Final Project Report: Replication of Towards Accurate Duplicate Bug Identification and Retrieval Using Deep Learning Techniques

Guneet Kaur
kaurg33@myumanitoba.ca
University of Manitoba
Winnipeg Manitoba, Canada

ABSTRACT

Bug tracking systems play a crucial role in software maintenance and development. The majority of software companies require proper bug maintenance and tracking systems that assist them in managing bugs along with their fixes. With an increase in software's scale and complexity, it becomes more bug-prone. Huge client base software companies usually report duplicate bugs. These bugs need to be manually identified by the engineers to avoid re-work. Thus one can't manually identify the redundant bugs. It leads to unscalability and an increase in the project time and cost. An increase in the rise and research fields like artificial intelligence and machine learning leads to better management of the DBDR (Duplicate Bug Detection and Retrieval). To save time, we aim to utilize them in automating the process of duplicate bug detection. Also, we employ NLP (Natural language processing) techniques to retrieve similar bugs that appeared in the past. We consider Deshmukh et al. [10] as a reference and perform a replication of the paper on a new dataset to test the generalizability of the technique in addition to proposing a different architecture to achieve the same goal. We present detailed experimental results, demonstrating the robustness of the approach and effectiveness of the proposed model.

KEYWORDS

NLP, Convolution Neural Networks, Recurrent Neural Networks, Bi-LSTM's, Word Embeddings

1 INTRODUCTION

Softwares go through the phases of the software development life cycle (SDLC) as it helps to produce software of the highest quality and low cost in a lesser amount of time. There are six different phases in the software development life cycle. "Software maintenance" is one of the most vital parts of the life cycle. It contributes to about 2/3rd of the project cost. Hence, it takes an immense of time to make software bug-free. Big tech companies host bug tracking systems like Jira, Bugzilla. It helps in managing and keeping a record of their bugs and fixes. The software testing team and the end-users make up the majority of the consuming entities. They report the bugs in a highly disorganized manner. Due to this, many times a bug gets reported frequently. It leads to engineers reworking on the bug prioritization and its solution. In turn, it reduces efficiency and productivity. Therefore, it becomes essential to have some mapping between a newly reported bug and previous similar bugs. Moreover, having this sort of mapping will aid in finding the corresponding fixes for similar bugs that were fixed earlier.

For large software projects in the industry, manual identification of the bugs and finding similar past bugs becomes very time-consuming and costly, and hence, infeasible. Therefore, it becomes crucial to develop an automating solution for detecting bugs and retrieving similar past bugs. A typical bug tracking system includes the master list for storing all the bugs. Whenever a new bug is reported, the system checks the master list. If it is a duplicate, the bug is flagged. Then corresponding fixes are looked upon. If not, that bug is marked as non-duplicate and added to the master list along with its solution. Numerous strategies have been tried in the past for this problem. Since the bug report contains both structured and unstructured information about the bug, Natural Language is employed for its description. And the linguistic variance to express the similar bugs makes the task of duplicate bug identification arduous.

The earlier approaches mentioned are based on ML and Information Retrieval but are highly inaccurate. This paper focuses on using "Siamese networks" as described in [10] to test the generality and replicating their technique on a new dataset. Siamese Networks are a combination of CNNs and RNNs. The text proposes them for duplicate bug identification and retrieval from bug reports. It is important to highlight their novelty of using a deep learning model for duplicate bug detection. Furthermore, it becomes interesting to have deeper insights and explore more on the approach. The primary contributions of this paper can be summarized as follows:

- We use Siamese Networks as presented in [10] for detection and retrieval of duplicate / similar bugs. Here, we use both structured and unstructured information available in the bug reports.
- Different loss objective functions are used for the retrieval and classification models. Max Margin Objective function is used to distinguish similar from non-similar bugs.
- We present detailed experimental results by replicating the technique on a new dataset Mozilla made available by Lazar et al. [22] just like they did on three large published datasets (NetBeans, Eclipse, and Open Office).
- We present a proposed model by replacing the BiLSTM network with Attention Based LSTM networks.
- We test the project in a cross-project setting to verify its effectiveness in a scenarios where labelled training data is lacking and we perform training on other projects belonging to the same domain.

Following research questions are consider to see the effectiveness of the approach:

- RQ 1: Does Accuracy and Recall generalizewell on the new dataset (Mozilla)?
- RQ 2: Can we deduce some generalizedtrends in bug reports? If yes, what are theprimary recurring bugs?
- RQ 3: Can some other strategies like attention mechanism or Tree-LSTM insteadof Bi-LSTM help in improving?
- RQ 4: How efficient is the approach in across-project detection setting?

The paper is divided in sections as follows. We cover related work in the next section. Data collection section 3 covers the steps and resources for collecting the data for the experiment. Later, we present our methodology. Various research questions along with their approach and results are presented in section 5. We present limitations in section 6. Finally, we present conclusion in section 7 by discussing future work.

2 RELATED WORKS

Researchers have implemented a variety of unsupervised and supervised approaches for duplicate bug identification. Here, we provide an overview of the relevant past work that focused on detecting duplicate bug reports:

In the beginning, we present the related work in characterizing duplicate bugs. Then approaches are explained by dividing them into classification and retrieval. First step to determine whether two bugs are similar or not was taken by Runeson et al. [32]. They employed NLP text processing techniques including stop word removal, stemming, and tokenizing to for processing each report. To compare similarity between two bug reports, cosine similarity score was considered. They were able to detect 40% of duplicate bug reports successfully. Kim et al. [18] used graphs-vector space model to extract the feature vectors and use cosine similarity to find the duplicity in 2 or more bugs.

Kucuk et al. [19] perform an empirical analysis for characterizing duplicate bugs. Whilst analysis, they state that different costs are required to resolve the different types of duplicate bugs. For open-source projects, users have the freedom to report the bugs. Which leads to the report of many redundant bugs. They categorize the repeated bugs by grouping them according to the severity and resolution of the master bug and further analysis its relationship with the master bug. Rocha et al. [29] characterize the bugs in Mozilla based on the workflow of developers in resolving those bugs.

Scientists have put forward several approaches to detect and classify duplicate bugs. Many of them save time and optimize the cost of manual intervention. Xu et al. [15] implemented a similar duplicate bug detection using dual-channel CNN. They used both the structured and unstructured information of the bug report. They constructed bug report pairs using a dual-channel matrix. Then they fed it into the CNN to see the association of the bug pairs. Xie et al. [39] also implemented CNN's for detecting duplicate bug reports.

There exists two main strategies: Information Retrieval and Machine Learning for detecting duplicate bug reports. Soleimani et al. [25] considers both the approaches and aims to compare their performances. They utilized Android dataset for experiment and showed that ML based strategy give better results in terms of recall, precision and accuracy. In another work, Budhiraja et al. [4] presents usage of word embedding, for instance, CBOW for the retrieval of duplicate bug reports. They extract the summary and description of the bug, modelling each bug as the dense vector. It employs embedding to convert into a numerical representation to retrieve its similar reports.

Rodrigues et al. [31] also focuses on detecting duplicate bugs using machine learning techniques. They aim to increase software quality by decreasing triage team load. They employed three approaches: Decision Making, Binary Classification and Ranking. Since most of the works use machine learning methods for duplicate bug detection, it becomes crucial to consider best features for model training and best results. Sabor et al. [33] presents DURFEX - a feature extraction technique for detecting duplicate bugs efficiently. They aim to classify bugs using functions invoked in the stack trace. Since there exists millions of stack traces, their approach may suffer from high dimensional problem. They use feature extraction procedure to reduce the feature size, keeping only the features that are most relevant. Another work by Koopaie E. [11] also utilizes stack traces for detecting duplicate bugs. Stack traces proves to be advantageous because of the reduced number of pre-processing tasks like text processing using NLP. They evaluated the approach on GNOME and Firefox datasets and compared results to those of state-of-the-art approaches.

