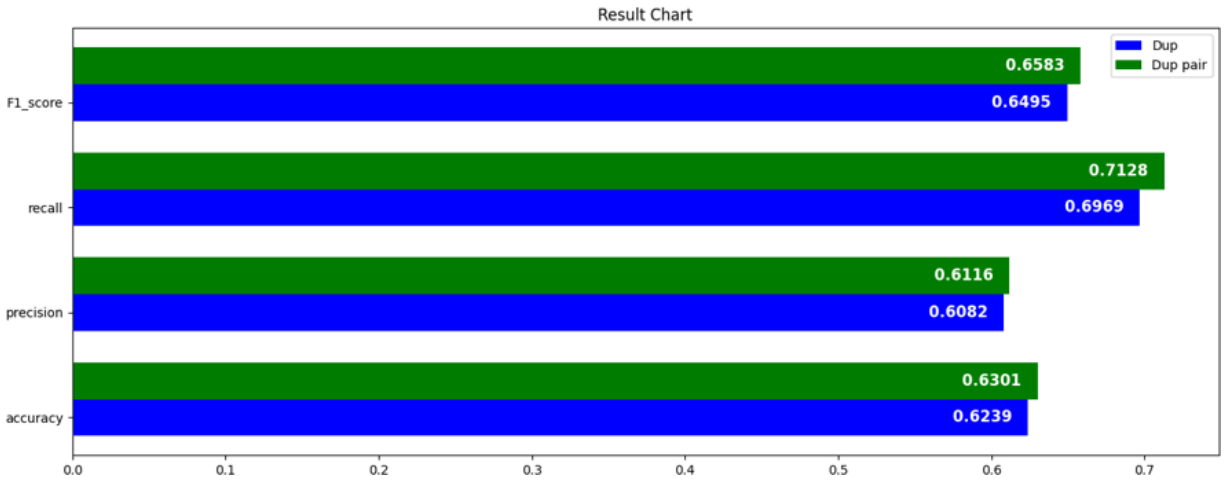


Figure 5: Result Bar Graph

In this section, we have summarized and provided answers to each of the previously mentioned research questions one by one. Figure 5 presents four different result metrics – F1 score, recall, precision, and accuracy – for two datasets. The evaluation results can be the answers for RQ1 and RQ2.

5.1. Experimental Results for RQ1

The answer to RQ1 can be understood by examining Figure 5. In Figure 5, the green bars represent the results for the Duplicate pair dataset. In the case of the Duplicate pair dataset, the F1 score is 0.6583. The recall is 0.7128, while the precision is 0.6116. Finally, the accuracy is 0.6301. If we consider the previous duplicate bug report detection techniques [4], the performance of our proposed method is not high but presents a reasonable performance in that our methods does not require the cost of training a model.

