Int J Syst Assur Eng Manag (July 2024) 15(7):3538–3550
https://doi.org/10.1007/s13198-024-02361-4

ORIGINAL ARTICLE

PR-DupliChecker: detecting duplicate pull requests in Fork-based workflows

Montassar Ben Messaoud[1] · Rania Ben Chekaya[2] · Mohamed Wiem Mkaouer[3] · Ilyes Jenhani[4] · Wajdi Aljedaani[5]

Received: 13 February 2023 / Revised: 6 November 2023 / Accepted: 27 April 2024 / Published online: 19 June 2024

Abstract Pull requests (PR) are a fundamental aspect of collaborative software development, allowing developers to propose changes to a codebase hosted on platforms like GitHub. They serve as a mechanism for peer review, enabling team members to assess the proposed changes before merging them into the main code repository. Duplicate pull requests occur when multiple contributors submit similar or identical proposed changes to a code repository. Such duplicate pull requests can be problematic because they create redundancy, waste developers' time, and complicate the review process. In this paper, we propose an approach which is based on a pre-trained language model, namely BERT (Bidirectional Encoder Representations from Transformers) to automatically detect duplicate PRs in GitHub repositories. A dataset of 3328 labeled PRs collected from 26 GitHub repositories is built. This data is then fed to a BERT model in order to get the embeddings which represent the contextual relationships between the words used in pairs of pull requests. Then, the BERT's classification outputs are fed to a Multilayer Perceptron (MLP) classifier which represents our final duplicate pull requests detector. Experiments have shown that BERT provided good performance and achieved an accuracy of 92% with MLP classifier. Results have proven that BERT's word representation features achieved an increase of 13% (resp., 17 and 23%) compared to Siamese-BERT model (resp., DC-CNN and Word2Vec) in term of accuracy.

Keywords Duplicate pull requests · Deep neural networks · Pre-trained neural language model · BERT

✉ Mohamed Wiem Mkaouer
mmkaouer@umich.edu

Montassar Ben Messaoud
montassar.benmessaoud@tbs.u-tunis.tn

Rania Ben Chekaya
ben.chekaya.rania@gmail.com

Ilyes Jenhani
ijenhani@pmu.edu.sa

Wajdi Aljedaani
wajdialjedaani@my.unt.edu

1 LARODEC, Institut Supérieur de Gestion de Tunis, Université de Tunis, Tunis, Tunisia

2 Institut Supérieur de Gestion de Sousse, Université de Sousse, Sousse, Tunisia

3 University of Michigan-Flint, Flint, MI, USA

4 College of Computer Engineering and Science, Prince Mohammad Bin Fahd University, Khobar, Kingdom of Saudi Arabia

5 University of North Texas, Denton, TX, USA

1 Introduction

Collaborative software development through pull requests streamlines the contribution process by allowing developers to propose changes, share their work, and engage in peer reviews effectively. Pull requests serve as a platform for collaborative discussions, enabling team members to provide feedback, suggest improvements, and ensure code quality before integration. This transparent and iterative approach enhances collaboration, code reliability, and the overall success of software projects. Nowadays, an increasing number of open source software projects on GitHub are using the pull-based development model, like Rails repository[1] which has received more than 28,906 Pull-Requests (PR). Moreover, this model supported by distributed version control

[1] https://github.com/rails/rails/pulls.

system (e.g., git) (Yu et al. 2016), in which any contributor can clone a repository for local changes to fix some bugs, add new functional features or make improvements (in term of performance, usability, reliability, etc.) (Eyal Salman et al. 2022). Then, submit PRs for community discussion before being merged into the original repository (Yu et al. 2018).

Despite all these advantages, the PR process has some drawbacks related to the uncoordinated and distributed process, especially for popular projects which attract many contributors and receive a lot of PRs daily (Li et al. 2017). Developers could work on the same problem and submit duplicate PRs (Eyal Salman et al. 2022; Li et al. 2017, 2021; Wang et al. 2019). Having duplicate pull requests can lead to confusion and inefficiencies in the software development process. It creates redundancy, making it difficult for the team to identify the best solution among similar or identical changes. Reviewers can end up wasting time evaluating duplicate submissions, preventing the integration of valuable code updates. Additionally, duplicate pull requests can cause fragmented discussions and feedback, making it harder to maintain clear communication within the development team. In summary, the existence of duplicate pull requests has negative side effects on the project's development, as it increases the developers' unnecessary workload. It can also cause merge conflicts when two duplicate pull requests are accidentally pushed into production. Merge conflicts can become a sudden burden that would trigger more developers to be involved in this untangling process, before realizing that both pushed code changes are similar. Additionally, duplicate pull requests increase the number of delayed and overdue fixes that are very much needed to maintain the software.

To tackle this problem, we present an automated approach to detect duplicate PRs using textual information. Knowing that, a typical PR includes textual information (e.g., title and description) and non-textual information (e.g., patch content, changed files list, location of code changes, reference to issue tracker and time). Note that in our study, we focus only on textual information.

The studies related to the research problem addressed in this paper take two complementary directions. The first one focused on the recommendation and ranking approaches using word embedding techniques such as TF-IDF (Li et al. 2017, 2021). While the second direction concentrated on automatically extracting features from PRs using basic machine learning algorithms (Ren et al. 2019; Wang et al. 2019). None of these studies have dealt with duplicate PRs using deep-learning language models.

Therefore, we propose to use the Bidirectional Encoder Representations from Transformers (BERT) (Devlin et al. 2019), since it shows highly competitive performance across a wide range of text classification tasks.

We compared our approach to other models, namely, Siamese-BERT (Ciborowska and Damevski 2021; Feifei et al. 2020), Dual-Channel Convolutional Neural Network (DC-CNN) (He et al. 2020) and Word2Vec (Mikolov et al. 2013).

To evaluate our approach, we built a dataset of 3,328 PRs that we have collected from a public available dataset.[2] Experimental results showed that BERT model provided the best performance and achieved an accuracy of 92% with Multilayer Perceptron (MLP) classifier, compared to Siamese-BERT model (resp., DC-CNN and Word2Vec) which achieved an accuracy of 79% (resp., 75 and 69%).

In this paper, we aim to answer the following research questions:

RQ1: Which classifier to use for fine-tuning BERT to better detect duplicate PRs?

In this RQ, we aim to show how BERT is performing for the task of duplicate PRs detection using textual information (i.e., title and description). Then we aim to determine which classifier to use in fine-tuning BERT in term of accuracy, precision, recall and F1-score.

RQ2: How does BERT-MLP perform when we trained PR titles only, PR descriptions only and when we combined PR titles and descriptions?

In this RQ, we aim to show whether we can achieve a good PR detection rate by looking at PR titles only and whether PR descriptions can contain relevant information to improve the detection rate.

RQ3: How does BERT-MLP model compare with the other models, Siamese-BERT, DC-CNN and Word2Vec?

In this RQ, we aim to compare our model with other models, namely, Siamese-BERT, DC-CNN and Word2Vec.

To support the replication and extension of our work, we provide the community with the artifacts we use for our work, in this replication package link: https://github.com/smilevo/DuplicatePullRequests.