Machine learning algorithms are of two types: supervised and unsupervised learning. Unsupervised learning algorithms for problem of duplicate bug detection involves using similarity scores, ranking them and retrieving top k similar bugs based on the scores. Some other unsupervised approaches use topic modeling techniques like Latent Dirichlet Allocation (LDA) in order to calculate similarity scores. Budhiraja et al. [5] use combination of both LDA and word embeddings for duplicate bug report detection and evaluate their approach on dataset of Firefox. Lazar et al. [23] focused on improving accuracy of duplicate bug report detection using text similarity features.

Machine learning and IR based approaches are the two most powerful approaches for bug triaging. Goyal et al. [13] takes into consideration and performs a deep investigation on which approach is better. According to their study, IR based approaches yields better results. A lot of work has been done in detecting duplicate bugs using contextual features. Hindle et al. [17] and Alipour et al. [1] used contextual features about software attributes and architecture terms can result in better duplicate bug detection. In recent works by Rocha et al. [30] instead of considering individual bug, considers a cluster of bugs for extracting information. This is done on the basis that a bug can have multiple duplicates, hence multiple descriptions. Their technique called SiameseQAT uses both context and semantic learning. They were able to achieve recall of 85% using top 25 bugs and 84% ROC score on the datasets of Eclipse, Open Office and NetBeans. Kukkar et al. [20] aims to automate the process of duplicate bug detection using deep learning techniques. They consider similarity measurement too in account and utilize CNN model architecture to extract the features. A similar work of detecting duplicity using deep learning techniques by Wang et al. [37] aims to identify duplicate stack overflow questions using CNN, RNN and LSTM models.

3 DATA COLLECTION

The original paper focused on three bug datasets (Open Office, Eclipse, and NetBeans) published by [22]. Open Office is an open-source software used for productivity. NetBeans and Eclipse are Java integrated development environments (IDEs). They created a larger combined dataset by combining the aforesaid datasets. For this project, we are using the bug dataset of Mozilla [22]. Bug datasets contain multiple bug reports wherein each report describes the bug using its properties, for instance, bug_id, product, description, bug_severity, short_description, priority, dup_id, version, component, status, resolution, etc. They use 'bug_id' to uniquely identify a bug, 'description' & 'short_desc' both are used to describe the bug in a detailed and brief manner respectively. Table 1 shows a sample bug with all its properties and values. [22] provides pairs datasets for Open Office, NetBeans, and Eclipse but not for Mozilla. Therefore, we describe the steps for generating pairs for Mozilla.

Table 1: Sample Bug Properties

Bug Property	Value
"_id"	ObjectId("52eaece754dc1c410c4fbc12")
"bug_id"	"102"
"product"	"MozillaClassic"
"description"	"Created by Darrell Kindred (dkindred=mozilla@cs.cmu.edu) on Tuesday, April 7, 1998 10:57:25 AM PDT. Additional Details :Pressing Alt-w seems to have no effect in the Page Source window. Updated by Steve Lamm (slamm@netscape.com) on Thursday, April 30, 1998 10:26:04 AM PDT. Additional Details : I have this on my internal buglist too, so I will take it."
"bug_severity"	"minor"
"dup_id"	"537"
"short_desc"	"Alt-w doesn"t close view-source window"
"priority"	"P2"
"version"	"1998-03-31"
"component"	"XFE"
"delta_ts"	"2008-06-17 06:13:19 -0700"
"bug_status"	"VERIFIED"
"creation_ts"	"1998-04-07 17:57:25 -0700"
"resolution"	"DUPLICATE"

3.1 Cleaning & Processing Bugs Dataset

The first step towards generating pairs dataset starts with checking whether bug_id presented in dup_id field exists in the dataset. If no such bug is present and resolution is "DUPLICATE", we change the resolution of such bugs from "DUPLICATE" to "NDUPLICATE".

Next, if for a bug, dup_id is an empty field whereas the resolution field says "DUPLICATE", we again change the resolution of such bugs from "DUPLICATE" to "NDUPLICATE". Cycle problem exists when two bugs are duplicates of each other. In such scenarios, it becomes difficult to figure out which is the master bug. To resolve this, we consider the recent one as the master bug and the other one duplicate of the former bug. Lastly, we remove "OPEN" bugs. A bug is considered as "OPEN" until its resolution is decided. Since our goal is to classify a bug as duplicate or non-duplicate, we are considering only "CLOSED" bugs for training and testing.

3.2 Creating Groups & Pairs

The figure 1 shows that duplicate bugs constituted 23% of a total number of bugs making a count of duplicate bugs lesser than non-duplicate bugs. In order to generate bugs pairs for the training classification model, we first arrange bugs in groups. A group consists of a master bug and a list of all other bugs that are duplicates of the master bug represented by their *dup_id*.

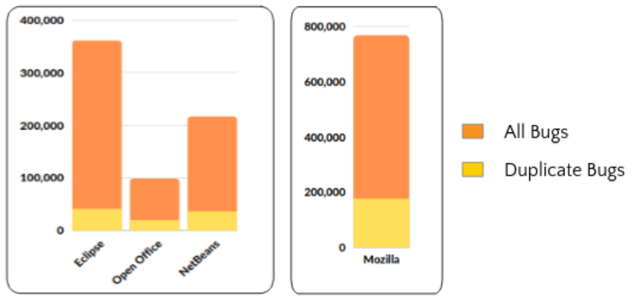


Figure 1: Number of total and duplicate bugs

Let's assume 5 bugs, *bugA, bugB, bugC, bugD and bugE* of which *bugA* is the master bug as it has no *dup_id*. Bugs *bugB, bugC, bugD and bugE* contains *bugA.bug_id* in their *dup_id* field making all bugs *bugA, bugB, bugC, bugD and bugE* constitute a group. Since the count of duplicate bugs is already less than the count of non-duplicate bugs, we intend to generate maximum possible duplicate pairs dataset. In order to do this, we consider all the pairs rather than just focusing on the pairs containing the master bug [22]. For the example considered above, following 10 duplicate pairs will be generated: *(bugA, bugB), (bugA, bugC), (bugA, bugD), (bugA, bugE), (bugB, bugC), (bugB, bugD), (bugB, bugE), (bugC, bugD), (bugC, bugE), (bugD, bugE).*

The table 2 shows the number of bugs and pairs generated for the Mozilla dataset as well as for the three datasets considered in the original paper. The Mozilla dataset is larger as compared to Eclipse, Open Office, and NetBeans. We also plotted a histogram of log of several groups versus group size for Mozilla as was done for eclipse in [22] as shown in Figure 2. Similar to eclipse, the Mozilla dataset had most of the groups of size 1. And having just one other duplicate bug.