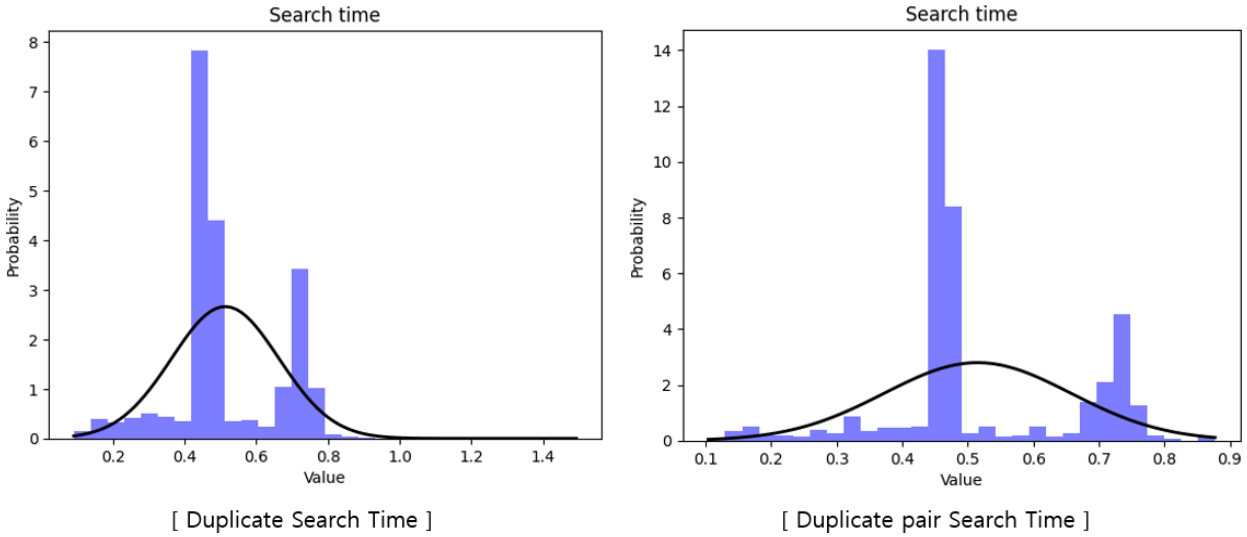
5.2. Experimental Results for RQ2

The answer to RQ2 can also be understood by examining Figure 5. In Figure 5, the blue bars represent the results for the Duplicate dataset. the F1 score is 0.6495. The experimental results can be compared to the previous study [9], which achieved an F1-score of 0.329 for duplicate Labels. This is the most interesting point, because we can see an improvement in the Duplicate dataset with an F1-score of approximately 0.3205. This means that the results of the current study achieved F1 scores that were roughly twice as high as those from the previous research [9].

When comparing the results for the Duplicate pair dataset and the Duplicate dataset, we can see that in each aspect, the Duplicate pair dataset outperforms the Duplicate dataset. Specifically, the F1-score for the Duplicate pair dataset is 0.0088 higher, recall is 0.0159 higher, precision is 0.0034 higher, and accuracy is 0.0062 higher.

These improvements indicate that the Duplicate pair dataset has undergone more stringent data refinement compared to the Duplicate dataset. Notably, the recall value exhibits the largest difference. Since the Duplicate pair dataset eliminates cases where authorized developers wrongly assigned Duplicate Labels to non-duplicate issues, it is expected that the recall value would be significantly higher in the Duplicate pair dataset compared to the Duplicate dataset.

5.3. Experimental Results for RQ3



[Duplicate Search Time]

[Duplicate pair Search Time]

Figure 6: Search Time Graph

Table 2
Search Time

Test Data set	Max(sec)	Min(sec)	Mean(sec)	Variance(sec)
Duplicate	1.4929	0.0908	0.5132	0.0225
Duplicate pair	0.8773	0.1041	0.5150	0.0204

Figure 6 represents the time it takes to compute the similarity between the bug reports in the test dataset and the entire dataset and to find the K most similar bug reports. The graph on the left pertains to the Duplicate dataset, while the one on the right is for the Duplicate pair dataset.

Table 2 presents the maximum, minimum, average, and variance of the search times. In response to RQ3, it can be observed from Table 2 that when using the Faiss library, it takes approximately 0.5 seconds to search the entire dataset of 19,855 bug reports. Even in cases where the search time is longer, it remains within 1.5 seconds. The small variance, approximately 0.02, indicates that the search time for most data points is around 0.5 seconds.

6. Discussion

We have proposed a new approach if a new issue report is a duplicate report of the existing one or not and evaluated the performance of the proposed approach. However, we have not qualitatively analyzed the samples of the evaluation. Therefore, we selected one sampled issue report from our evaluation datasets and investigated what happened to determine if the issue report is duplicated or not. The selected sample was a issue report of the youtube-dl project, #21639 "TED support is broken3". once the issue report is given, our approach gets the title and content of the report as input and returns the id and distance values of the five bug reports that are most similar to the issue. In our experiments, the model returned [#22317, 0.907], [#21947, 1.022], [#22374, 1.063], [#21390, 1.065], and [#23378, 1.082]. The distance values were calculated by the Euclidean distance between the two bug reports. When the distance value of a bug report is 14.5 or less, the approach determines that the new issue is a duplicate bug report. Because the bug report "TED support is broken" has a set of bug reports with a distance not greater than 14.5, the bug report is determined to be a duplicate. The most similar bug report is the bug report #22317 "younow extractor is broken4." With the mechanism, the proposed approach successfully predict if a new bug report is duplicated or not.

The reason that we have proposed a new approach in this paper is that existing approaches that determine if two bug reports are duplicate or not have several limitations. The first group of approaches, relying on Information Retrieval (IR) methods [1, 2, 3, 19, 20], were based on word similarity of bug reports. However, However, the paper "Towards Understanding the Impacts of Textual Dissimilarity on Duplicate Bug Report Detection" [15], published by Sigma Jahan and Mohammad Masudur Rahman in 2023, claims that there are many duplicate bug reports with word dissimilarity. The second group of approaches, relying on Machine Learning (ML) methods [4, 10, 11, 5, 6], were based on the training data for creating machine learning models. However, the paper "Duplicate Bug Report Detection: How Far Are We?"[13] published by Ting zhang in 2023 argued that many methods use biased data (e.g., testing using only old bug reports, collecting data using only certain bug tracking systems, etc. Besides, machine learning based approaches take a significant amount of time for training a model with data.