The rest of the paper is organized as follows. Section 2 reports the related works and positions our work with respect to the existing works. Section 3 provides basic background about the pull request mechanism and the BERT model. Section 4 describes the different phases of the proposed approach. Experimental results are presented and discussed in Sect. 5. Finally, Sect. 6 concludes the paper and provides some future works.

[2] https://github.com/whystar/MSR2018-DupPR/blob/master/dup_prs.md.

Table 1 The most recent studies on duplicate PRs

Study	Year	Featues	Methodology	Repositories
Retrieval approaches				
Li et al. (2017)	2017	Textual information	TF-IDF, Cosine similarity	3 repositories
Li et al. (2021)	2021	Textual and non-textual information	TF-IDF, Cosine similarity	26 repositories
Classification approaches				
Ren et al. (2019)	2019	Textual and non-textual information	AdaBoost	26 repositories
Wang et al. (2019)	2019	Textual and non-textual information	AdaBoost	26 repositories
Eyal Salman et al. (2022)	2022	Textual information	K-Means, agglomeration hierarchical clustering	20 repositories
Our work	2022	Textual information	BERT	26 repositories

2 Related works

In the literature, many approaches have been proposed for automatic detection of duplicate PRs. These studies are classified into two categories (Eyal Salman et al. 2022):

- *Pull-Request retrieval*: For a given new PR, retrieve a list of ranked PRs.
- *Pull-Request classification*: For a given new PR, assign a label whether it's duplicate or non-duplicate.

In the first category, Li et al. (2017) proposed a method to detect duplicate PRs. For a given PR, the authors compare the textual information (i.e., title and description) between the new PR and the historical PRs, then return a candidate list of the most similar ones using TF-IDF with cosine similarity. Their approach has been tested on three popular projects hosted on GitHub, namely Rails, Elasticsearch, and Angular.JS. The results showed that (55.3–71.0%) duplicates are found when both title and description similarities are used in a combination.

To improve their previous work Li et al. (2021) proposed an approach to automatically detect duplicate PRs combined textual information (i.e., title and description) and non-textual information (i.e., changed files and changed lines of code) to return a candidate list of historical PRs that are most similar to the new-arriving PR, using TF-IDF with cosine similarity. Their evaluation showed that the combined similarities achieved the best performance (83%) instead of using only textual similarity (54.8%) and only change similarity (78.2%) in terms of recall-rate@k metric.

In the second category, Ren et al. (2019) proposed an approach to improve Li et al.'s (2017) work to automatically detect duplicate PRs. Taking into account other factors than the textual information that can be extracted from both the title and description of PRs. These factors have five dimensions (namely, change description, patch content, changed files list, changed lines of code, and reference to issue tracker) with nine features. They assign a label (duplicate or not duplicate) to a new coming PR instead of retrieving a ranked list of PRs. To evaluate their approach they used machine learning algorithm (AdaBoost). The results showed that the work of Ren et al. achieves better results than the work of Li et al. (16–21%) in terms of recall.

Similarly, Wang et al. (2019) proposed an approach to improve the work of Ren et al. (2019) by considering the creation time of PRs in combination with the nine features identified by Ren et al. They assume that two PRs are most likely duplicate when their created times are close to each other. To train the values of these features they used AdaBoost algorithm. Results showed that Wang et al. improve the performance of Ren et al.'s work by 14.36 and 11.93% in terms of F1-score metric.

Eyal Salman et al. (2022) proposed an automatic approach to group similar PRs together into clusters, each cluster is assigned to the same reviewer or the same reviewing team, using machine learning algorithms (K-Means clustering and agglomeration hierarchical clustering). To evaluate their approach, they have applied it to 20 popular repositories from a public dataset. The experimental results showed that the proposed approach is efficient in identifying relevant clusters. K-Means algorithm achieves 94% (resp., 91%), while agglomeration hierarchical clustering performs 93% (resp., 98%) in terms of precision (resp., recall) metrics.

The common point of the previous works is that they are all limited to detecting duplicate PRs using traditional NLP techniques or standard machine learning algorithms which led to unsatisfied detection accuracies in most of them.

Therefore, what distinguishes our work from existing literature on this problem, is that we aim to perform a deeper analysis using state-of-the-art contextual pre-trained language models for text classification, namely BERT. In this way, we can provide a broad understanding of the intrinsic semantic structures of the two PRs and better interpretability of their corresponding word embeddings through the joint reasoning between their different elements.

Table 1 summarizes the related works.

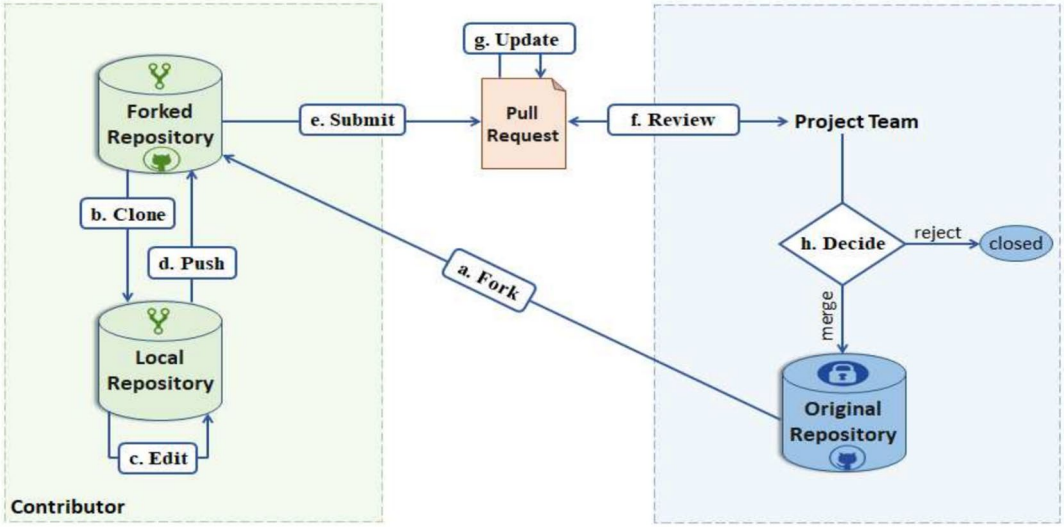


Fig. 1 An overview of pull-request mechanism in GitHub (Eyal Salman et al. 2022; Li et al. 2020)

3 Background

In this section, we provide an overview of PR, we start by presenting the mechanism of PR and defining the concept of duplicate PPs and transfer learning, then we introduce the pre-trained BERT language model in the context of duplicate PRs detection.

3.1 Pull-request mechanism

PR is a distributed development model (Wang et al. 2019), involving an increasing number of developers to contribute on code-sharing platforms such as GitHub, which is the world's largest collection of open source software (Li et al. 2021). In addition, PR allows other contributors to learn about, discuss and review the potential changes before whether these changes are accepted or rejected in GitHub repositories (Eyal Salman et al. 2022).