Table 2: Number of groups and pairs in Existing Paper v/s Replication

Dataset	Groups	Max Group Size	Duplicate Pairs
Mozilla	79,872	588	11,491,116
Eclipse	25,455	51	110,137
Open Office	7,257	90	111,121
NetBeans	19,779	56	148,343

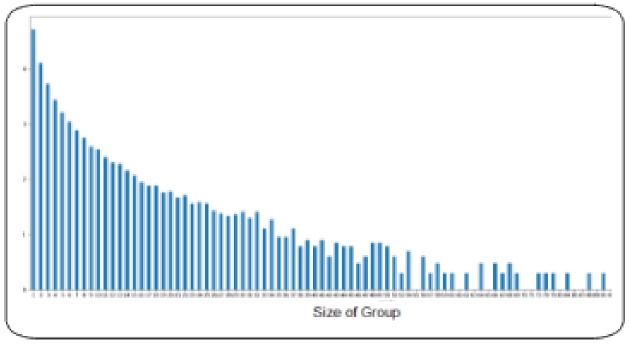


Figure 2: Histogram for Mozilla

To create complete pairs, both duplicate and nonduplicate pairs are generated. We generate non-duplicate pairs equal to 4 times the duplicate pairs. Pairs dataset was used for training the classification model. For retrieval model, we create triplets of form *(b, b+, b−−)* where b is the bug, b+ is its duplicate, and b− is it's non-duplicate. Table 3 shows the number of bug pairs in the train and test splits (using 80−20%) for the Mozilla dataset.

Table 3: Train-Test Split

	Train	Test
Mozilla	4,596,464	1,149,116

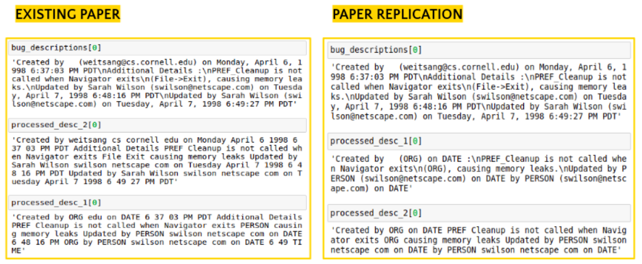
4 METHODOLOGY

We plan on using a deep learning model for duplicate bug detection and retrieval. The two-fold methodology is used: using word embeddings to convert the bug descriptions to numerical representations. These are then fed to the proposed model for training. The classification model is used to detect if a bug is duplicated or not. For a duplicate bug, a retrieval model identifies the top k similar past bugs. All the three parts of bug descriptions: structured information, a short description, and a long description after processing are inputted to different models. We discuss here two models used for training: baseline and proposed model. The baseline model is the same as presented in the original paper [10].

4.1 Data Pre-processing

Dataset here consists of bug reports. In the data collection section, we already discussed how to create a duplicate and non−duplicate pairs dataset from groups to be used for the classification model. We also create triplets of the form (b, b+, b−−) for the retrieval model. A bug report contains both structured and unstructured information. Structured information includes information related to priority, component, product, version, etc. Unstructured information on the other hand includes bug descriptions, both short and long. All three types of information: structured information, a short description, and a long description need to be processed separately before they can be fed to the next stage of model training.

Text pre-processing steps are done on the bug descriptions, both short and long. In the current paper, they cleaned the text by removing all the non-alphanumeric characters. Next, they used Named Entity Recognition Module (NER) module to process the sentences wherein the named entities were replaced with keywords like "PERSON", "DATE", "ORG" etc. In the contrast, here we reverse the order. We pass the sentences through the NER module. And later remove the special characters. This is done because as can be seen from Figure 3 if we first remove nonalphanumeric characters, ":" character within date and time gets removed and NER module fails to recognize it as time but if we first pass through the NER module, it gets successfully replaced with "DATE". Next, the text is converted to lower case after which it is tokenized into words. This is followed by creating word embedding using GloVe word embeddings [27]to form word vectors. GloVe embeddings are used to get an idea of the co-occurrence of 2 words and it does so by taking the dot product of two vectors. There exist two other similar models proposed by Mikolov et al. [24]: CBOW (Continous Bag of words) and Word2Vec Models. CBOW model predicts the present word given the words that surround it, whereas the Word2Vec model predicts the surrounding words given the current word.


EXISTING PAPER

```
bug_descriptions[0]
'Created by   (weitsang@cs.cornell.edu) on Monday, April 6, 1
998 6:37:03 PM PDT\nAdditional Details :\nPREF_Cleanup is not
called when Navigator exits\n(File->Exit), causing memory lea
ks.\nUpdated by Sarah Wilson (swilson@netscape.com) on Tuesda
y, April 7, 1998 6:48:16 PM PDT\nUpdated by Sarah Wilson (swi
lson@netscape.com) on Tuesday, April 7, 1998 6:49:27 PM PDT'

processed_desc_2[0]
'Created by weitsang cs cornell edu on Monday April 6 1998 6
37 03 PM PDT Additional Details PREF Cleanup is not called wh
en Navigator exits File Exit causing memory leaks Updated by
Sarah Wilson swilson netscape com on Tuesday April 7 1998 6 4
8 16 PM PDT Updated by Sarah Wilson swilson netscape com on T
uesday April 7 1998 6 49 27 PM PDT'

processed_desc_1[0]
'Created by ORG edu on DATE 6 37 03 PM PDT Additional Details
PREF Cleanup is not called when Navigator exits PERSON causin
g memory leaks Updated by PERSON swilson netscape com on DATE
6 48 16 PM ORG by PERSON swilson netscape com on DATE 6 49 TI
ME'
```

PAPER REPLICATION

```
bug_descriptions[0]
'Created by   (weitsang@cs.cornell.edu) on Monday, April 6, 1
998 6:37:03 PM PDT\nAdditional Details :\nPREF_Cleanup is not
called when Navigator exits\n(File->Exit), causing memory lea
ks.\nUpdated by Sarah Wilson (swilson@netscape.com) on Tuesda
y, April 7, 1998 6:48:16 PM PDT\nUpdated by Sarah Wilson (swi
lson@netscape.com) on Tuesday, April 7, 1998 6:49:27 PM PDT'

processed_desc_1[0]
'Created by   (ORG) on DATE :\nPREF_Cleanup is not called whe
n Navigator exits\n(ORG), causing memory leaks.\nUpdated by P
ERSON (swilson@netscape.com) on DATE by PERSON (swilson@netsc
ape.com) on DATE'

processed_desc_2[0]
'Created by ORG on DATE PREF Cleanup is not called when Navig
ator exits ORG causing memory leaks Updated by PERSON swilson
netscape com on DATE by PERSON swilson netscape com on DATE'
```

Figure 3: Existing Paper v/s Replication

4.2 The Baseline Model

The baseline model consists of three models: RNN, CNN and LSTM's which are described next.

4.2.1 *Recurrent Neural Network (RNN) and Long Short Term Memory (LSTM).*

These neural networks are fed with the numerical representation (word vectors) of the sequence of words to compute the output. Please note that while computing, output from the previous word is considered as feedback. As the name suggests, recurrence neural networks perform this computation for every word. Equations 4.2.1 represents the RNN. RNN's suffer from the problem of vanishing gradient and have difficulty in learning long sequences as explained by Bengio et al. [3]. To solve this, LSTM's were introduced. LSTM's are of different types. For example, single-layer LSTM, bi-LSTM, Tree LSTM, etc. They make use of hidden states and gates to learn the content. Forward LSTM's take the actual order of the input sequence, while backward LSTM's consider the input in the reverse order. Bi-LSTM [28] networks used here take into account both forward and backward LSTM's.

