# Combining Retrieval and Classification: Balancing Efficiency and Accuracy in Duplicate Bug Report Detection

1st Qianru Meng
*LIACS*
*Leiden University*
Leiden, Netherlands
mengqr@vuw.leidenuniv.nl

2nd Xiao Zhang
*CLCG*
*University of Groningen*
Groningen, Netherlands
xiao.zhang@rug.nl

3rd Guus Ramackers
*LIACS*
*Leiden University*
Leiden, Netherlands
g.j.ramackers@liacs.leidenuniv.nl

4th Visser Joost
*LIACS*
*Leiden University*
Leiden, Netherlands
j.m.w.visser@liacs.leidenuniv.nl

***Abstract*—In the realm of Duplicate Bug Report Detection (DBRD), conventional methods primarily focus on statically analyzing bug databases, often disregarding the running time of the model. In this context, complex models, despite their high accuracy potential, can be time-consuming, while more efficient models may compromise on accuracy. To address this issue, we propose a transformer-based system designed to strike a balance between time efficiency and accuracy performance. The existing methods primarily address it as either a retrieval or classification task. However, our hybrid approach leverages the strengths of both models. By utilizing the retrieval model, we can perform initial sorting to reduce the candidate set, while the classification model allows for more precise and accurate classification. In our assessment of commonly used models for retrieval and classification tasks, sentence BERT and RoBERTa outperform other baseline models in retrieval and classification, respectively. To provide a comprehensive evaluation of performance and efficiency, we conduct rigorous experimentation on five public datasets. The results reveal that our system maintains accuracy comparable to a classification model, significantly outperforming it in time efficiency and only slightly behind a retrieval model in time, thereby achieving an effective trade-off between accuracy and efficiency.**

***Keywords-Duplicate Bug Detection; Deep Learning; Natural Language Processing; Transformer; Running Time; Accuracy.***

## I. INTRODUCTION

Bug reports are crucial in the software development and maintenance phase, providing valuable information to software developers [1][2]. It commonly comprises structured text (e.g., timestamp, version, component, and bug status) and unstructured text, such as title and description [3]. Typically, bugs are recorded in the Bug Database (also known as Bug Tracking System) by developers, testers and users [3][4][5]. Unfortunately, the inconsistent understanding of bug descriptions by different writers leads to the continuous generation of numerous duplicate bug reports [6], which increases maintenance costs. Consequently, significant research efforts have been devoted to detecting duplicate bugs, aiming to reduce redundant work involving the testing of bugs that have already been resolved [5][7][8], thereby enhancing the efficiency of the bug fixing process [9].

DBRD task can be defined as: the automatic process of identifying and comparing the semantic content in bug reports to discover new reports that are duplicate or highly similar to existing reports. As shown in Table I, there are two instances of duplicate bug reports where similar features have been highlighted. These features are not limited to exact word matching, but also extend to semantic similarity and context. Therefore, this places high demands on the capacity of automatic text processing techniques.

TABLE I: COMPARISON OF DUPLICATE BUG REPORTS FROM ECLIPSE

| | |
|---|---|
| Bug_id | 178 |
| Title | Maintain sync view expansion state when switching modes |
| Description | It would be nice if when things got filtered out, their expansion would be remembered, so that when the item is revealed again it has the correct expansion. For example, if you have one outgoing change; switch to the catchup pane and then come back, the tree is completely collapsed. |
| Bug_id | 226 |
| Title | Switching between sync UI modes should preserve expansion state |
| Description | When you switch between Catch Up and Release modes, it loses the expansion state of the tree. It should remember this and probably the selection and top item (scroll bar position) as well. |

Traditionally, the automatic approach to DBRD has been divided into two distinct tasks: Information Retrieval (IR) and classification[1]. Early methods for IR primarily relied on word-based approaches (e.g., Vector Space Model), as well as topic-based models like Latent Dirichlet Allocation (LDA) and Latent Semantic Analysis (LSA), which transformed bug reports into feature vectors. More recently, embedding models, such as Word2Vec [10][11], GloVe [12], and sentence BERT [13] have gained traction. These models generate embeddings that are then utilized to calculate similarities between bugs, typically using distance measurements, such as Cosine similarity. These retrieval methods have demonstrated promising performance, particularly in terms of recall rate [1]. Simultaneously, classification models, particularly deep-learning-based approaches, have emerged as prominent research focus in DBRD [1][5][9]. Initially, the classifier employed the Convolutional Neural Network (CNN) [6][11][14][15], followed by the Recurrent Neural Network (RNN) [15] and eventually transitioning to the Long Short-Term Memory (LSTM) model [15]. However, due to the challenges associated with processing lengthy text, the performance of these three models has been surpassed in recent years by transformer-based classi-

 ISBN: 978-1-68558-098-8

fiers, most notably BERT [3][5], sentence BERT [3][13] and RoBERTa [3]. These large language models are pre-trained on large corpus and fine-tuned on domain-specific data, which enables them to capture contextual semantic information and generate word and sentence representations efficiently. Not surprisingly, transformer-based models became the state-of-the-art for this task [3].

However, in previous studies, we found that commonly used dataset splitting methods have data leakage issues, which may lead to biased results. Specifically, it is possible for a single data within a pair in the train set to be combined with another data and consequently appear in the development or test set. This unintentional leakage has not been explicitly addressed by most existing methods, with only one work taking this matter into account without explicitly acknowledging it [3]. Therefore, one of the main contributions of our work is the design of a Cluster-based dataset partition mechanism to address this problem.

Most importantly, while there has been a considerable emphasis on performance metrics, such as recall and precision in existing studies, the evaluation of these approaches' efficiency in terms of speed has often been overlooked. As highlighted by Haruna et al. [13] in their research, with the advent of large language models, such as BERT, the performance of retrieval and classification tasks has shown remarkable advancements. Nevertheless, the deployment and execution of these models can present difficulties due to their relatively slower inference times. Especially when it comes to practical applications, the speed plays a critical role. As a result, it's essential to evaluate a model not just based on its accuracy, but also on its efficiency.

In our research, we propose a novel system based on the transformer architecture that combines the advantages of retrieval model and classification model. Our approach integrates retrieval techniques to retrieve an initial set of potential duplicate instances, which is then fed into a classification model for further triage. This innovative methodology enables us to achieve faster performance without compromising accuracy. By effectively merging these two components, we attain a balance between efficiency and accuracy in DBRD task.