As shown in Fig. 1 a typical PR process on GitHub involves the following steps. First of all, contributors can find an interesting project by following some well-known developers (Wang et al. 2019). Then, he can fork the original project on his GitHub account. On the basis of forking repository, contributors can safely add their changes, such as implementing a new feature or fixing bugs without disturbing the original repository branch. When the changes are ready, he pushes the local changes to the forked repository and submits the modified code from the forked repository to its source by PR to review and discuss them with the team members of the project. In addition to the commits, the contributor must provide a title and description to specify the purpose of his PR. Finally, after several rounds of rigorous evaluations, the project team leader considers all the opinions of the reviewers and makes the decision to merge or reject (close) the PR based on its eventual quality.

3.2 Duplicate pull request

PRs that aim to achieve the same goal (e.g., fixing the same bug or suggesting equivalent functional features) in the same GitHub repository are called duplicate PRs (Li et al. 2017). In addition, duplicate PRs may be submitted to review in parallel time by different reviewers, this increases review time and effort.

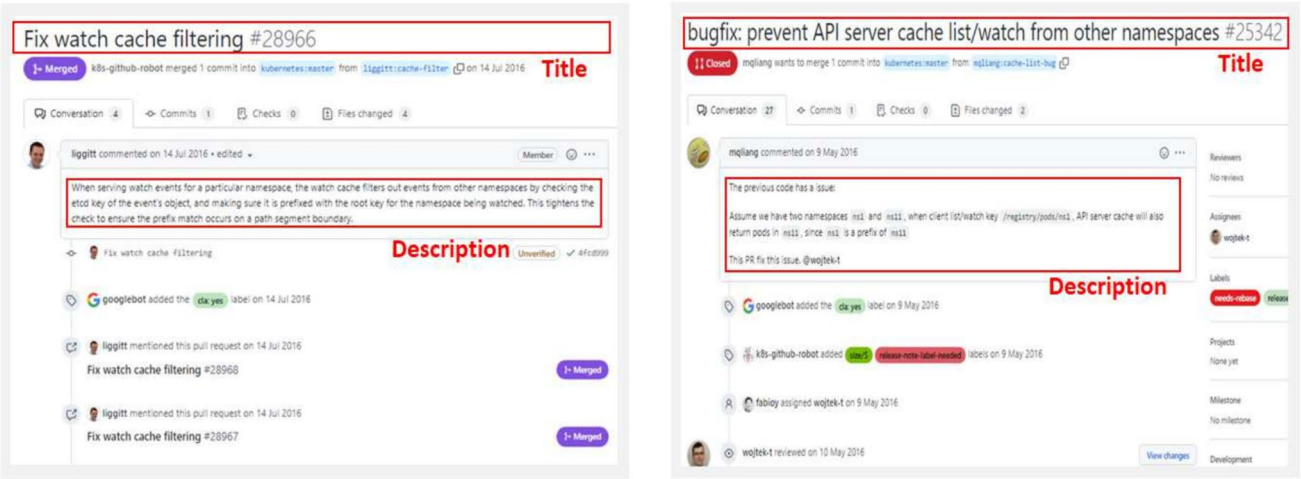
Figure 2 shows a real example of two duplicate PRs (i.e., a[3] and b[4]) from kubernetes repository,[5] both of them aiming to resolve the same problem by different contributors. Therefore, they are duplicates.

3.3 Transfer learning for natural language processing

In machine learning, reusing a pretrained model as a starting point for another model on a new task is known as transfer learning (TL). Therefore, the concept of TL is very important in the field of deep learning, especially if we are dealing with a model that we cannot train it from scratch.

In TL, machines use knowledge learned from previous tasks to increase the prediction of a new one, reduce training time, and improve neural network performance. Moreover, TL have proven to achieve the state of the art in several NLP tasks (e.g., Text classification, Question answering, Machine

[3] https://github.com/kubernetes/kubernetes/pull/28966.
[4] https://github.com/kubernetes/kubernetes/pull/25342.
[5] https://github.com/kubernetes/kubernetes/.



(a) Pull-Request 28966 (b) Pull-Request 25342

Fig. 2 A pair of duplicate pull-requests of kubernetes project in GitHub

translation, etc.). That is what made TL a successful solution to many supervised NLP tasks (Ghadhab et al. 2021).

3.4 Bidirectional encoder representations from transformers

BERT, or Bidirectional Encoder Representations from Transformers (Devlin et al. 2019), is a deep learning model that has given state-of-the-art results in several NLP problems. Besides, BERT offers the golden example of the most successful transformers, presented by Vaswani et al. (2017b) to introduce the novel architecture of transformers which is based on two major components: encoder to read the input text and decoder to produce a prediction task.

Unlike to the standard models, which read the input text from left-to-right or right-to-left, BERT uses Bidirectional Encoder Representation to learn information from both the left and the right side of token context during the training process. This mechanism allows the model to learn the context of a word based on all its surroundings (i.e., left and right of the word).

There are two pre-trained versions of BERT according to the model structure scale, the first is the BERT-Base model (composed of 12 layers, 768 hidden-nodes, 12 attention heads, 110 M parameters) and the second is BERT-Large model (composed of 24 layers, 1024-hidden-nodes, 16 attention heads and 340 M). A pre-trained BERT model takes as input a sequence of words of no more than 512 tokens. The identifiers of these tokens will pass through the different encoder layers of BERT to produce the word representations as vectors of size 768 each.

To model BERT for a text classification task, we go through two main steps, namely, pre-training and fine-tuning (Devlin et al. 2019). The first one consists of embedding the inputs (e.g., single text or pair of sentences) of BERT model which allows one to transform a word into a vector through a tokenizer object. Besides, BERT uses the two special tokens [CLS] and [SEP] to identify the start and the end of a sentence ([SEP] is also used to separate a pair of sentences). In addition to another special token [PAD], which is used to represent paddings (i.e., empty tokens) to make all BERT inputs equal in size. However, BERT is built on top of three embedding inputs: Token Embedding, Position Embedding, and Segment Embedding.

Moreover, BERT has been pre-trained on Wikipedia and BooksCorpus (Zhu et al. 2015) for two unsupervised tasks: MLM (Masked Language Model), it randomly masked words in the sentence, then attempts to predict them based on the context provided by the other non-masked words (15% of the words in each sequence are replaced with a [MASK] token Devlin et al. 2019), and NSP (Next Sentence Prediction) to produce a pre-trained model. This tasks makes BERT different from other language models such as ELMo (Peters et al. 2018) and OpenAI GPT (Radford and Narasimhan 2018). For fine-tuning, the model is initialized with the pre-trained parameters, and all the parameters are fine-tuned using labeled data for specific tasks such as classification (Nugroho et al. 2021).

Here BERT uses to predict similarities between two sentences, for a given two sentences (i.e., PR1 and PR2) as inputs, the model predicts if the second sentence (i.e., PR2) logically is the subsequent sentence of the first one (i.e., PR1), by adding on top of BERT another classification layer. In this way, our approach will benefit from the rich knowledge learned by BERT during the pre-training process.