Bug's structured information is encoded using a single layer network ($\sigma(W^T.b)$) to output I_b. After converting it to a word vector, the short description of the bug is encoded using Bi-LSTM to output S_b.

$$RNN(t_i) = f(W * xt_i + U * RNN(t_{i-1}))$$

4.2.2 Convolutional Neural Networks (CNN).
The long description property of the bug contains the bug description presented over a long text. LSTM networks are not good enough to handle long sequences. Convolutional neural networks are employed for the task. Although CNN's were used for images, nowadays they have been used for text classification problems [41]. CNN accounts filters that use a partial input, send into the network, compute the output. This filter spans over the entire input. Every computed output is then combined using a pooling layer to form a reduced input presentation. Different types of pooling layers are used to aggregate the outputs. For instance, min pooling, max pooling and mean pooling. For textual problems, the filter size is also known as window size, which keeps on moving forward in the right direction until the entire text is exhausted. We can use the final encoding for prediction. For our task, long descriptions are encoded using a stack of CNN layers with max-pooling and varied filter sizes to form an encoding L_b.

$$For\ all\ i \in 0, s, 2s, .., C_i = f(W * x + i : i + h - 1 + b)$$

Final Embedding F_b is then formed by combining embeddings I_b, S_b, L_b. Using these embeddings, two models are trained: the classification model and the retrieval model. We use the max-margin training loss objective function (Equation 4.2.2) to train the retrieval model since our goal is to obtain a high score for similarity bugs and a lower score for dissimilar bugs.

$$Max\ [0, M - Cosine(F_b, F_{b+}) + Cosine(F_b, F_{b-})]$$

The complete model can be viewed as a combination of Siamese LSTM and CNN figure 4. Each sample in the training set is a combination of 2 pairs (b, b+) and (b, b−). Two networks with their shared weights encode these two pairs of bugs. We use cosine similarity to compute the similarity score. The score of new bugs is computed with those present in the master list. Then we use a threshold value to retrieve top k similar bugs. The classification model differs from the retrieval model in terms of its objective function. Given two bugs, our goal is used to find whether they are duplicates of each other or not. For this task, let's say we have bugs a and b, their encodings are fed into a two-layer network and the softmax function is applied to compute the final prediction as shown in Equation 4.2.2. When a new bug is encountered, its pairs are formed using all other bugs present and a trained model is utilized to predict whether pairs are duplicate or not.

$$label_{predict}(a, b) = softmax(\sigma((W_2^i)^T * \sigma((W_1^i)^T [F_a : F_b])))$$

4.3 The Proposed Model

To answer our research question, we propose a different architecture. Our objective is to compare whether we see an increase in accuracy and recall if we use a different type of LSTM network, for instance, Tree-Based LSTM or Attention Based LSTM instead of Bi-LSTM. The current approach [10] does not have any attention mechanism. Xu et al. [40] used an attention mechanism to generate captions in an image. Attention Mechanism helps to selectively focus on some information while ignoring other things in the network. It emerged as an improvement over LSTM's and RNN's in NLP problems and was introduced to address the problem of long sequences. So we intend to incorporate an appropriate attention mechanism to enable automatic learning prioritized part of the bug, to yield better results.

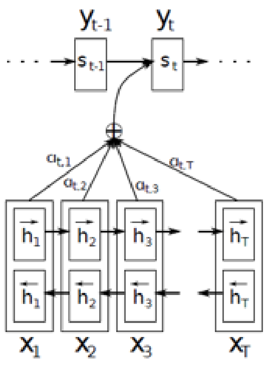
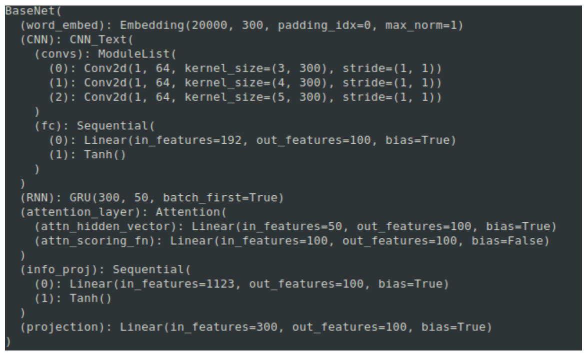

y_{t-1}
y_t
s_{t-1}
s_t
$\alpha_{t,1}$
$\alpha_{t,2}$
$\alpha_{t,3}$
$\alpha_{t,T}$
$\overrightarrow{h_1}$ $\overrightarrow{h_2}$ $\overrightarrow{h_3}$ $\overrightarrow{h_T}$
$\overleftarrow{h_1}$ $\overleftarrow{h_2}$ $\overleftarrow{h_3}$ $\overleftarrow{h_T}$
X_1 X_2 X_3 X_T

Figure 5: The Attention Mechanism[2]

We modify the architecture to include attention-based LSTM instead of BiLSTM as can be seen from Figure 6. Deng et al. [9] proposed a text classification model based on attention-based BiLSTM fused with CNN networks wherein key information was extracted from the context of textual information. In another work by Xie et al. [38] used an attention-based layer for the task of sentiment classification.

We developed the models employing PyTorch. It is an open-source machine learning framework. For GPU, we have utilized Google Colab[6]-product from Google research that makes available GPU for faster training. We even executed some of our code and training on Kaggle as it provides 16GB of RAM and can better handle the data.



```
BaseNet(
  (word_embed): Embedding(20000, 300, padding_idx=0, max_norm=1)
  (CNN): CNN_Text(
    (convs): ModuleList(
      (0): Conv2d(1, 64, kernel_size=(3, 300), stride=(1, 1))
      (1): Conv2d(1, 64, kernel_size=(4, 300), stride=(1, 1))
      (2): Conv2d(1, 64, kernel_size=(5, 300), stride=(1, 1))
    )
    (fc): Sequential(
      (0): Linear(in_features=192, out_features=100, bias=True)
      (1): Tanh()
    )
  )
  (RNN): GRU(300, 50, batch_first=True)
  (attention_layer): Attention(
    (attn_hidden_vector): Linear(in_features=50, out_features=100, bias=True)
    (attn_scoring_fn): Linear(in_features=100, out_features=100, bias=False)
  )
  (info_proj): Sequential(
    (0): Linear(in_features=1123, out_features=100, bias=True)
    (1): Tanh()
  )
  (projection): Linear(in_features=300, out_features=100, bias=True)
)
```