The contributions of our work are as follows:

- Cluster-based dataset partition mechanism: To address the problem of train set leakage, we introduce a cluster-based dataset partition mechanism. This mechanism ensures that duplicate instances are evenly distributed across the train and test sets, effectively mitigating any potential data leakage issues.
- Comparison with previous models: We conduct a comprehensive comparison between the performance of the transformer-based models and that of previous methods in retrieval and classification. Through rigorous experiments and evaluations, we demonstrate that our transformer-based models outperform on both tasks, surpassing the performance of previous models.
- Integration of retrieval model and classification: Our proposed system leverages the strengths of both retrieval models and classification models. As demonstrated in the experiments, our system can achieve a balanced between speed and accuracy in two real-world scenarios.

We introduce the related work in Section II, detail our approach in Section III and validate the experiments in large open source projects to demonstrate the effectiveness in Section IV and Section V.

## II. Related work

As previously mentioned, solutions to DBRD can be viewed as IR task and classification task. Approaches to IR tasks focus on identifying duplicates by computing similarities between textual representations, while classification tasks typically utilize deep learning techniques to train models in distinguishing between "duplicated" and "non-duplicated" instances based on learned patterns. In the following subsections, we present related work on these methods.

*1) Information Retrieval Methods:* Hiew [16] introduces a retrieval method for unstructured text including titles and descriptions. Textual fields are converted to TF-IDF vectors, which are then organized into clusters based on their similar characteristics to identify duplicates. Runeson et al. [7] utilized a Vector Space Model to present text-based information and determined the text similarity by using three similarity calculation methods. Wang et al. [17] integrated execution data into their strategy to detect similar bug reports. Sun et al. [18] proposed a REP model that incorporates similarity of lexical features and categorical features from bug reports. Nguyen et al. [9] introduced the DBTM model that processes topic features extracted by LDA model and unstructured textual features. It combines topic model and retrieval model to show both similarity and dissimilarity between bug reports. Some follow-up studies [19][20][21] adopt a similar approach to previous studies, also implementing topic models for retrieval, but differentiate their studies by analyzing distinct corpora and utilizing varied feature inputs in bug reports.

Therefore, traditional IR methods primarily focus on the calculation of word frequency feature to detect duplicates, which show advantages in processing structured text and keyword-based queries. However, IR methods exhibit limitations in processing contextual information and complex semantic features, areas where deep learning (DL) methods demonstrate proficiency.

*2) Deep Learning Methods:* Deshmukh et al. [14] were the first to introduce deep learning into duplicate bug report detection, proposing a model that uses Siamese Convolutional Neural Networks and Long Short Term Memory to process hybrid input from bug reports for retrieval and classification. Budhiraja et al. [22][23] proposed Deep Word Embedding Networks (DWEN), a framework designed to retrieve similar reports by processing unstructured input, including bug report titles and descriptions. Xie et al. [10] introduced a deep learning framework named DBR-CNN, which enhances

 ISBN: 978-1-68558-098-8 

traditional CNN by integrating domain-specific features extracted from bug reports. The hybrid features are fed into the CNN model to obtain concatenated vectors, which are utilized for classification task. Poddar et al. [24] proposed a neural architecture for multi-task learning, with joint tasks of classifying duplicates and clustering latent topics, operating on unstructured descriptions as input. Building upon the CNN framework, He et al. [11] subsequently developed a Dual-Channel CNN (DC-CNN) method to classify duplicate bug reports using hybid-structured text as input. Kukkar et al. [6] presented a deep learning based classification model applied on hybrid features, also leveraging CNN to extract relevant features that are subsequently used to compute similarities for classification purposes.

Following these advancements, transformer-based language models have gained considerable attention and popularity within the present landscape of duplicate bug report detection, due to their rich context-based learning capabilities. Isotani et al. [13] introduced transformer-based deep learning embedding model of SBERT to vectorize the unstructured textual features (title and description) and then computes the similarity of the embedding representations, enabling retrieval of similarly ranked bug reports. Rocha et al. [5] proposed a SiameseQAT approach, using BERT and MLP to concatenate structured and unstructured features and features extracted based on corpus topics for retrieval and classification tasks respectively. Messaoud et al. [3] proposed a BERT-MLP model for classifying duplicate bug reports, which considers only unstructured data. The model utilizes BERT to generate contextualized word representations and applies an MLP for classification. Jiang Y et al. [25] suggested a CombineIRDL method, which utilizes different deep learning models to extract lexical, categorical, and semantic features from hybrid input and then employs a retrieval model to obtain ranked duplicates.

Building on these deep learning methods discussed in the literature, we find that most of them accomplish duplicate detection by implementing retrieval and classification tasks separately [5][6][14] or focus on a single task [3][10][11][13][22][23][24][25]. Furthermore, deep learning methods have demonstrated significant effectiveness in both tasks. In IR tasks, deep learning enhances similarity assessments by employing advanced word embedding models, such as transformer models. In classification tasks, it trains models (such as CNN, LSTM or transformer models) to predict whether two bug reports are duplicates by leveraging their learning capabilities to discern complex textual patterns. By employing these advanced deep learning models, classification tasks can achieve higher accuracy than IR tasks through the extraction of comprehensive textual features, but at the cost of thousands of computations to achieve such precision. Conversely, IR tasks can achieve more significant efficiency in reducing the search space than classification tasks. However, previous work has not considered the trade-off between accuracy and efficiency. Therefore, our approach combines those two tasks in order to fully exploit their strengths in terms of efficiency and accuracy, thereby achieving a balance. In doing so, we apply transformer-based models in our approach, which are widely recognized as the state-of-the-art for Natural Language Processing (NLP) tasks by exploiting their ability to learn semantic and contextual information. These models are utilized to generate embedding representations in the retrieval task and to identify duplicate pairs in the classification task. The following section contains more details of our methodology.

## III. Methodology

In this section, we outline our methodology for the DBRD task, including the overall architecture, pre-processing, data split, and model fine-tuning.

### A. Overall Architecture

The overall architecture of our proposed approach is shown in Figure 1, which consists of three important phases: data split, retrieval and classification.

- **Data split** is to split the test and train set for preparing for the training of retrieval and classification models and we introduce it detailed in Section III-C.
- In the **Retrieval** phase, the retrieval model is responsible for generating the embedding representation of bug report input. Using cosine similarity [26], it calculates the embedding similarity and selects the Top K similar bug reports. This ”ranking” is primarily used to identify the top-K candidates, which serve as the target input for the subsequent classification model.
- In the **Classification** phase, the model takes the top-K candidates from the retrieval phase and aims to output the final results by labeling them as duplicates or non-duplicates.