In this paper, we considered these arguments and used SBERT to encompass the semantic context of issue reports, and used data from real Github projects to reflect realism. In addition, we implemented a fast duplicate bug report detection using the Faiss library. However, we also observe the limitations of this paper. First, we have not provided the performance of baseline approaches to compare with our proposed approach. Although we certainly observed that our approach with the

[3] https://github.com/ytdl-org/youtube-dl/issues/21639

[4] https://github.com/ytdl-org/youtube-dl/issues/22317

Faiss yields a fast search speed based on the clusters and indices of Faiss, we acknowledge that we have provided no performance comparison experimental results on how fast similar search is compared to other ML methods. Second, the paper "Duplicate Bug Report Detection: How Far Are We?"[13] reported that the IR-based REP method presented in the paper "Towards More Accurate Retrieval of Duplicate Bug Reports"[3] by Chennian Sun in 2011 performed the best. However, we have not compared the performance of our approach with the existing approach. Therefore, further experiments are needed. Third, we assume that duplicate bug reports have a negative impact and should be detected and marked in advance. However, in 2008, Nicolas Bettenburg published "Duplicate Bug Reports Considered Harmful . . . Really?"[17] argued that duplicate bug reports have a positive impact because they provide additional information, allowing us to learn more about the bug. According to their point of view, it would be better to preserve the additional information by merging duplicate bug reports rather than disregarding them. Therefore, we need to deeply deliberate how to utilize the information of bug reports for facilitating software evolution.

7. Threats to Validity

7.1. Threats to Internal Validity

There are three main internal factors that threaten the validity of the experiments in this paper. First, we arbitrarily set a threshold of 14.5 as a constant to determine how far two issues should be to not be considered as duplicate issues. In other words, when calculating the similarity between a new issue and existing issues, if the Euclidean distance between the two issues is less than or equal to 14.5, they are considered duplicates. This setting could lead to entirely different results when applied to different datasets.

Second, we imposed a maximum limit of 1024 characters on the text of all data. Without this limit, not only would a significant amount of data be lost in the SBERT process, but also the embedding speed would slow down considerably.

Third, there is the issue of inaccurate data labeling. The labeling is solely based on the judgment and decision of authorized developers who attach Duplicate Labels. As a result, variables where authorized developers either mistakenly attach Duplicate Labels to issues that should not have them or omit them from issues that should be labeled as duplicates are reflected in the dataset.

7.2. Threats to External Validity

In this paper, we conducted a performance analysis on a single open-source project. However, it's important to note that performance metrics can vary depending on the open-source project and the dataset used. Therefore, in future work, we plan to increase the number of projects to measure the metrics for each project and calculate their average values.

Additionally, we did not include a comparison experiment with existing methods in this study. In future research, we intend to include such comparisons to provide a more comprehensive analysis.

8. Conclusion & Future Work

In this paper, we introduced a novel method for duplicate bug report detection using SBERT and Faiss. Addressing the challenge of training and search speed delays for large datasets was a significant concern, and we effectively addressed it by leveraging Faiss, achieving search times of approximately 0.5 seconds. Additionally, our proposed method demonstrated a notable performance improvement, about twice as effective, compared to previous research[9] in detecting “duplicate” labels.

However, there are certain limitations and areas for future work. Firstly, since we employed real-

world project bug report data, we couldn't completely eliminate the impact of data noise. Therefore, further experiments with denoised data are necessary. Secondly, our experiments were conducted using bug report data from a single project, Youtube-dl. Future research should assess the applicability of this methodology to various issue datasets. Particularly, it would be essential to use datasets with a more extensive range of data to effectively evaluate Faiss's performance. Additionally, exploring alternative similarity metrics beyond Euclidean Distance or implementing more rigorous data pre- processing steps are directions for future research and improvement.

References

[1] A. Hindle and C. Onuczko, “Preventing duplicate bug reports by continuously querying bug reports”, Empirical Software Engineering, 2019.

[2] Nicholas Jalbert and Westley Weimer, “Automated Duplicate Detection for Bug Tracking Systems”, 2008 IEEE International Conference on Dependable Systems and Networks With FTCS and DCC(DSN), 2008.