Figure 6: The proposed model

5 RESEARCH QUESTIONS

5.1 RQ1: Does Accuracy and Recall generalize well on the new dataset (Mozilla) ?

5.1.1 Motivation.
Since in this study, our objective is to replicate the baseline model

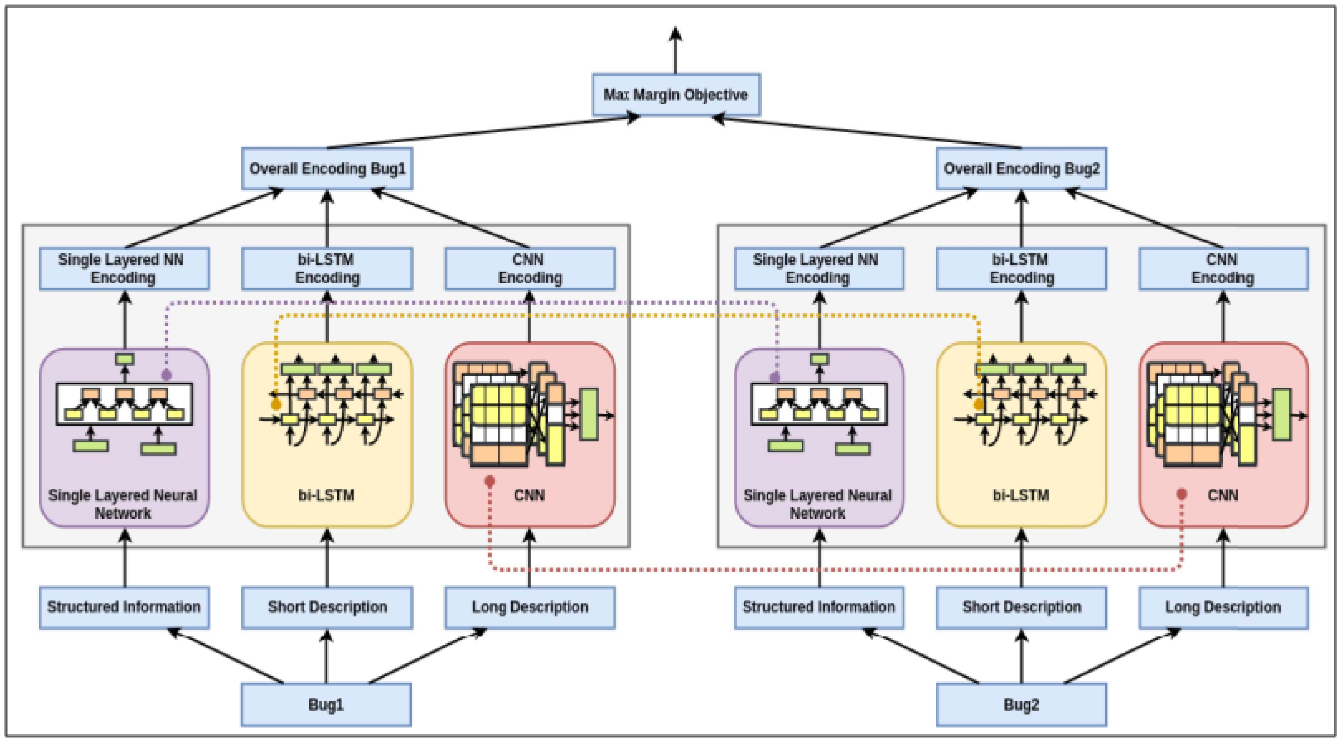


Figure 4: The Baseline Model

approach on a new dataset (Mozilla), we intend to check the generalizability of the technique by evaluating the same metrics for the recent dataset. We report accuracy and recall as evaluation metrics similar to the original paper. Here, [10] claims to have a recall rate of 80%. We aim to see if this model generalizes well to other datasets and gives outperforming results or not.

5.1.2 *Approach.*

We processed the dataset as explained in section 3 and created batch triplets. The Mozilla dataset was huge compared to Eclipse or NetBeans or Open Office. A total number of bugs and bug pairs are listed in Table 4. We trained the baseline classification and retrieval model as explained in section 4 after splitting the dataset in the train-test split of 80%-20%.

Table 4: Number of bugs and pairs

Number of bugs	Number of Pairs
779457	5,745,580

5.1.3 *Results.*

The model was trained for about 30 epochs. Due to the overwhelming dataset size and lack of GPU resources, we took a subset of the data for training. Loss v/s epoch graph can be seen from Figure 7. Accuracy comparison of existing paper v/s replication is evident from Figure 8. Additionally, the similar plot of recall rate versus different values of K is plotted to depict how the recall rate varies when k varies in the case of the retrieval model. An almost similar trend can be seen from the existing paper and our replication as is evident from Figure 9. These plots depict, that accuracy and recall scores do generalize well on the new dataset (Mozilla).

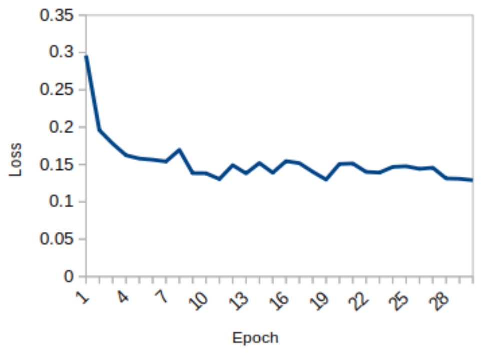
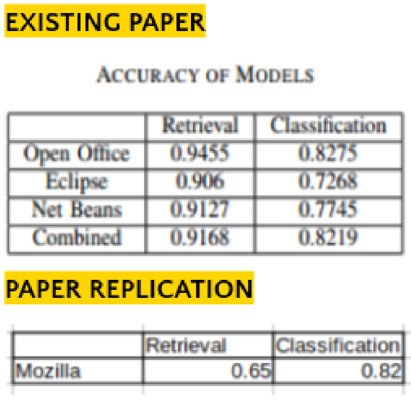


Figure 7: Loss v/s Epoch

5.2 RQ2: Can we deduce some generalized trends in bug reports? If yes, what are the primary recurring bugs?

5.2.1 *Motivation.*

This research question aims to find out if some patterns exist even when talking about similar bugs. It may be the case, that a set of bugs may be similar to each other as reported by the model



EXISTING PAPER

Accuracy of Models

	Retrieval	Classification
Open Office	0.9455	0.8275
Eclipse	0.906	0.7268
Net Beans	0.9127	0.7745
Combined	0.9168	0.8219

PAPER REPLICATION

	Retrieval	Classification
Mozilla	0.65	0.82

Figure 8: Accuracy

but at the same time, they can be associated more. For instance, a module or component may be erroneous in the software and most of the time, bugs are related to that particular module. We aim to discover such patterns if they exist. We also aim to perform feature selection [8]. Feature selection is an important step for machine learning modeling as it greatly impacts the training and results. In most cases, not all the features contribute equally towards model training. Hence, they might be noisy for the model. Such features hinder the model's performance.

5.2.2 Approach.
We start by doing EDA (Exploratory Data Analysis) which is a crucial step before utilizing the model for training [36]. Results of EDA are displayed in the next section. Hanam et al. [14] in his work also aims at locating bug patterns in JavaScript using AST (Abstract Syntax Trees). We can also utilize statistical techniques like ANOVA, Factor Analysis to look for variance in two groups to figure out the relationship between two groups [26].

5.2.3 Results.

- **Top 5 Products** Figure 10 displays the top 5 products that exist in the bugs dataset of Mozilla. It is evident that maximum bugs are from product "Core" and "Firefox". When a new bug is reported, rather than pairing with every bug in the master list and using the classification model to check which pairs are duplicates, to save the computation and resources, it is recommended to first check for these products.

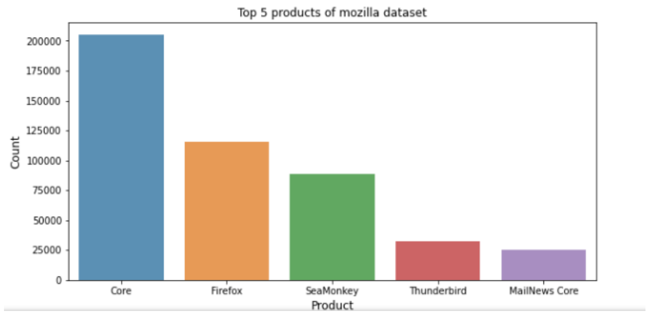


Figure 10: Top 5 Products

- **Top 5 Components** Figure 11 displays the top 5 components that exist in the bugs dataset of mozilla. Maximum bugs are from the "General" component. Similar to products, it would be recommended to first check bugs related to this component for duplicity when a new bug is reported.