### B. Pre-processing

In DBRD task, the input of bug reports can typically be unstructured input containing only unstructured textual fields, or hybrid input including both unstructured textual features and structured categorical features. Our emphasis lies on unstructured input, specifically title and description. These text constitute the most critical component of bug reports and they are also noisy and complex, covering a large number of domain-specific technical fields. To eliminate the redundant and invalid data in the content of bug report datasets, the following operations are used to clean the datasets:

- Remove all non-English words from the text (note: adjust this based on the primary language of the dataset)
- Remove some special characters but keep periods and commas
- Remove stop words
- Unify all letters to lowercase

### C. Data Split

As outlined in Section I, most prior studies have neglected the leakage from the train set to the development and test set. To address this issue, we introduce a cluster-based dataset generation method, as illustrated in the Data Split Phase of

 ISBN: 978-1-68558-098-8

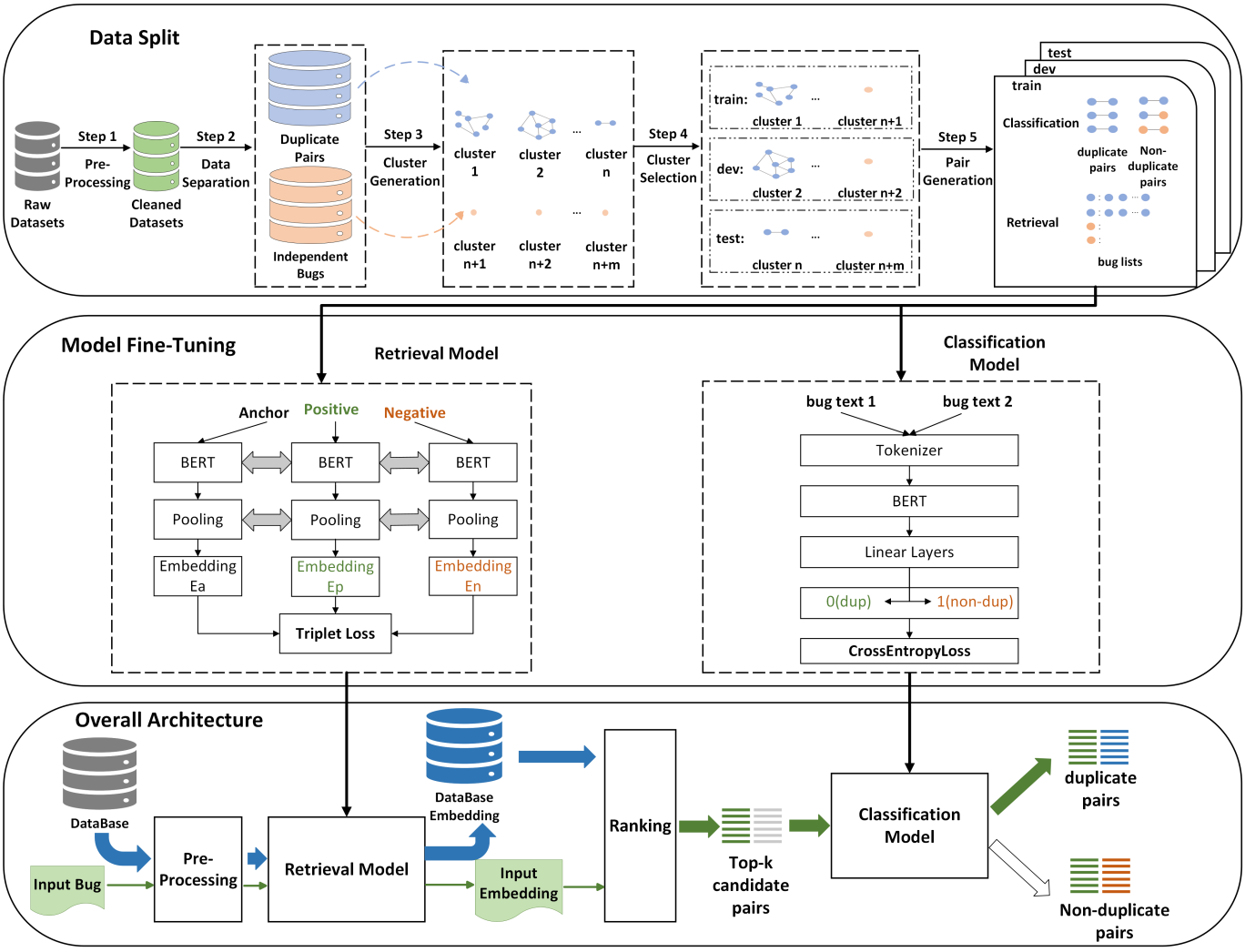



Fig. 1: Transformer-Based Framework split into Two Phases: 1). Data Generation Phase and 2). Model Fine-Tuning Phase

Figure 1. This approach ensures a stringent separation between the train set, development set, and test set.

**Data Separation.** After pre-processing, the bug reports are categorized into two groups: independent bug reports and duplicate bug reports. Independent bug reports refer to those that do not have any duplicates among the dataset.

**Cluster Generation.** In this particular step, we adopt the assumption of transitivity in the relationship between duplicate bug reports. This means that if Bug A is a duplicate of Bug B, and Bug B is a duplicate of Bug C, then Bug A and Bug C are also considered duplicates. Consequently, based on our example, Bugs A, B, and C would be grouped together in a single distinct cluster. Leveraging this assumption, we employ a cluster-based method to group pairs of duplicated bug reports into distinct clusters. Within each cluster, every two bugs are marked as duplicates.

**Cluster Selection.** Unlike prior studies that randomly select duplicate pairs for train/test sets, we choose clusters to form train, development, and test sets. This ensures each bug appears only once in any set, and the clusters across these sets are distinct. This method reduces potential biases and data leakage from directly selecting duplicate pairs.

**Pair Generation.** In this step, we generate two train/test datasets with different data structures for retrieval and classification models respectively. The retrieval dataset follows the format of [Bug ID: Bug IDs], where bugs sharing the same cluster are considered duplicates of each other. On the other hand, the classification dataset comprises both duplicated pairs and non-duplicated pairs in a one-to-one format of [Bug ID: Bug ID], which are generated based on the clusters. For a cluster with $n$ bugs, $\frac{n*(n-1)}{2}$ duplicated pairs can be derived. Furthermore, for two clusters (with sizes n and m, respectively), we can generate $n * m$ non-duplicated pairs.

Notably, the train set is carefully balanced in terms of positive (duplicate) and negative (non-duplicate) data, while the development and test sets are intentionally left unbalanced. This setting aims to reflect the real-world scenario, where the number of non-duplicate bug pairs far exceeds the number of

 ISBN: 978-1-68558-098-8 

duplicate bugs.

### D. Model Fine-tuning

As shown in Figure 1, Model Fine-tuning phase encompasses the process of training two models: the retrieval model and the classification model.