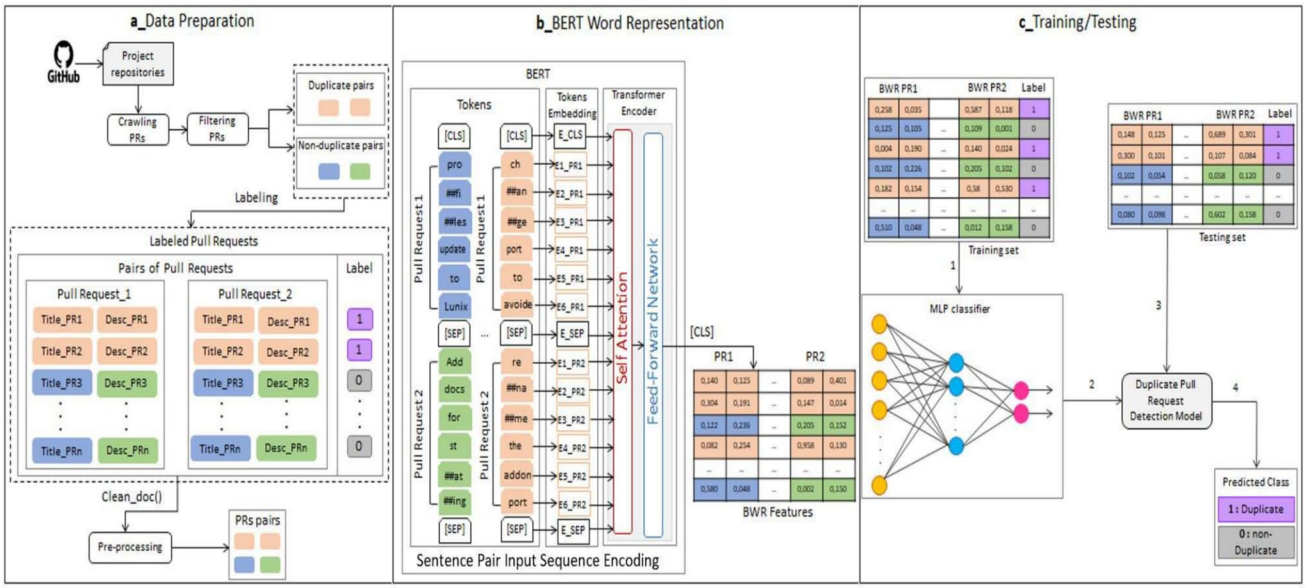


Fig. 3 Duplicate pull request detection process

4 Methodology

In this section, we present our methodology for detecting duplicate PRs. The proposed BERT model uses the idea of transfer learning to solve the problem of duplicate PRs detection. As shown in Fig. 3, our overall approach includes three main phases. First, we provide the data preparation, BERT's contextualized Word Representations (BWR), and finally, we present the training and testing phase.

4.1 Data preparation

The goal of this section is to describe our data collection and preprocessing steps.

4.1.1 Data collection

To evaluate our approach, we use a popular dataset named DupPR (Yu et al. 2018), which contains 2323 pairs of duplicate PRs collected from 26 repositories on GitHub. In DupPR, each pair has been manually verified after an automatic identification process, to ensure the quality of the dataset (Li et al. 2021).

However, this dataset only provides examples of duplicate PRs, and does not provide non-duplicate PRs which are more common in practice (Gousios et al. 2014). To solve this problem and provide a dataset in line with our approach (i.e., a dataset where PRs are labeled as duplicate and non-duplicate), some pairs of merged PRs from the same repositories are randomly sampled according to the assumption that if two PRs are merged, they are most likely non-duplicate (Ren et al. 2019; Wang et al. 2019).

Moreover, in DupPR, there are two versions of PRs (duplicate and triple pairs). For our study, we only consider the duplicated pairs. Therefore, as shown in Table 2, we ended up with 1664 pairs of duplicate PRs assigned as (1), which are the complete dataset in the DupPR and 1664 pairs of non-duplicate PRs assigned as (0), which are selected randomly from the same projects that we have in the list of DupPR. Furthermore, when we select the random PRs, we make sure that we selected merged ones, therefore, we can avoid repeating them with any other PRs.

Overall, we built a balanced dataset of 3328 labeled PRs, in which duplicate and non duplicate training and testing pairs were evenly distributed as provided in Table 2.

Besides, the corpus of our dataset contains both duplicate and non-duplicate PRs. As illustrated in Fig. 3a, a pair of PRs is a combination of two PRs, and each is represented by a title and description.

Table 2 Distribution of pull requests

Training		Testing		Total
Duplicates	Non-duplicates	Duplicates	Non-duplicates	
1331	1331	333	333	3328
2662		666		

4.1.2 Data pre-proceessing

Data cleaning and pre-processing present the most important steps to improve the performance of our model. Therefore, each PR pair undergoes through some pre-processing steps. To do so, we first remove stop words, digits, special characters (e.g., @; ! # ’ ? = [|] * + − $, etc.), spaces, emojis, emoticons, hashtags. Then, we fill the empty description with spaces because the model cannot learn empty rows.

4.2 BERT’s words representations

Figure 3b details the most important steps of BERT word representation. First of all, each pair of our dataset is fed to the tokenizer in order to split them into smaller subwords and words by using the tokenizer object, which is created with a WordPiece technique (Wu et al. 2016), (e.g., merged: ##me, ##rg, ##ed). Then, a [CLS] token is inserted at the beginning to be used later in the classification task. Since our input is a pair of PRs, therefore we need a way to inform the model where PR1 ends and PR2 begins. Therefore, a [SEP] token is used to separate them and another [SEP] token is added automatically to indicate the end of the sequence.

As shown in Fig. 3b, the structure of BERT input is as following: pair = [CLS] + [’Title_PR1’] + [’Description_PR1’] + [SEP] + [’Title_PR2’] + [’Description_PR2’] + [SEP].

To avoid memory and computation overhead, we also set the maximum length of 300 tokens for each pair (i.e., 150 tokens for each PR), and if some PR is shorter than this limit, the [PAD] token is used to represent paddings. Then, the obtained tokens will be replaced by their identifiers, which constitute the token embeddings.

Subsequently, the token embeddings will traverse all the transformer encoders (Ghadhab et al. 2021), and each one is based on two layers, namely, Self-attention layer and feed forward neural network layer.

Self-attention layer, is an attention mechanism relating to different positions of a single sequence to compute its representation (Vaswani et al. 2017a). This mechanism enables the model to pay attention to the other words in the input sequence and get a better understanding of certain words in the sequence.

In Fig. 4, we used the following PR example: “*The text was successfully updated but these errors were encountered*”. The word “*errors*” refers to the word “*text*” in the first case, but to *“successfully”* and “*encountered*” in the second case. This dependency between words is detected by the self-attention layer and considered when encoding the initial token embeddings to another contextual representation.

The contextualized embedding resulting from self-attention is then passed through a feedforward neural network to produce the BERT’s word representations (BWR). At this stage, the MLP classifier is ready for training and testing in our BWR of PR pairs.