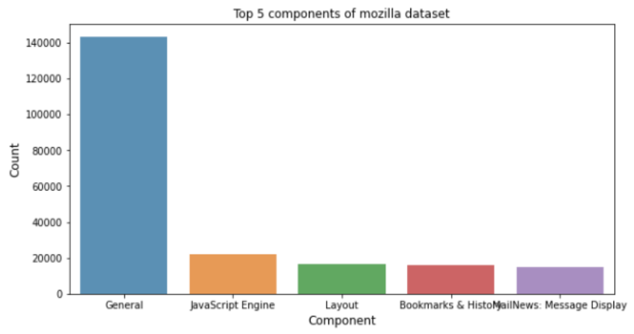


Figure 11: Top 5 Components

- **Word Cloud** Figure 12 displays the word cloud constructed for duplicate bug reports. Text-based analysis can be done using word cloud as done by Heimer et al. [16]. Word Cloud visualization provides an overview of what is being talked about on the topic.

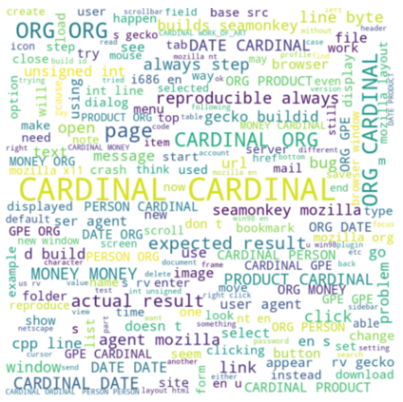


Figure 12: Word Cloud for Duplicate Bugs

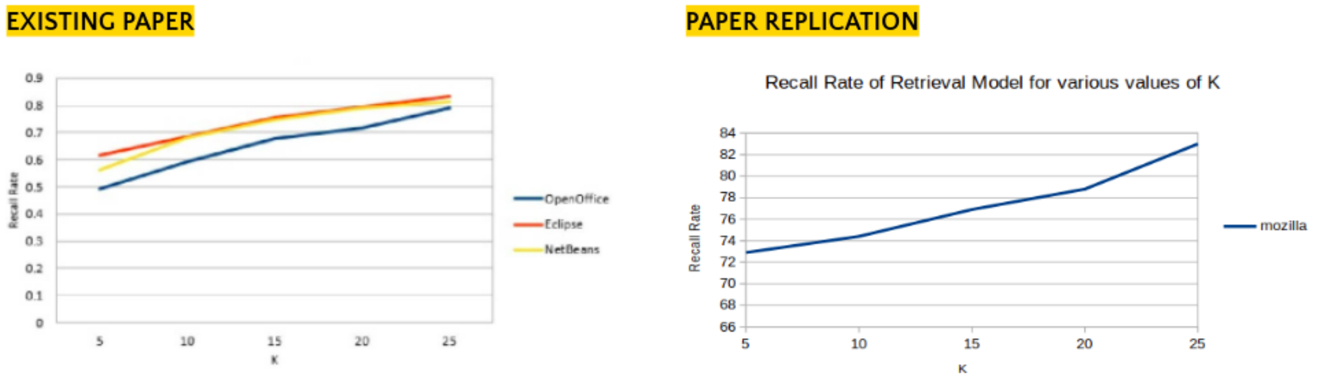


Figure 9: Recall Rate for various values of K

- **GroupBy Plots** Figure 13 and 14 displays the group by plots for bug severity and priority versus resolution. In these plots, instead of plotting value counts for a single categorical variable, two categorical variables are plotted to analyze their relationship. It is evident from the below figure 13 that normal bugs are the most duplicate ones. Similarly, figure 14 indicates bugs with no priority contain maximum duplicate bugs.

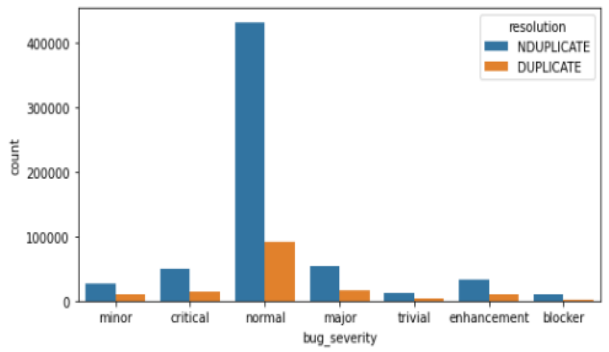


Figure 13: Bug Severity versus Resolution

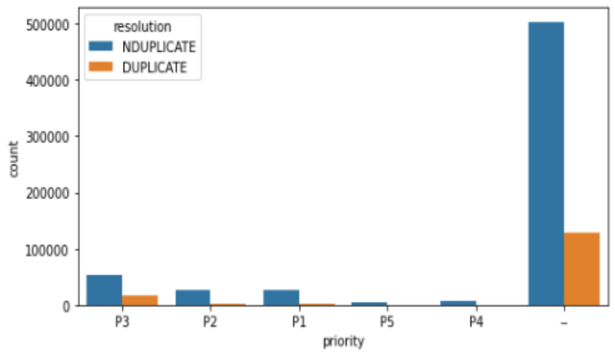


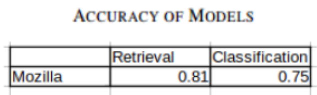
Figure 14: Priority versus Resolution

5.3 RQ3: Can some other strategies like attention mechanism or Tree-LSTM instead of Bi-LSTM help in improving?

5.3.1 Motivation.
Here, our main objective is to have a deeper insight into the variants available for their approach and explore them. LSTMs works well as seen from the paper currently, they have used Bi-LSTMs. We were motivated to use attention-based LSTMs or Tree-Based LSTM's to see their effect on the performance of the model. Chen et al. [7] employed Tree LSTMs for enhancing natural language descriptions. At the same time, we would be doing hyperparameter tuning to get the optimized set of parameters. This, in turn, yields the best-performing model. [12] highlights the importance of hyperparameter tuning to improve the results once an approach is selected.

5.3.2 Approach.
Here, to include attention-based LSTM in model architecture, we modify the baseline model architecture to include an attention layer. So the modified architecture includes CNN fused with Attention Based LSTM to perform duplicate bug detection and retrieval. The remaining approach is kept unaltered and structured, short and long descriptions encodings are computed using Single-layer RNN, CNN, and Attention Based LSTM respectively. We train the model for about 30 epochs. The results are shown in the next section.

5.3.3 Results.



Accuracy of Models

	Retrieval	Classification
Mozilla	0.81	0.75

Figure 15: Accuracy of Proposed Model

- **Accuracy Score and Loss Curve** Figure 15 shows the recall and accuracy score computed for the proposed model architecture. The recall score improved but with a trade-off in accuracy score. We believe this reduction inaccuracy is because model complexity increased. Such a complex model is not trained with sufficient training data because of a lack of computation resources. Figure 16 shows the loss versus epoch graph for the training of the proposed model. We didn't see much reduction in loss again due to training on a smaller subset of data.

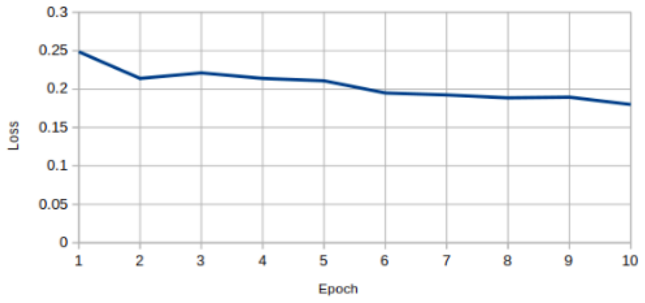


Figure 16: Loss versus Epoch curve of Proposed Model

5.4 RQ4: How effective is the approach in a cross-project setting?

5.4.1 Motivation.
In some situations, when working on a project we may not have sufficient data. For scenarios like these, it is recommended to utilize the labeled data from another project of the same domain as the training data (i.e., a cross-project setting). And the current lesser data is used as test data to evaluate the performance of the trained model. He et al. [15] indicates that when a project has lesser labeled data, similar data can be used as the training data to train the model for detection.