To adapt SBERT for the retrieval task, we modify the dataset structure into triplets [Anchor, Positive, Negative], where we aim to fine-tune the model's ability to distinguish between relevant (positive) and irrelevant (negative) instances. The loss function employed is the Triplet Loss [27], represented by 1. In this equation, $||\cdot||$ denotes a distance metric used to assess the similarity between embeddings. It is important to note that this loss function imposes a condition that the distance between the anchor text and the positive text should be at least $\theta$ greater than the distance between the anchor text and the negative text.

$$Triplet\ Loss = Max(||E_a - E_p|| - ||E_a - E_n|| + \epsilon, 0) \quad (1)$$

In the context of classification, BERT operates by taking in two texts simultaneously, and using [SEP] token to differentiate them. The embedding of the [CLS] token, obtained from the final layer of BERT, is processed through a linear layer. The softmax function is then applied to generate the final prediction. The loss function employed in this case is the CrossEntropy (CE) Loss, as represented by 2.

$$CE(y, p) = -\sum_{i=1}^{N} y_i \log p_i \quad (2)$$

where the $y_i$ is the true label and $p_i$ is the predicted label.

## IV. Experimental Settings

In this section, we detail the experiments, discussing setup, datasets, hyperparameters, evaluation, baselines, and model selection.

### A. Setup

To ensure a fair and consistent comparison between the models, we maintain uniformity by implementing and building the models using Python within the PyTorch framework. We will provide the structured code in subsequent documentation. All experimental procedures were conducted on a Linux server featuring an AMD EPYC-Rome processor and an NVIDIA A40 GPU card. This setting allows for efficient execution and reliable performance evaluation of the models.

### B. Datasets

We use five open source bug report repositories [1] to verify the effectiveness of our system, namely Eclipse, Firefox, Mozilla, JDT, and ThunderBird (TBird), as the experimental datasets in our study. These repositories have been extensively utilized in previous research [1]. We focus on the following statistical attributes to characterize the datasets as Table II shown.

We have detailed the process of Data Split in Section III-C, where we employ an 8:1:1 ratio to split train, development, and test sets respectively. As previously discussed, we introduce skew to the development and test data, while maintaining balance in the train set. Consequently, we adhere to the 'Dup Bug Ratio' as indicated in Table II, to establish the ratio of duplicate pairs in both the development and test sets. Since a substantial number of duplicate and non-duplicate pairs can be generated, we limit the size of the train/test/dev set as shown in Table III.

### C. Hyperparameters

We leverage pre-trained transformer-based models along with their respective tokenizers. Fine-tuning of these models is performed using the AdamW optimizer [28] with a learning rate of $10^{-5}$.

In the classification scenario with SBERT, we introduce a linear layer comprising two hidden layers of 768 hidden size each. For the Bi-LSTM model, we utilize the SGD optimizer, implementing a learning rate of 0.5 and a decay rate of 0.25. For the CNN model, we adhere to the configurations outlined in DC-CNN [11].

To mitigate overfitting, we apply a 0.5 dropout across all models. We process training data in 32-size batches. To bolster the robustness and reliability of the results, each experiment is conducted five times.

### D. Evaluation

*1) Individual Evaluation:* The performance of retrieval and classification models is individually assessed in our study. For the retrieval model, we evaluate the performance by measuring the recall and precision under different Top-k settings. For the classification model, we employ precision, recall, and the corresponding F1 score to indicate the performance.

**Metrics**: Utilizing a confusion matrix, which tabulates the counts of True Positives (TP), False Positives (FP), False Negatives (FN), and True Negatives (TN), we characterize the recall, precision, and F1 score as delineated in 3, 4, and 5 respectively. Above fomulas are the performance indicators for classification. However, in the context of retrieval, the computation of recall@k and precision@k deviates slightly, as demonstrated in 6 and 7.

$$Recall = TP/(TP + FN) \quad (3)$$

$$Precision = TP/(TP + FP) \quad (4)$$

$$F1 = \frac{2 * (Precision * Recall)}{(Precision + Recall)} \quad (5)$$

$$Recall@k = \frac{(relevant\ items\ in\ top - k)}{(relevant\ items)} \quad (6)$$

$$Precision@k = \frac{(relevant\ items\ in\ topk)}{k} \quad (7)$$

[1]Datasets available at: [Online]. Available: https://github.com/logpai/bugrepo

TABLE II: STATISTICS OF FIVE OPEN SOURCE DATASETS

| Dataset | Bugs | Dup Pairs | Separate Bugs | Dup Bug Ratio | Cluster Numbers | Cluster size |
|---|---|---|---|---|---|---|
| Eclipse | 84020(85156) | 13231 | 70752 | 0.1564 | 7519 | 2.760 |
| Firefox | 96258(115814) | 15742 | 80000 | 0.1689 | 6654 | 3.366 |
| Mozilla | 195248(205069) | 34507 | 160378 | 0.1786 | 17263 | 2.998 |
| JDT | 44154(45296) | 6513 | 37608 | 0.1483 | 3828 | 2.701 |
| TBird | 24767(32551) | 4404 | 20050 | 0.1905 | 2133 | 3.065 |

TABLE III: DISTRIBUTION OF BUG PAIRS IN TRAIN/DEV/TEST SET

| Dataset | Train | | Dev | | Test | | Total |
|---|---|---|---|---|---|---|---|
| | Dup | Nondup | Dup | Nondup | Dup | Nondup | |
| Eclipse | 6615 | 6615 | 258 | 1394 | 258 | 1394 | 16534 |
| Firefox | 7871 | 7871 | 332 | 1635 | 332 | 1635 | 19676 |
| Mozilla | 17253 | 17253 | 770 | 3542 | 770 | 3542 | 43130 |
| JDT | 3256 | 3256 | 120 | 693 | 120 | 693 | 8138 |
| TBird | 2202 | 2202 | 104 | 445 | 104 | 445 | 5502 |

*2) Architecture Evaluation:* We conduct a comprehensive evaluation of our proposed system, comparing its performance and efficiency against individual retrieval and classification methods in two common real-world scenarios.

**One VS All**: In this scenario, when a user enters a bug, the system compares the user's input bug with existing bug reports in the entire database. To evaluate this scenario, we divided the test set into two parts: 20% for user input and 80% for the database.

**All VS All**: This scenario often arises on the database side, where developers need to locate and eliminate all duplicate errors in the database. It resembles the One VS All scenario, except that we utilize the entire test set as the database.

The applications of our proposed system as well as retrieval and classification methods in the above scenarios are as follows:

One VS All scenario: when a user submits a bug report,

- **retrieval** scans the entire database to find the report most similar to the submitted report;
- **classification** predicts which reports in the database are relevant to the submitted report based on certain characteristics;
- **proposed system** firstly retrieves the top K most similar reports from the database based on user submissions, and then the classifier further predicts these K reports to ultimately determine the duplicate results.