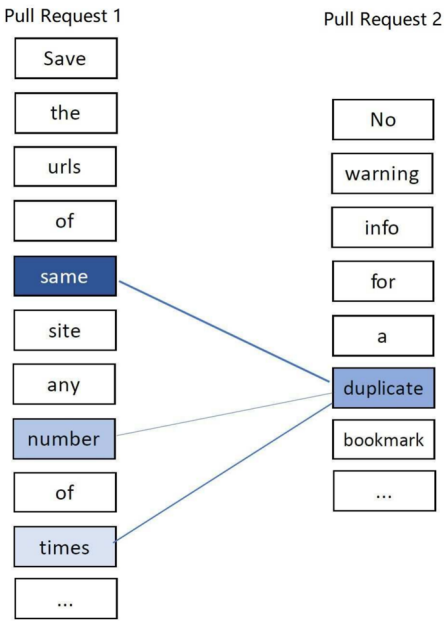


Fig. 4 Self-attention token correspondences

4.3 MLP Training and testing

This phase consists of training a multi-layer perceptron (MLP) model that will be used to detect duplicate PRs.

Therefore, in order to improve the performance of our model, we found the most appropriate hyperparameters using the Scikit-learn GridSearch() function, which has been used for the fine-tuning step.

First, we have tested different numbers of hidden layers (5 layers). In each hidden layer, we experimented with different numbers of neurons. Then, we generated 50, 100 and 150 neurons, respectively for the five hidden layers. The weighted sum of BWR inputs is first calculated, then passed through a Rectified Linear Unit (ReLU) activation function. For the optimization strategy, two solvers have been used (i.e., Sgd Robbins and Monro 1951 and Adam Kingma and Ba 2014), and tested with different learning rate values (i.e., constant and adaptive). The maximum number of iterations has been fixed to 150. Alpha hyper-parameter has been tested with values from 0.0001 to 0.05. Table 3 shows the best MLP parameters.

5 Experimental results and analysis

In this section, we evaluate our proposed approach. Firstly, we will cite the evaluation metrics. Then, we will present the

Table 3 MLP hyper-parameters

Parameter	Score
Hidden layers	5
Hidden layer sizes	(50, 100, 150)
Max iter	150
Activation	Relu
Solver	sgd
Learning rate	adaptive
Alpha	0.05

different experiments proposed to discuss and evaluate our method by suggesting research questions.

5.1 Evaluation metrics

We evaluate the performance of our approach using the following metrics:

- *Accuracy*: Accuracy is a metric that generally describes the performance of a model across all classes,the highest accuracy is always better. It is defined as:

$$Accuracy = \frac{Number\ of\ correct\ predictions}{Total\ number\ of\ predictions} \quad (1)$$

- *Precision*: Called positive predictive value. It is defined as:

$$Precision = \frac{True\ Positives}{True\ Positives + False\ Positives} \quad (2)$$

- *Recall*: Called sensitivity, probability of detection, True Positive Rate. It is defined as:

$$Recall = \frac{True\ Positives}{True\ Positives + False\ Negatives} \quad (3)$$

- *F1-score*: Combines the precision and recall metrics into a single metric. It is defined as:

$$F1\text{-}score = 2 * \frac{Precision * Recall}{Precision + Recall} \quad (4)$$

5.2 Results and analysis

In this section, We will try to answer the following research questions described in the introduction:

5.2.1 *RQ1: Which classifier to use for fine-tuning BERT to better detect duplicate PRs?*

To answer this research question, we adopted standard classification techniques available in Scikit-learn python library (Pedregosa et al. 2012), namely, Logistic Regression (LR) (resp., Decision Tree (DT), Support Vector Classification (SVC), MultiLayer Perceptron (MLP), Random Forest (RF), Naive Bayes (NB), XGBoost (XGB) and K Nearest Neighbor (KNN)) to select our base classifier.

Table 4 Precision, recall and F1-score for each classifier (Highest scores are shown in bold)

Classifier	Precision	Recall	F1-score
BERT-LR	0.89	0.89	0.89
BERT-DT	0.79	0.79	0.79
BERT-SVC	0.91	0.90	0.90
BERT-MLP	**0.92**	**0.92**	**0.92**
BERT-RF	0.87	0.87	0.87
BERT-NB	0.72	0.64	0.61
BERT-XGB	0.86	0.85	0.85
BERT-KNN	0.88	0.86	0.86

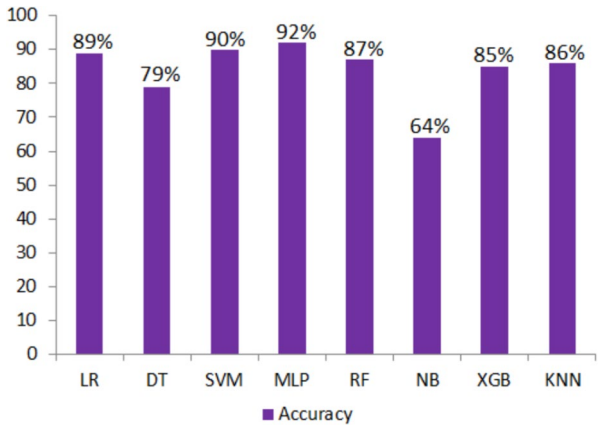


Fig. 5 Accuracy results for each classifiers

Figure 5 shows the accuracy of all classifiers when combined with BERT.

Results show that BERT-MLP achieved the highest accuracy (92%) and the lowest accuracy achieved with BERT-NB (79%). For the other classifiers, BERT-LR (resp., BERT-DT, BERT-SVC, BERT-RF, BERT-XGB and BERT-KNN) results are 89% (resp., 79, 90, 87, 85 and 86%). We also provide precision, recall, and F1 score for each BERT-classifier combination, as illustrated in Table 4, our main classifier (i.e., MLP) achieved the best results (0.92) in term of precision, recall and F1-score even though SVM and LR are also doing well.

For the non duplicate class, our approach reached 0.91 (resp., 0.92 twice) in terms of precision (resp., recall and F1-Score). While for the duplicate class, our model achieved 0.92 (resp., 0.91 and 0.92) in terms of precision (resp., recall and F1-Score), as shown in Table 5.

Table 5 Precision, recall and f1-score for each class (Highest scores are shown in bold)

Classifier	Metrics					
	Precision		Recall		F1-score	
	Non-duplicate	Duplicate	Non-duplicate	Duplicate	Non-duplicate	Duplicate
BERT-LR	0.86	0.93	0.93	0.85	0.90	0.89
BERT-DT	0.80	0.78	0.77	0.80	0.78	0.79
BERT-SVC	0.88	0.94	0.94	0.87	0.91	0.90
BERT-MLP	**0.91**	0.92	0.92	0.91	**0.92**	**0.92**
BERT-RF	0.87	0.87	0.87	0.86	0.87	0.87
BERT-NB	0.85	0.59	0.35	**0.94**	0.49	0.72
BERT-XGB	0.88	0.83	0.82	0.89	0.85	0.86
BERT-KNN	0.80	**0.96**	**0.97**	0.76	0.88	0.85

Table 6 BERT-MLP performance when using (a) only titles, (b) only descriptions and (c) titles and description combined (Highest scores are shown in bold)

	Accuracy	Precision	Recall	F1-score
(a)	0.83	0.84	0.83	0.83
(b)	0.85	0.85	0.85	0.85
(c)	**0.92**	**0.92**	**0.92**	**0.92**

Based on these results, we found that MLP is the best performing classifier for PR classification. Therefore, in the next research questions, we will adopt the BERT-MLP as a primary model for our approach.