[3] Chengnian Sun, David Lo, Siau-Cheng Khoo and Jing Jiang, “Towards More Accurate Retrieval of Duplicate Bug Reports”, 2011 26th IEEE/ACM International Conference on Automated Software Engineering(ASE 2011), 2011.

[4] Jianjun He, Ling Xu, Meng Yan, Xin Xia and Yan Lei, “Duplicate Bug Report Detection Using Dual-Channel Convolutional Neural Networks”, ICPC '20: Proceedings of the 28th International Conference on Program Comprehension, 2020.

[5] Montassar Ben Messaoud, Asma Miladi, Ilyes Jenhani and Mohamed Wiem Mkaouer, “Duplicate bug report detection Using an Attention-Based Neural Language Model”, IEEE Transactions on Reliability (Volume: 72, Issue: 2, June 2023), 2023.

[6] Lahari Poddar, Leonardo Neves, William Brendel, Luis Marujo, Sergey Tulyakov and Pradeep Karuturi, “Train One Get One Free: Partially Supervised Neural Network for Bug Report Duplicate Detection and Clustering”, arXiv preprint arXiv:1903.12431, 2019.

[7] Nils Reimers and Iryna Gurevych, “Sentence-BERT: Sentence Embeddings using Siamese BERT-Networks”, arXiv preprint arXiv:1908.10084, 2019.

[8] Jacob Devlin, Ming-Wei Chang, Kenton Lee and Kristina Toutanova, “BERT: Pre-training of Deep Bidirectional Transformers for Language Understanding”, arXiv preprint arXiv …, 2018.

[9] Jueun Heo and Seonah Lee, “An Empirical Study on the Performance of Individual Issue Label Prediction”, 2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR), 2023.

[10] Irving Muller Rodrigues, Daniel Aloise, Eraldo Rezende Fernandes and Michel Dagenais, “A Soft Alignment Model for Bug Deduplication”, 2023 IEEE/ACM 20th International Conference on Mining Software Repositories (MSR), 2023.

[11] Jayati Deshmukh, Annervaz K M, Sanjay Podder, Shubhashis Sengupta and Neville Dubash, “Towards Accurate Duplicate Bug Retrieval using Deep Learning Techniques”, 2017 IEEE International Conference on Software Maintenance and Evolution (ICSME), 2017

[12] Guanping Xiao, Xiaoting Du, Yulei Sui and Tao Yue, “HINDBR: Heterogeneous Information Network Based Duplicate Bug Report Prediction”, 2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)1, 2020.

[13] Ting zhang, Donggyun han, Venkatesh vinayakarao, Ivana clairine irsan, bowen xu, ferdian thung, david lo and lingxiao jiang“Duplicate Bug Report Detection: How Far Are We?”, ACM Transactions on Software Engineering and Methodology, 2023.

[14] Avinash Patil, Kihwan Han and Sabyasachi Mukhopadhyay, “A Comparative Study of Text Embedding Models for Semantic Text Similarity in Bug Reports”, arXiv preprint arXiv:2308.09193, 2023.

[15] Sigma Jahan and Mohammad Masudur Rahman, “Towards Understanding the Impacts of Textual Dissimilarity on Duplicate Bug Report Detection”, 2023 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER), 2023.

[16] Xiaoxue Wu, Wenjing Shan, Wei Zheng, Zhiguo Chen, Tao Ren and Xiaobing Sun, “An Intelligent

Duplicate Bug Report Detection Method Based on Technical Term Extraction", 2023 IEEE/ACM International Conference on Automation of Software Test (AST), 2023.
[17] Nicolas Bettenburg, Rahul Premraj, Thomas Zimmermann and Sunghun Kim"Duplicate Bug Reports Considered Harmful . . . Really?", 2008 IEEE International Conference on Software Maintenance, 2008.
[18] Per Runeson, Magnus Alexandersson and Oskar Nyholm, "Detection of Duplicate Defect Reports Using Natural Language Processing ", 29th International Conference on Software Engineering (ICSE'07), 2007.
[19] Cheng-Zen Yang, Hung-Hsueh Du, Sin-Sian Wu and Ing-Xiang Chen "Duplication Detection for Software Bug Reports based on BM25 Term Weighting", 2012 Conference on Technologies and Applications of Artificial Intelligence, 2012.
[20] Karan Aggarwal, Tanner Rutgers, Finbarr Timbers, Abram Hindle, Russ Greiner, and Eleni Stroulia, "Detecting Duplicate Bug Reports with SoftwareEngineering Domain Knowledge", 2015 IEEE 22nd International Conference on Software Analysis, Evolution, and Reengineering (SANER), 2015.