All VS All scenario: each method repeats the One VS All process, aiming to identify and eliminate all duplicates in the database.

In our approach, the classification process primarily serves to enhance the quality of retrieval results, so using the retrieved metrics allows for a more comprehensive comparison of overall performance, while also displaying the improvements attained by the classification part. Therefore, this evaluation consists of the following metrics: recall@k, precision, accuracy (as depicted by 8) and inference time.

$$Accuracy = TP + TN/TP + TN + FP + FN \quad (8)$$

It should be noted that the maximum k set in our experiments is 100 which builds also in the real word scenario. In practical information retrieval settings, users rarely browse beyond the top 100 results due to the huge amount of data and the limitation of their own attention. Therefore, capping k at 100 strikes a balance between presenting enough relevant results and preventing users from being overwhelmed by too many results.

### *E. Baselines*

Based on individual evaluation in retrieval and classification respectively, we employ GloVe [12] and FastText [29] as retrieval baseline methods. Since both methods generate word-level embeddings, we compute sentence embeddings by averaging the word embeddings. In the evaluation of classification models, we incorporate the Bi-LSTM and DC-CNN as the baselines. Both the Bi-LSTM model and the CNN model have been previously applied as classification models in DBRD research [11][14].

In the overall evaluation, we select outperforming retrieval and classification models as baselines and compare them with our combined approach.

### *F. Model Selection*

In our proposed approach, we select transformer-based models for retrieval and classification. For the retrieval model, we leverage sentence BERT (SBERT) to generate text embeddings and evaluate its efficacy on the retrieval task. In the classification task, we choose three transformer-based models as classification models, BERT, ALBERT and RoBERTa, and compare their performance. We selected these transformer-based models because they have demonstrated effectiveness in previous state-of-the-art studies [3][5][13] for DBRD.

## V. EXPERIMENT RESULTS

We analyze our experimental results by answering following two research questions.

**RQ1: Compared to baseline models, how do the transformer-based models perform on retrieval and classification?**

In our first evaluation, we conduct experiments on retrieval models and classification models, presenting the results in Table IV and Table V, respectively.

Consistent with previous research, our evaluation of the retrieval model primarily focuses on the recall value. Notably, as the k value increased, we observed a substantial improvement in the recall of the model. This result is expected as increasing the value of k allows for more candidates to be considered,

 ISBN: 978-1-68558-098-8 

TABLE IV: RECALL@K OF MODELS IN DUPLICATE BUG RETRIEVAL FOR ALL DATASETS

| | Eclipse | | | Firefox | | | Mozilla | | | JDT | | | TBird | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | r@20 | r@60 | r@100 | r@20 | r@60 | r@100 | r@20 | r@60 | r@100 | r@20 | r@60 | r@100 | r@20 | r@60 | r@100 | Avg r@100 |
| Fasttext | 0.489 | 0.678 | 0.783 | 0.596 | 0.716 | 0.809 | 0.414 | 0.526 | 0.588 | 0.608 | 0.785 | 0.972 | 0.627 | 0.874 | **1.000** | 0.8304 |
| Glove | 0.602 | 0.727 | 0.824 | 0.705 | 0.789 | 0.843 | 0.478 | 0.608 | 0.662 | 0.579 | 0.798 | 0.975 | 0.689 | 0.888 | **1.000** | 0.8608 |
| SBERT | **0.848** | **0.935** | **0.960** | **0.892** | **0.956** | **0.973** | **0.771** | **0.892** | **0.919** | **0.872** | **0.990** | **0.997** | **0.880** | **0.983** | 1.000 | **0.9698** |

TABLE V: PRECISION, RECALL & F1 SCORES OF MODELS IN DUPLICATE BUG CLASSIFICATION TASK FOR ALL DATASETS

| | Eclipse | | | Firefox | | | Mozilla | | | JDT | | | TBird | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | Precision | Recall | F1 | Precision | Recall | F1 | Precision | Recall | F1 | Precision | Recall | F1 | Precision | Recall | F1 | Avg F1 |
| Bi-LSTM | 0.511 | 0.506 | 0.473 | 0.510 | 0.515 | 0.469 | 0.507 | 0.506 | 0.506 | 0.490 | 0.490 | 0.490 | 0.621 | 0.510 | 0.474 | 0.4824 |
| DC-CNN | 0.752 | 0.813 | 0.785 | 0.744 | 0.765 | 0.753 | 0.792 | 0.765 | 0.736 | 0.763 | 0.781 | 0.773 | 0.833 | 0.752 | 0.781 | 0.7660 |
| BERT | **0.825** | 0.888 | 0.848 | 0.881 | 0.921 | 0.899 | 0.824 | **0.892** | 0.849 | 0.772 | 0.857 | 0.797 | 0.870 | 0.898 | 0.883 | 0.8552 |
| ALBERT | 0.806 | **0.896** | 0.834 | 0.874 | 0.920 | 0.893 | 0.819 | 0.889 | 0.845 | **0.825** | **0.872** | **0.843** | **0.885** | **0.902** | **0.893** | 0.8616 |
| RoBERTa | 0.846 | 0.892 | 0.866 | **0.886** | **0.925** | **0.903** | **0.835** | 0.891 | **0.857** | 0.824 | 0.868 | 0.841 | 0.846 | 0.898 | 0.866 | **0.8666** |

thereby raising the probability of identifying duplicate bugs. Upon setting k to 100, we discovered that nearly all duplicates are successfully detected, resulting in an approximate recall value of 1.

Furthermore, by comparing the retrieval performance of the three models, as shown in Table IV, we find that SBERT achieves the highest recall value, under different k values. It outperforms Glove and FastText in all five datasets by an average lead of 12.42%. This significantly demonstrates the superiority of the transformer model in information retrieval capabilities.

In the classification task, we compared the performance of traditional models, such as Bi-LSTM, DC-CNN, with transformer-based models on the three indicators of precision, recall and f1. Our results in Table V show that f1 is significantly improved by 20% to 38% when using transformer-based models compared to traditional methods. When comparing transformer models, their performance does not exhibit significant variations. However, RoBERTa has emerged as the frontrunner, surpassing the others with a slightly higher F1 score of 0.8666.

Therefore, the above experimental results indicate that transformer-based models outperform traditional models in both classification and retrieval performance.