5.2.2 *RQ2: How does BERT-MLP perform when we trained PR titles only, PR descriptions only and when we combined PR titles and descriptions?*

To answer this research question, we evaluated our model (i.e., BERT-MLP) in three different cases, in the first case (a) we used only PR titles, in the second case (b) we used only PR descriptions, and in the third case (c) we combined titles and descriptions together. The evaluation showed that case (c) (i.e., using combined titles and descriptions) achieved the highest accuracy (92%) compared to 83% (resp., 85%) when we used only titles (resp., using only descriptions). The evaluation has also proven that (c) achieved the best results in terms of precision (resp., recall and F1-score) compared to (a) and (b) cases, as shown in Table 6. Based on these results, we can prove that the combination of titles and descriptions could achieve better result than only using titles or descriptions to train our classification model.

Table 7 BERT-MLP vs Siamese-BERT-MLP, DC-CNN and Word-2Vec-MLP (Highest scores are shown in bold)

Metrics	Accuracy	Precision	Recall	F1-score
BERT-MLP	**0.92**	**0.92**	**0.92**	**0.92**
Siamese-BERT-MLP	0.79	0.80	0.79	0.79
DC-CNN	0.75	0.75	0.75	0.75
Word2Vec-MLP	0.69	0.69	0.69	0.68

5.2.3 *RQ3: How does BERT-MLP model compare to the other models, Siamese-BERT, DC-CNN and Word2Vec?*

To demonstrate the effectiveness of our approach, we compare our model with three other models, namely, Siamese-BERT (Ciborowska and Damevski 2021; Feifei et al. 2020), DC-CNN (Dual-Channel Convolutional Neural Network) (He et al. 2020) and Word2Vec (Mikolov et al. 2013). For Siamese-BERT and Word2Vec we selected MLP as our base classifier to adapt the same steps as our approach.

The evaluation showed that the best performance was achieved by BERT-MLP (i.e., our main approach) with an accuracy of 92%, while Siamese-BERT-MLP came in second with an accuracy of 79%. Then, running DC-CNN with a higher number of epochs (i.e., epochs = 100) allows us to record an accuracy of 75%. However, Word2VEC was our least impressive model with an accuracy of 69%.

Besides, to better understand and compare our models performances, we specified the scores of precision (resp., recall and F1-score) for each model as shown in Table 7. Also, for each model, we show their associated hyperparameters in Table 8. Results have proven that our approach achieved an improvement of 12% (resp., 17 and 23%) compared to Siamese-BERT (resp., DC-CNN and Word2Vec) in terms of precision. For the next metric (i.e., recall), BERT-MLP achieved the highest score, and the same for F1-score

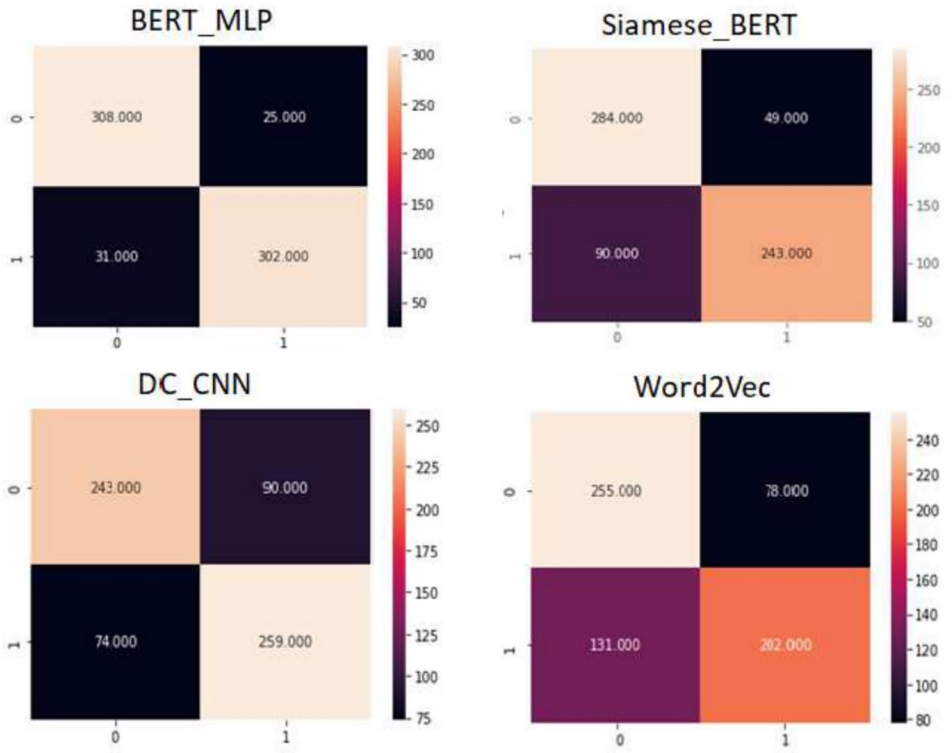


Fig. 6 Confusion matrices of BERT-MLP, Siamese-BERT, DC-CNN and Word2Vec

Table 8 Hyper-parameters of BERT, Siamese-BERT, DC-CNN and Word2Vec

Model	Layers	Embedding-dim	Attention heads	Parameters	Max-length
BERT	12	768	12	150 M	300
Siamese-BERT	12	768	12	150 M	300
DC-CNN	6	100	–	–	300
Word2Vec	4	190	–	150 M	300

our approach achieved the best score compared to Siamese-BERT (resp., DC-CNN and Word2Vec).

In addition, to explore the differences of accuracy between our approach (i.e., BERT-MLP), Siamese-BERT, DC-CNN and Word2VEC, we have studied the confusion matrices for the four approaches as shown in Fig. 6, the class labels are named as follows: 1 for duplicate PRs and 0 for non-duplicate PRs. Hence, according to Fig. 6, MLP-BERT misclassified only 56 PR instances, achieving the lowest False Positive (FP) and False Negative (FN) rate across all models. While, Siamese-BERT (resp., DC-CNN and Word-2Vec) mis-classified 139 (resp., 164 and 209) PR instances.

For more explanation we represent an example of a PR pair in Table 9, where it is correctly classified by BERT-based model, but mis-classified with Siamese-BERT, DC-CNN and Word2Vec. From this table, we can demonstrate the importance of our model. In addition to that, this example shows the impact of pre-training deep bidirectional representation of duplicate PRs detection by allowing the model to learn the context of a word based on all of its surroundings (left and right of the word).

To visualize our data, we have used t-Distributed Stochastic Neighbor Embedding (t-SNE), which visualizes high-dimensional data by giving each data point a location in a two- or three-dimensional map (Hinton and Roweis 2002; van der Maaten and Hinton 2008).

Figure 7 presents the t-SNE plot of the training and testing set with BERT, Siamese-BERT, DC-CNN and Word-2Vec representations. As we can see from the figure, only the data points (i.e., training and testing set) from our based approach (i.e., BERT), are grouped into a well-separated regions. This shows that the BERT model has a better discrimination for the features than Siamese-BERT, DC-CNN and Word2Vec models.

Based of all these analysis and comparison, we can conclude that BERT performed better than the other models.