5.4.2 Approach.
Here, to evaluate the effectiveness of cross-project setting, models were trained and tested interchangeably on varied datasets such as Mozilla, Open Office, NetBeans, and Eclipse. For instance, a model trained on Eclipse can then be tested on Mozilla, Open Office, and Net Beans, and vice-versa. With such combinations, an evaluation metric F-score was chosen to report the results. We choose F-score as the evaluation metrics instead of accuracy because, in the case of imbalanced classification, accuracy is not a good metric as also mentioned by Sokolova et al. [34].

5.4.3 Results.
Similar results as that in [15] were obtained concluding that cross-project setting is workable. We tested this for Mozilla and Eclipse. We trained the model on the Eclipse dataset and tested the model on Mozilla which gave an F-score of 0.57. Additionally, we trained the model on the Mozilla dataset and tested the model on Eclipse which gave an F-score of 0.61. So, it can be concluded that it is doable. One thing to note here is that utmost care must be taken when deciding on the project. Furthermore, we could test projects from a different domain to show that this approach fails in such situations.

6 THREATS TO VALIDITY AND LIMITATIONS

Although we have tried to give a novel perspective to the problem statement in the best possible manner. But it has various comebacks. In this section, we talk about the drawback and limitations of our work. We discuss both threats to internal validity as well as external validity.

6.1 Threats to Internal Validity

Errors in our experimental process come in the category of internal validity.

6.1.1 Tree LSTM.
[35] has proved Tree LSTMs exploit syntactic relation between words. And can perform better than traditional LSTM structures. If we switch to Tree LSTM we can represent descriptions of bugs with information between the words. More meaningful relations would lead to better detection of redundant bugs. But there is a tradeoff, with better results comes increased complexity. And due to time and resource limitations, and the complex nature of implementing a Tree LSTM was beyond the scope of the project.

6.1.2 Dataset Dependency.
Furthermore, our results are dependent on the type of dataset. Any sort of error or bias in datasets can lead to bias or wrongful results.

6.2 Threats to External Validity

This threat to the generalizability of our experiment.

6.2.1 Reduced Data.
Any natural-language research requires heavy computational power. And the need for computational power increases with the data size. To efficiently train our model with limited resources, we had to reduce our data size. Originally, the dataset consisted of 5.7 million rows, which was reduced to 100K rows employing the publicly available GPUs provided by Google Colab and Kaggle. Due to training on limited data, we doubt our results may not generalize well with other datasets.

6.2.2 Cross-Project Prediction.
Our work employs a cross-project setting. In a cross-project prediction, another dataset is used for training instead of the focussed dataset because of shortage of labelled data. And the primary dataset is used for testing the model. Although [15] successfully shows appreciable results with a similar approach. But there is a scope for testing this hypothesis for our project. The same could have been done by training varied datasets under similar conditions. And a thorough analysis of the results of the predictions [21] talks about testing this hypothesis with ten different sets of experiments.

6.3 Limitations

6.3.1 Hyperparameter tuning.
Deep learning models exhibit a sensitive nature to hyperparameter tuning. And the sensitivity is directly proportional to the size of the dataset. Hence, for the same network tuned parameters could have given better results in terms of recall. But hyperparameter tuning is a time and resource-consuming methodology. One needs to set a possible range of parameters considering the lower and upper bound. Then the model is trained repeatedly trained on all

combinations of the given range of parameters. To find the best parameters, each training needs to be analyzed thoroughly. Hence, hyper-parameter tuning could have given better results.

6.3.2 Lack of Computational Resources.

It would have been great if we could have obtained access to GPU as there was a limit on GPU power available on Open Source tools like Google Collab and Kaggle.

7 CONCLUSION

In this paper, we propose to replicate the novel approach of using Siamese networks as mentioned in [10]. At first, we prepare the pair datasets for Mozilla because, unlike Eclipse, Open Office, NetBeans, [22] did not provide the pairs dataset for Mozilla. So, we created groups, formed duplicate and non-duplicate pairs to make the pairs dataset. This process itself involved some preprocessing. At the later stage, we first replicated the Siamese network - a combination of CNN and LSTM networks to test on Mozilla Dataset. We calculated the evaluation metrics and investigate the results with Eclipse, Open Office, and NetBeans. Each graph and the metric was replicated to check whether it generalizes well on a different dataset. It was observed that the original technique did generalize well on the Mozilla dataset. In summary, our conclusion is as follows:

- Compared baseline model results on the different datasets (Mozilla).
- Calculate Recall, Accuracy, loss curves, recall rate for different k values for the experiment.
- A comparative study by modifying the model architecture to include attention-based modeling is tested on the Mozilla dataset to make a comparison with the baseline model to which is better.
- We did a cross-project setting experiment. And interpreted that it is possible. The only thing to take care of is that project selection should be done appropriately.

For future work, we plan on handling our limitations. We will be working on Tree-LSTM's and Hyperparamteter tuning CNN network parameters. Additionally, we will cluster similar bugs and validate whether the approach suits our use case. We aim to do more experiments on varied datasets to evaluate our approach for other open-source and industry-based projects.

REFERENCES

[1] *Alipour Anahita, Hindle Abram, Stroulia Eleni.* A contextual approach towards more accurate duplicate bug report detection // 2013 10th Working Conference on Mining Software Repositories (MSR). 2013. 183–192.

[2] *Bahdanau Dzmitry, Cho Kyunghyun, Bengio Yoshua.* Neural machine translation by jointly learning to align and translate // arXiv preprint arXiv:1409.0473. 2014.

[3] *Bengio Yoshua, Simard Patrice, Frasconi Paolo.* Learning long-term dependencies with gradient descent is difficult // IEEE transactions on neural networks. 1994. 5, 2. 157–166.

[4] *Budhiraja Amar, Dutta Kartik, Shrivastava Manish, Reddy Raghu.* Towards word embeddings for improved duplicate bug report retrieval in software repositories // Proceedings of the 2018 ACM SIGIR International Conference on theory of information retrieval. 2018. 167–170.

[5] *Budhiraja Amar, Reddy Raghu, Shrivastava Manish.* LWE: LDA Refined Word Embeddings for Duplicate Bug Report Detection // Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings. New York, NY, USA: Association for Computing Machinery, 2018. 165–166. (ICSE '18).

[6] *Carneiro Tiago, Da Nóbrega Raul Victor Medeiros, Nepomuceno Thiago, Bian Gui-Bin, De Albuquerque Victor Hugo C, Reboucas Filho Pedro Pedrosa.* Performance analysis of google colaboratory as a tool for accelerating deep learning applications // IEEE Access. 2018. 6. 61677–61685.

[7] *Chen Qian, Zhu Xiaodan, Ling Zhenhua, Wei Si, Jiang Hui.* Enhancing and combining sequential and tree lstm for natural language inference // arXiv preprint arXiv:1609.06038. 2016.

[8] *Dash Manoranjan, Liu Huan.* Feature selection for classification // Intelligent data analysis. 1997. 1, 1-4. 131–156.