**RQ2: Compared to single retrieval and classification model, how does the proposed system perform in case of recall precision, accuracy and time?**

Figure 2 presents a comparative performance overview showing the difference in recall, precision, accuracy and running time for single retrieval model, classification model and proposal system in the One VS All scenario, and Figure 3 presents the overall performance in the All VS All scenario. The results obtained in the One vs All and All VS All scenarios of the five datasets are relatively similar, so we choose one of datasets, firefox, to show the results.

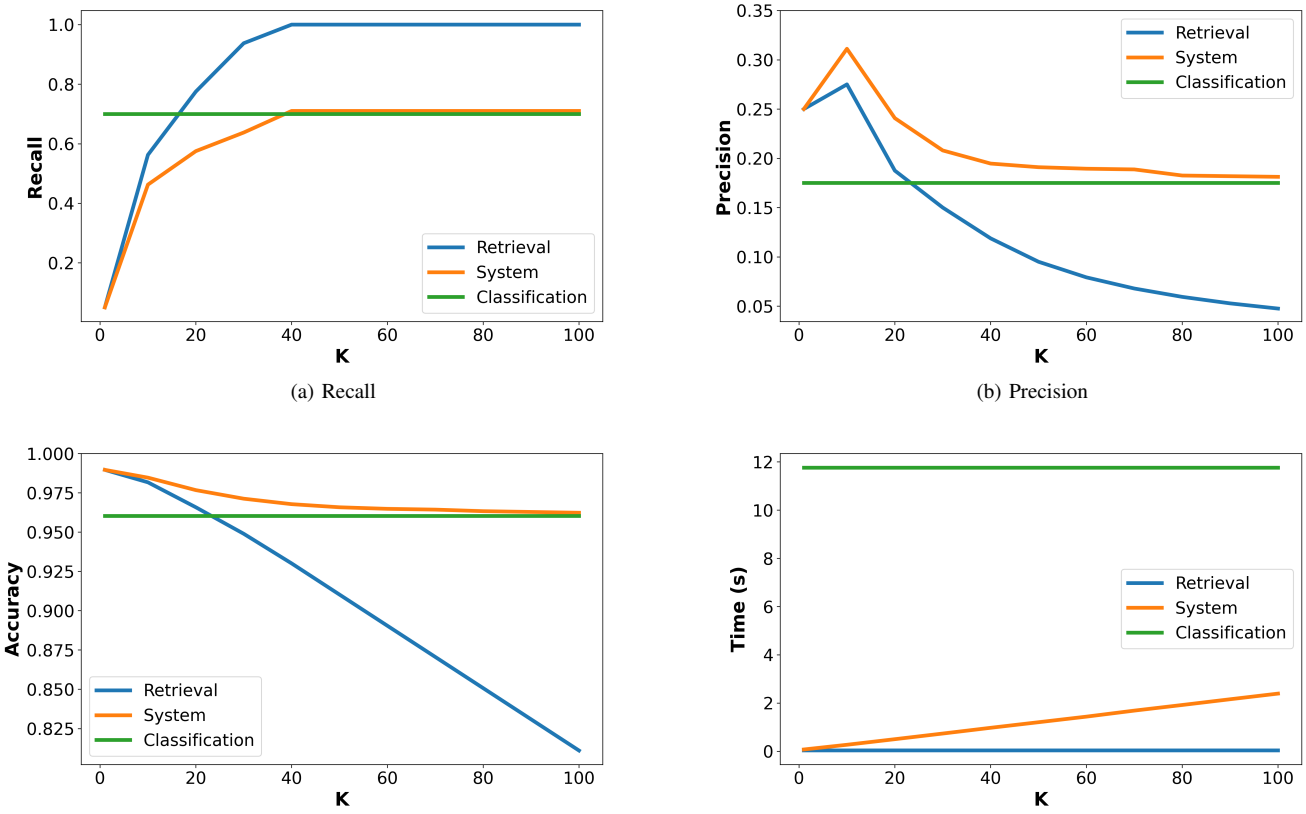
It is important to emphasize that, as shown in Figure 2, the performance of the classification model is not affected by changes in k, thus presenting a horizontal line in the figure.

In Figure 2a, we observed that at lower k, both the retrieval model and our system exhibit lower recall scores compared to the classification model. The reason is that the limited k prevents the retrieval of a large number of duplicate pairs. However, as k gradually increases, after reaching around 20, the recall of the retrieval model exceeds that of the classification model. At the same time, since the classification step of the system may introduce positive and false samples, the recall rate of the system is lower than that of the retrieval model, resulting in a decrease in the overall recall rate. Given that our system incorporates retrieval, such recall aligns with the rising trend demonstrated by retrieval models as k increases, albeit at a slightly slower pace. For example, it is not until ”k” equals 40 that the recall of our method starts to be comparable to that of classification. This suggests that as the value of k rises to higher values (e.g., over 100), our recall will continue to rise, thereby establishing an increasingly discernible gap from the recall of classification.

In the Figure 2b, we note that the precision of the retrieval model and our proposed system decrease as k increases, a consequence of increasing the number of retrieved candidates. Nevertheless, it is worth mentioning that compared with the retrieval precision, the decrease of system precision is not obvious. Even when k is equal to 100, the precision of the system is still higher than the classification precision, which demonstrates the effectiveness of our system in terms of precision. Similar to precision in Figure 2c, accuracy also exhibits a sonsistent trend. As the k increases, both the accuracy of the system and retrieval decrease. The decline in system accuracy is also slower compared to retrieval accuracy. This highlights the advantage of our system for introducing a classification step after retrieval, as it can efficiently preserve the performance in precision and accuracy, especially better than classification.

Comparatively, as displayed in Figure 2d, the classification

 ISBN: 978-1-68558-098-8 




(a) Recall

(b) Precision

(c) Accuracy

(d) Running Time

Fig. 2: Time-Performance Evaluation on Firefox Dataset in One VS All scenario

process is more time-consuming than both retrieval and our system. This observation can be explained through a simple calculation. Assuming we have $n$ user input bugs and $m$ database bugs, and assuming that both the classification model and the retrieval model require the same time for a single inference, the following holds: The classification model requires $n*m$ inferences. The retrieval model requires $n+m$ inferences, along with $n * m$ calculations of embeddings similarity and subsequent sorting. As the similarity calculation and sorting are considerably faster than model inference, the retrieval process roughly takes n + m seconds per inference. Our system incorporates the results of retrieval. Therefore, it includes the retrieval model inference time $n + m$, similarity calculation, and sorting time, followed by $n * topk$ classifications. Consequently, the system's required time amounts to $n+m+n*topk$ inferences. The preceding calculations are equally applicable to the All VS All scenario as well. It is clear that our method exhibits almost the same remarkable efficiency as retrieval and classification in terms of time. This is proven by the fact that the time consumed by the classification is approximately 60 times greater than our approach.