6 Conclusion and future work

In this paper, we propose an automatic approach to automatically detect duplicate PRs in GitHub. Our method employs textual information (i.e., title and description) to calculate the similarity between two PRs based on a pre-trained

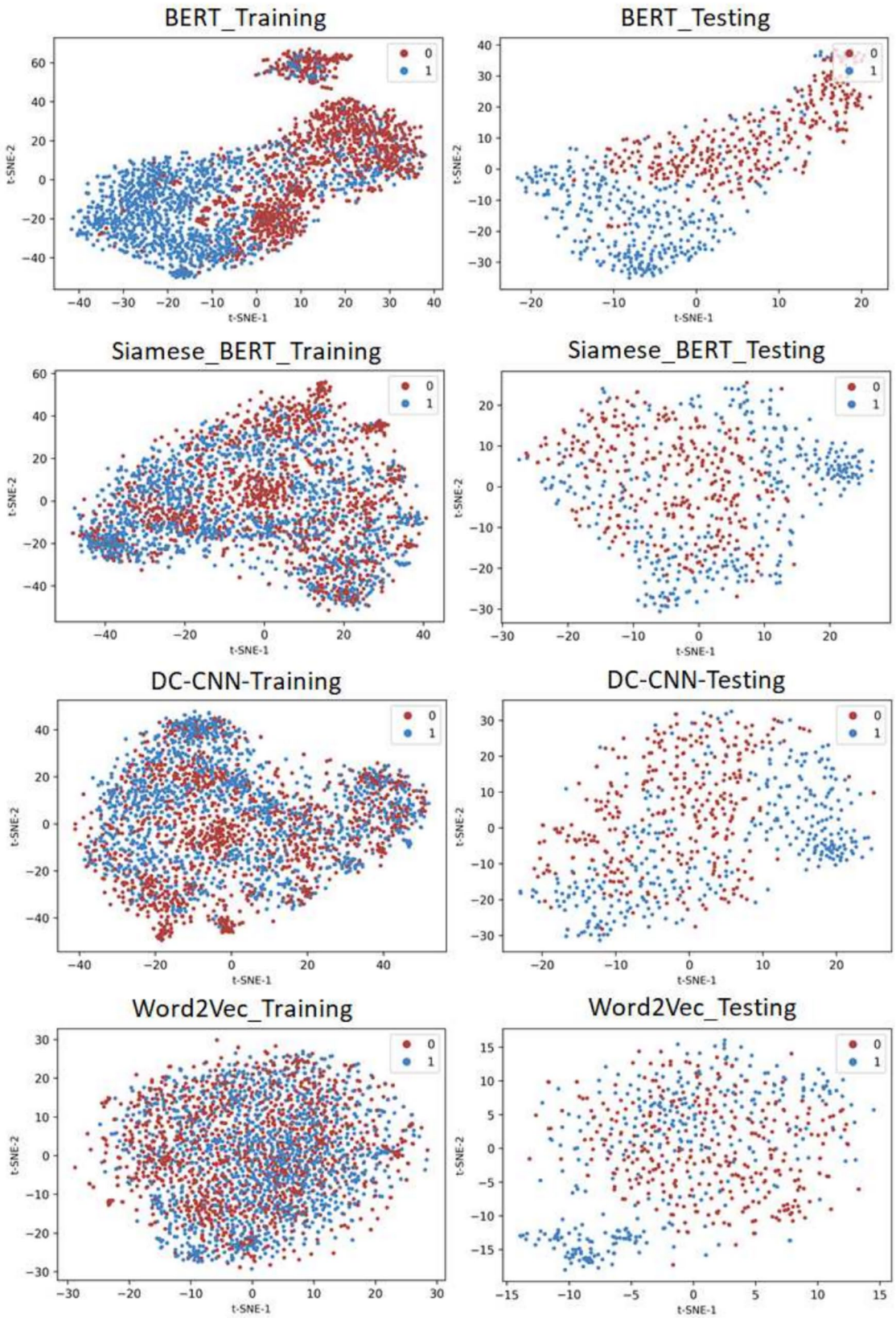


Fig. 7 T-SNE plot for BERT, Siamese-BERT, DC-CNN and Word2Vec

Table 9 Example of a pull request pair

Fields	Pull Request 1	Pull Request 2
Title	Enable Kubelet profiling	Enable profiling on kubelet healthz port
Description	This PR adds a new command line flag to kubelet that enables debug ppr of, similar to what has already been added to apiserver and controller manager. The text was updated successfully, but these errors were encountered	This scheduler are the last two kubeapps without an enable profiling flag. The text was updated successfully, but these errors were encountered

language model named BERT (Bidirectional Encoder Representations from Transformers). To do, we evaluated our approach on a balanced dataset of 3328 labeled PRs collected from 26 popular GitHub projects. This data is entered into the BERT model in order to learn the contextual relationships between the words used in the PR pairs. Then, BERT's model outputs are fed into the Multilayer Perceptron (MLP), which represents our base duplicate PRs detector. The evaluation showed that BERT provided the best performance and achieved an accuracy of 92% with the MLP classifier. Moreover, results have proven that BERT's word representation features achieved a 13% (resp., 17 and 23%) compared to Siamese-BERT model (resp., DC-CNN and Word2Vec) in term of accuracy.

In the future, we plan to explore more features and models that can be used to detect duplicate PRs (e.g., rest of the comments) (Oyelade et al. 2022; Abualigah et al. 2022; Maayah et al. 2022; Arqub and Abo-Hammour 2014). In addition, we aspire to enrich our test dataset and evaluate our method on a larger number of repositories from different social coding platforms.

Funding This study has been funded by the Tunisian Young Researchers' Encouragement Program (Ed. 2022) (22PEJC-D3P2).

Declarations

Conflict of interest Authors have no conflict of interest to declare.

Ethical approval This article does not contain any studies with human participants or animals performed by any of the authors.

References

Abualigah L, Elaziz MA, Sumari P, Geem ZW, Gandomi AH (2022) Reptile search algorithm (RSA): a nature-inspired meta-heuristic optimizer. Expert Syst Appl 191:116158. https://www.sciencedirect.com/science/article/pii/S0957417421014810

Arqub OA, Abo-Hammour Z (2014) Numerical solution of systems of second-order boundary value problems using continuous genetic algorithm. Inf Sci 279:396–415. https://www.sciencedirect.com/science/article/pii/S0020025514004253

Ciborowska A, Damevski K (2021) Fast changeset-based bug localization with BERT. CoRR. arXiv:2112.14169

Devlin J, Chang M-W, Lee K, Toutanova K (2019) BERT: Pre-training of deep bidirectional transformers for language understanding. In: Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, volume 1 (long and short papers). Association for Computational Linguistics, Minneapolis, pp 4171–4186. https://aclanthology.org/N19-1423

Devlin J, Chang M, Lee K, Toutanova K (2019) BERT: pre-training of deep bidirectional transformers for language understanding. In: Burstein J, Doran C, Solorio T (eds) Proceedings of the 2019 conference of the North American chapter of the association for computational linguistics: human language technologies, NAACL-HLT 2019, Minneapolis, MN, USA, June 2–7, volume 1 (long and short papers). Association for Computational Linguistics, pp 4171–4186. https://doi.org/10.18653/v1/n19-1423