[9] *Deng Jianfeng, Cheng Lianglun, Wang Zhuowei.* Attention-based BiLSTM fused CNN with gating mechanism model for Chinese long text classification // Computer Speech & Language. 2021. 68. 101182.

[10] *Deshmukh Jayati, Annervaz KM, Podder Sanjay, Sengupta Shubhashis, Dubash Neville.* Towards accurate duplicate bug retrieval using deep learning techniques // 2017 IEEE International conference on software maintenance and evolution (ICSME). 2017. 115–124.

[11] *Ebrahimi Koopaei Neda.* Machine Learning And Deep Learning Based Approaches For Detecting Duplicate Bug Reports With Stack Traces. 2019.

[12] *Feurer Matthias, Hutter Frank.* Hyperparameter optimization // Automated machine learning. 2019. 3–33.

[13] *Goyal Anjali, Sardana Neetu.* Machine learning or information retrieval techniques for bug triaging: Which is better? // e-Informatica Software Engineering Journal. 2017. 11, 1.

[14] *Hanam Quinn, Brito Fernando S de M, Mesbah Ali.* Discovering bug patterns in JavaScript // Proceedings of the 2016 24th ACM SIGSOFT international symposium on foundations of software engineering. 2016. 144–156.

[15] *He Jianjun, Xu Ling, Yan Meng, Xia Xin, Lei Yan.* Duplicate bug report detection using dual-channel convolutional neural networks // Proceedings of the 28th International Conference on Program Comprehension. 2020. 117–127.

[16] *Heimerl Florian, Lohmann Steffen, Lange Simon, Ertl Thomas.* Word cloud explorer: Text analytics based on word clouds // 2014 47th Hawaii International Conference on System Sciences. 2014. 1833–1842.

[17] *Hindle Abram, Alipour Anahita, Stroulia Eleni.* A contextual approach towards more accurate duplicate bug report detection and ranking // Empirical Software Engineering. 2016. 21, 2. 368–410.

[18] *Kim Sunghun, Zimmermann Thomas, Nagappan Nachiappan.* Crash graphs: An aggregated view of multiple crashes to improve crash triage // 2011 IEEE/IFIP 41st International Conference on Dependable Systems & Networks (DSN). 2011. 486–493.

[19] *Kucuk Berfin, Tuzun Eray.* Characterizing Duplicate Bugs: An Empirical Analysis // 2021 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER). 2021. 661–668.

[20] *Kukkar Ashima, Mohana Rajni, Kumar Yugal, Nayyar Anand, Bilal Muhammad, Kwak Kyung-Sup.* Duplicate bug report detection and classification system based on deep learning technique // IEEE Access. 2020. 8. 200749–200763.

[21] *Kumar .* Transfer Learning for Cross-Project Change-Proneness Prediction in Object-Oriented Software Systems: A Feasibility Analysis // ACM SIGSOFT Software Engineering Notes. 07 2017. 42. 1–11.

[22] *Lazar Alina, Ritchey Sarah, Sharif Bonita.* Generating duplicate bug datasets // Proceedings of the 11th working conference on mining software repositories. 2014. 392–395.

[23] *Lazar Alina, Ritchey Sarah, Sharif Bonita.* Improving the Accuracy of Duplicate Bug Report Detection Using Textual Similarity Measures // Proceedings of the 11th Working Conference on Mining Software Repositories. New York, NY, USA: Association for Computing Machinery, 2014. 308–311. (MSR 2014).

[24] *Mikolov Tomas, Chen Kai, Corrado Greg, Dean Jeffrey.* Efficient estimation of word representations in vector space // arXiv preprint arXiv:1301.3781. 2013.

[25] *Neysiani Behzad Soleimani, Babamir Seyed Morteza.* Automatic duplicate bug report detection using information retrieval-based versus machine learning-based approaches // 2020 6th International Conference on Web Research (ICWR). 2020. 288–293.

[26] *Osman Haidar, Ghafari Mohammad, Nierstrasz Oscar.* Automatic feature selection by regularization to improve bug prediction accuracy // 2017 IEEE Workshop on Machine Learning Techniques for Software Quality Evaluation (MaLTeSQuE). 2017. 27–32.

[27] *Pennington Jeffrey, Socher Richard, Manning Christopher D.* GloVe: Global Vectors for Word Representation // Empirical Methods in Natural Language Processing (EMNLP). 2014. 1532–1543.

[28] *Polignano Marco, Basile Pierpaolo, Gemmis Marco de, Semeraro Giovanni.* A comparison of word-embeddings in emotion detection from text using bilstm, cnn and self-attention // Adjunct Publication of the 27th Conference on User Modeling, Adaptation and Personalization. 2019. 63–68.

[29] *Rocha Henrique, Oliveira Guilherme de, Valente Marco Tulio, Marques-Neto Humberto.* Characterizing Bug Workflows in Mozilla Firefox // Proceedings of the 30th Brazilian Symposium on Software Engineering. New York, NY, USA: Association for Computing Machinery, 2016. 43–52. (SBES '16).

[30] *Rocha Thiago Marques, Carvalho André Luiz Da Costa.* SiameseQAT: A Semantic Context-Based Duplicate Bug Report Detection Using Replicated Cluster Information // IEEE Access. 2021. 9. 44610–44630.

[31] Duplicate bug report detection through machine learning techniques. // . 2018.

[32] *Runeson Per, Alexandersson Magnus, Nyholm Oskar.* Detection of duplicate defect reports using natural language processing // 29th International Conference on Software Engineering (ICSE'07). 2007. 499–510.

[33] *Sabor Korosh Koochekian, Hamou-Lhadj Abdelwahab, Larsson Alf.* Durfex: a feature extraction technique for efficient detection of duplicate bug reports // 2017 IEEE international conference on software quality, reliability and security (QRS). 2017. 240–250.

[34] *Sokolova Marina, Japkowicz Nathalie, Szpakowicz Stan.* Beyond accuracy, F-score and ROC: a family of discriminant measures for performance evaluation // Australasian joint conference on artificial intelligence. 2006. 1015–1021.

[35] *Tran Nam-Khanh, Cheng Weiwei.* Multiplicative tree-structured long short-term memory networks for semantic representations // Proceedings of the Seventh Joint Conference on Lexical and Computational Semantics. 2018. 276–286.

[36] *Tukey John W, others .* Exploratory data analysis. 2. 1977.

[37] *Wang Liting, Zhang Li, Jiang Jing.* Detecting Duplicate Questions in Stack Overflow via Deep Learning Approaches // 2019 26th Asia-Pacific Software Engineering Conference (APSEC). 2019. 506–513.

[38] *Xie Jun, Chen Bo, Gu Xinglong, Liang Fengmei, Xu Xinying.* Self-attention-based BiLSTM model for short text fine-grained sentiment classification // IEEE Access. 2019. 7. 180558–180570.

[39] *Xie Qi, Wen Zhiyuan, Zhu Jieming, Gao Cuiyun, Zheng Zibin.* Detecting duplicate bug reports with convolutional neural networks // 2018 25th Asia-Pacific Software Engineering Conference (APSEC). 2018. 416–425.

[40] *Xu Kelvin, Ba Jimmy, Kiros Ryan, Cho Kyunghyun, Courville Aaron, Salakhudinov Ruslan, Zemel Rich, Bengio Yoshua.* Show, attend and tell: Neural image caption generation with visual attention // International conference on machine learning. 2015. 2048–2057.

[41] *Zhang Ye, Wallace Byron.* A sensitivity analysis of (and practitioners' guide to) convolutional neural networks for sentence classification // arXiv preprint arXiv:1510.03820. 2015.