Figure 3 shows that the All VS All scenario exhibits similar performance trends as the One VS All scenario. The running time in Figure 3d represents the average time taken to match each bug with its similar ones, comparable to Figure 2d.

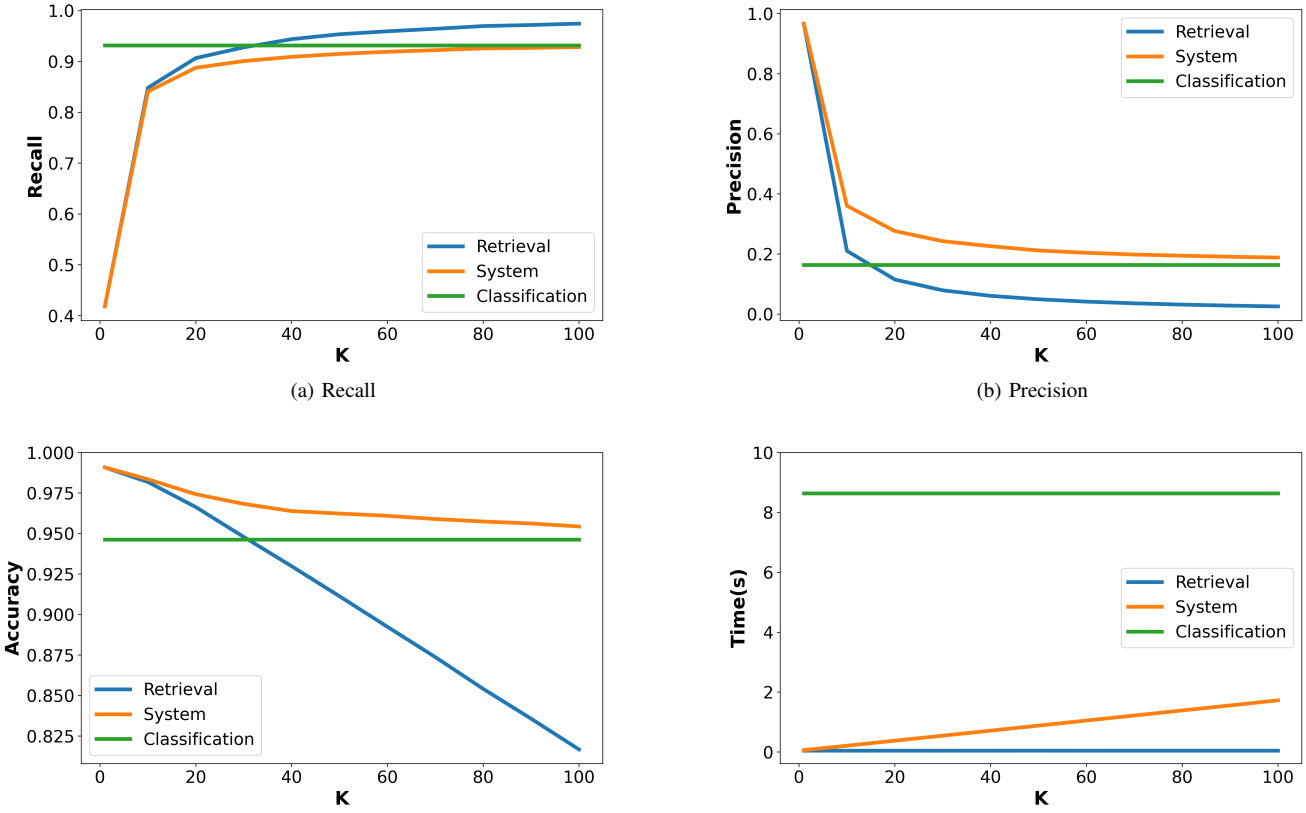
Overall, our system makes a trade-off by sacrificing some running time in order to maintain robust performance in terms of recall, precision, and accuracy. As k increases from 0 to 100, the recall of our system increases, demonstrating the ability of our model to successfully retrieve all relevant duplicates. At the same time, the accuracy and precision are only slightly reduced, effectively alleviating the sharp decline in retrieval, but never fall below those achieved by classification. The steady improvement in recall between retrieval and classification, coupled with the maintained superior precision, shows that we minimize the time cost while maintaining accuracy performance. This convincingly demonstrates our ability to obtain a trade-off between accuracy performance and time efficiency.

Therefore, selecting the appropriate value for k requires careful consideration. While a smaller k value may improve the time efficiency, it may also lead to a degradation in model performance. On the other hand, choosing a larger k value may result in increased time consumption. Thus, striking a balance between speed and model performance depends on selecting the optimal k value.

## VI. Conclusion and Future Work

In our work, we proposed a novel system based on the transformer models, that leverages the strengths of both re-

 ISBN: 978-1-68558-098-8 




(a) Recall

(b) Precision

(c) Accuracy

(d) Running Time

Fig. 3: Time-Performance Evaluation on Firefox Dataset in All VS All scenario

trieval and classification approaches for duplicate bug report detection task. We have evaluated the transformer-based models employed by our method on five datasets, demonstrating their effectiveness compared to traditional models for both classification and retrieval. More importantly, our method shows a competitive edge by achieving a balance between time efficiency and accuracy, compared to solutions employing only one of them. This advantage holds significant importance in real-time bug report detection where requires high-quality results in a short time. In other words, under resource constraints, combining retrieval and classification as a novel solution enables dynamic adjustments to efficiently address issues related to changes in data volume and quality, while flexibly adapting to time-sensitivity and shifts in user demands. This approach enhances resource efficiency and ensures the maintenance of response speed and accuracy in a constantly changing environment. Furthermore, our combined strategy can be expanded to tackle similar issues in other tasks, such as recommendation tasks.

While our system addresses the running time concern that previous methods overlooked, and achieves a trade-off between time and accuracy, there are several factors need to be considered for practical application, such as the size of the model. Our system relies on both the retrieval and classification models, resulting in a larger memory space requirement compared to a single model. As a result, future efforts could explore the possibility of employing multi-task learning to integrate these two models, allowing for the completion of both tasks with a single model simultaneously. Additionally, there are some limitations in our study that can be addressed in the future, such as expanding to new datasets. This not only includes datasets that are more current, but also those that are more diverse in terms of types, which would enhance the generalizability of our methods.

## ACKNOWLEDGMENT

This work was funded by the China Scholarship Council (CSC) and supported by the Leiden Institute of Advanced Computer Science (LIACS).

## REFERENCES

[1] S. Gupta and S. K. Gupta, “A systematic study of duplicate bug report detection,” *International Journal of Advanced Computer Science and Applications*, vol. 12, no. 1, pp. 578–589, 2021.