Eyal Salman H, Alshara Z, Seriai A-D (2022) Automatic identification of similar pull-requests in GitHub's repositories using machine learning. Information 13(2). https://www.mdpi.com/2078-2489/13/2/73

Feifei X, Shuting Z, Yu T (2020) Bert-based Siamese network for semantic similarity. J Phys Conf Ser 1684(1):012074. https://doi.org/10.1088/1742-6596/1684/1/012074

Ghadhab L, Jenhani I, Mkaouer MW, Messaoud MB (2021) Augmenting commit classification by using fine-grained source code changes and a pre-trained deep neural language model. Inf Softw Technol 135:106566. https://doi.org/10.1016/j.infsof.2021.106566

Gousios G, Pinzger M, Deursen AV (2014) An exploratory study of the pull-based software development model. In: Proceedings of the 36th international conference on software engineering, ser. ICSE 2014. Association for Computing Machinery, New York, pp 345–355. https://doi.org/10.1145/2568225.2568260

He J, Xu L, Yan M, Xia X, Lei Y (2020) Duplicate bug report detection using dual-channel convolutional neural networks. In: Guéhéneuc Y, Hayashi S (eds)Proceedings—2020 IEEE/ACM 28th international conference on program comprehension, ICPC 2020. United States of America: IEEE, Institute of Electrical and Electronics Engineers, 2020, pp 117–127, international Conference on Program Comprehension 2020, ICPC; Conference date: 13-07-2020 Through 15-07-2020. https://dl.acm.org/doi/proceedings/10.1145/3387904, https://conf.researchr.org/home/icpc-2020

Hinton GE, Roweis S (2002) Stochastic neighbor embedding. In: Becker S, Thrun S, Obermayer K (eds) Advances in neural information processing systems, vol 15. MIT Press. https://proceedings.neurips.cc/paper/2002/file/6150ccc6069bea6b5716254057a194ef-Paper.pdf

Kingma DP, Ba, J (2014) Adam: A method for stochastic optimization. arXiv:1412.6980

Li Z, Yin G, Yu Y, Wang T, Wang H (2017) Detecting duplicate pull-requests in GitHub. In: Mei H, Lyu J, Jin Z, Zhao W (eds) Internetware. ACM, pp. 20:1–20:6. http://dblp.uni-trier.de/db/conf/internetware/internetware2017.html#LiYYWW17

Li Z, Yu Y, Zhou M, Wang T, Yin G, Lan L, Wang H (2020) Redundancy, context, and preference: an empirical study of duplicate pull requests in oss projects. IEEE Trans Softw Eng 1–1

Li Z, Yu Y, Wang T, Yin G, jun Mao X, Wang H (2021) Detecting duplicate contributions in pull-based model combining textual and change similarities. J Comput Sci Technol 36:191–206

Maayah B, Moussaoui A, Bushnaq S, Arqub OA (2022) The multistep Laplace optimized decomposition method for solving fractional-order coronavirus disease model (covid-19) via the Caputo fractional approach. Demonstratio Mathematica 55(1):963–977. https://doi.org/10.1515/dema-2022-0183

Mikolov T, Chen K, Corrado G, Dean J (2013) Efficient estimation of word representations in vector space. arXiv:1301.3781

Nugroho KS, Sukmadewa AY, Yudistira N (2021) Large-scale news classification using BERT language model: spark NLP approach. CoRR. arXiv:2107.06785

Oyelade ON, Ezugwu AE, Mohamed TIA, Abualigah LM (2022) Ebola optimization search algorithm: a new nature-inspired metaheuristic optimization algorithm. IEEE Access 10:16 150-16 177

Pedregosa F, Varoquaux G, Gramfort A, Michel V, Thirion B, Grisel O, Blondel M, Prettenhofer P, Weiss R, Dubourg V, VanderPlas J, Passos A, Cournapeau D, Brucher M, Perrot M, Duchesnay E (2012) Scikit-learn: machine learning in python. CoRR. arXiv:1201.0490

Peters ME, Neumann M, Iyyer M, Gardner M, Clark C, Lee K, Zettlemoyer L (2018) Deep contextualized word representations. In: Walker MA, Ji H, Stent A (eds) NAACL-HLT. Association for Computational Linguistics, pp 2227–2237. http://dblp.uni-trier.de/db/conf/naacl/naacl2018-1.html#PetersNIGCLZ18

Radford A, Narasimhan K (2018) Improving language understanding by generative pre-training

Ren L, Zhou S, Kästner C, Wasowski A (2019) Identifying redundancies in fork-based development. In: 2019 IEEE 26th International conference on software analysis, evolution and reengineering (SANER), pp 230–241

Robbins H, Monro S (1951) A stochastic approximation method. In: The annals of mathematical statistics, pp 400–407

van der Maaten L, Hinton G (2008) Visualizing data using t-sne. J Mach Learn Res 9(86):2579–2605. http://jmlr.org/papers/v9/vandermaaten08a.html

Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser LU, Polosukhin I (2017a) Attention is all you need. In: Guyon I, Luxburg UV, Bengio S, Wallach H, Fergus R, Vishwanathan S, Garnett R (eds) Advances in neural information processing systems, vol 30. Curran Associates, Inc. https://proceedings.neurips.cc/paper/2017/file/3f5ee243547dee91fbd053c1c4a845aa-Paper.pdf

Vaswani A, Shazeer N, Parmar N, Uszkoreit J, Jones L, Gomez AN, Kaiser L, Polosukhin I (2017b) Attention is all you need. CoRR. arXiv:1706.03762

Wang Q, Xu B, Xia X, Wang T, Li S (2019) Duplicate pull request detection: when time matters. In: Proceedings of the 11th Asia-pacific symposium on internetware, ser. Internetware'19. Association for Computing Machinery, New York. https://doi.org/10.1145/3361242.3361254

Wu Y, Schuster M, Chen Z, Le QV, Norouzi M, Macherey W, Krikun M, Cao Y, Gao Q, Macherey K, Klingner J, Shah A, Johnson M, Liu X, Kaiser L, Gouws S, Kato Y, Kudo T, Kazawa H, Stevens K, Kurian G, Patil N, Wang W, Young C, Smith J, Riesa J, Rudnick A, Vinyals O, Corrado G, Hughes M, Dean J (2016) Google's neural machine translation system: bridging the gap between human and machine translation. CoRR. arXiv:1609.08144

Yu Y, Wang H, Yin G, Wang T (2016) Reviewer recommendation for pull-requests in GitHub: what can we learn from code review and bug assignment? Inf Softw Technol 74:204–218. https://www.sciencedirect.com/science/article/pii/S0950584916000069

Yu Y, Li Z, Yin G, Wang T, Wang H (2018) A dataset of duplicate pull-requests in GitHub. In: Zaidman A, Kamei Y, Hill E (eds) MSR. ACM, pp 22–25. http://dblp.uni-trier.de/db/conf/msr/msr2018.html#YuLYWW18

Zhu Y, Kiros R, Zemel R, Salakhutdinov R, Urtasun R, Torralba A, Fidler S (2015) Aligning books and movies: towards story-like visual explanations by watching movies and reading books. In: 2015 IEEE International conference on computer vision (ICCV), pp 19–27