[2] B. S. Neysiani and S. Morteza Babamir, “Automatic duplicate bug report detection using information retrieval-based versus machine learning-based approaches,” pp. 288–293, 2020.

 ISBN: 978-1-68558-098-8

[3] M. B. Messaoud, A. Miladi, I. Jenhani, M. W. Mkaouer, and L. Ghadhab, "Duplicate bug report detection using an attention-based neural language model," *IEEE Transactions on Reliability*, pp. 1–13, 2022.

[4] T. Zhang, H. Jiang, X. Luo, and A. T. Chan, "A literature review of research in bug resolution: Tasks, challenges and future directions," *The Computer Journal*, vol. 59, no. 5, pp. 741–773, 2016.

[5] T. M. Rocha and A. L. D. C. Carvalho, "Siameseqat: A semantic context-based duplicate bug report detection using replicated cluster information," *IEEE Access*, vol. 9, pp. 44 610–44 630, 2021.

[6] A. e. a. Kukkar, "Duplicate bug report detection and classification system based on deep learning technique," *IEEE Access*, vol. 8, pp. 200 749–200 763, 2020.

[7] P. Runeson, M. Alexandersson, and O. Nyholm, "Detection of duplicate defect reports using natural language processing," in *29th International Conference on Software Engineering (ICSE'07)*. IEEE, 2007, pp. 499–510.

[8] J. Uddin, R. Ghazali, M. Mat Deris, R. Naseem, and H. Shah, "A survey on bug prioritization," *Artificial Intelligence Review*, vol. 47, pp. 145–180, 2017.

[9] A. T. Nguyen, T. T. Nguyen, T. N. Nguyen, D. Lo, and C. Sun, "Duplicate bug report detection with a combination of information retrieval and topic modeling," in *2012 Proceedings of the 27th IEEE/ACM international conference on automated software engineering*. IEEE, 2012, pp. 70–79.

[10] Q. Xie, Z. Wen, J. Zhu, C. Gao, and Z. Zheng, "Detecting duplicate bug reports with convolutional neural networks," in *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*. IEEE, 2018, pp. 416–425.

[11] J. He, L. Xu, M. Yan, X. Xia, and Y. Lei, "Duplicate bug report detection using dual-channel convolutional neural networks," in *Proceedings of the 28th International Conference on Program Comprehension*, 2020, pp. 117–127.

[12] V. Pankajakshan and M. Sridevi, "Detecting duplicate question pairs using glove embeddings and similarity measures," in *Advances in Automation, Signal Processing, Instrumentation, and Control*. Springer, 2021, pp. 695–702.

[13] H. e. a. Isotani, "Duplicate bug report detection by using sentence embedding and fine-tuning," in *2021 IEEE International Conference on Software Maintenance and Evolution (ICSME)*. IEEE, 2021, pp. 535–544.

[14] J. Deshmukh, K. Annervaz, S. Podder, S. Sengupta, and N. Dubash, "Towards accurate duplicate bug retrieval using deep learning techniques," in *2017 IEEE International conference on software maintenance and evolution (ICSME)*. IEEE, 2017, pp. 115–124.

[15] G. Xiao, X. Du, Y. Sui, and T. Yue, "Hindbr: Heterogeneous information network based duplicate bug report prediction," in *2020 IEEE 31st International Symposium on Software Reliability Engineering (ISSRE)*. IEEE, 2020, pp. 195–206.

[16] L. Hiew, "Assisted detection of duplicate bug reports," Ph.D. dissertation, University of British Columbia, 2006.

[17] X. Wang, L. Zhang, T. Xie, J. Anvik, and J. Sun, "An approach to detecting duplicate bug reports using natural language and execution information," in *Proceedings of the 30th international conference on Software engineering*, 2008, pp. 461–470.

[18] C. Sun, D. Lo, S.-C. Khoo, and J. Jiang, "Towards more accurate retrieval of duplicate bug reports," in *2011 26th IEEE/ACM International Conference on Automated Software Engineering (ASE 2011)*. IEEE, 2011, pp. 253–262.

[19] A. Alipour, A. Hindle, and E. Stroulia, "A contextual approach towards more accurate duplicate bug report detection," in *2013 10th Working Conference on Mining Software Repositories (MSR)*. IEEE, 2013, pp. 183–192.

[20] A. Lazar, S. Ritchey, and B. Sharif, "Improving the accuracy of duplicate bug report detection using textual similarity measures," in *Proceedings of the 11th Working Conference on Mining Software Repositories*, 2014, pp. 308–311.

[21] K. e. a. Aggarwal, "Detecting duplicate bug reports with software engineering domain knowledge," *Journal of Software: Evolution and Process*, vol. 29, no. 3, p. e1821, 2017.

[22] A. Budhiraja, K. Dutta, R. Reddy, and M. Shrivastava, "Dwen: deep word embedding network for duplicate bug report detection in software repositories," in *Proceedings of the 40th International Conference on software engineering: companion proceeedings*, 2018, pp. 193–194.

[23] A. Budhiraja, K. Dutta, M. Shrivastava, and R. Reddy, "Towards word embeddings for improved duplicate bug report retrieval in software repositories," in *Proceedings of the 2018 ACM SIGIR International Conference on theory of information retrieval*, 2018, pp. 167–170.

[24] L. e. a. Poddar, "Train one get one free: Partially supervised neural network for bug report duplicate detection and clustering," *arXiv preprint arXiv:1903.12431*, 2019.

[25] Y. Jiang, X. Su, C. Treude, C. Shang, and T. Wang, "Does deep learning improve the performance of duplicate bug report detection? an empirical study," *Journal of Systems and Software*, p. 111607, 2023.

[26] F. Rahutomo, T. Kitasuka, and M. Aritsugi, "Semantic cosine similarity," in *The 7th international student conference on advanced science and technology ICAST*, vol. 4, no. 1, 2012, pp. 1–2.

[27] N. Reimers and I. Gurevych, "Sentence-bert: Sentence embeddings using siamese bert-networks," *CoRR*, vol. abs/1908.10084, 2019. [Online]. Available: http://arxiv.org/abs/1908.10084

[28] I. Loshchilov and F. Hutter, "Fixing weight decay regularization in adam," *CoRR*, vol. abs/1711.05101, 2017. [Online]. Available: http://arxiv.org/abs/1711.05101

[29] P. Bojanowski, E. Grave, A. Joulin, and T. Mikolov, "Enriching word vectors with subword information," *CoRR*, vol. abs/1607.04606, 2016. [Online]. Available: http://arxiv.org/abs/1607.04606

 ISBN: 978-1-68558-098-8 