UNIVERSITY OF TARTU
Faculty of Science and Technology
Institute of Computer Science
Computer Science Curriculum

Behrad Moeini

Detecting semantically equivalent issue reports using transformer models

Master's Thesis (30 ECTS)

Supervisor(s): Ezequiel Scott, PhD

Tartu 2021

Detecting semantically equivalent issue reports using transformer models

Abstract:
Developers support their software development by creating issue reports that can describe bugs, feature requests, or change requests. As the project grows over time, the number of issue reports also grows in number, and some issues are reported multiple times by different users. To avoiding this issue, several automated approaches have been proposed for retrieving duplicate issue reports. These approaches have been mainly based on information-retrieval techniques. This thesis aims to explore recent advances to detect semantically equivalent text to identify duplicate issue reports. Since several articles are published on this topic, this thesis's main challenge will be to replicate the existing approaches and compare their performance with the proposed solution. Part of my work is to extract and curate the data from sources such as issue trackers. This thesis will be tackling this as a natural language processing problem and apply advanced techniques to classify whether question pairs are duplicates or not. In this thesis, we take an open-source dataset from GitHub, which many projects have been done on that, so it is easy to compare the result with a different result. We applied different models build a model to detect whether two questions are semantically the same, beginning with simple models and use more complex models step by step. When we applied our model to the dataset that we have and got each model's result, we take each model their performances and see how are their results.

Keywords:
Github, Duplicated question, Natural language processing, Transformer model, Neural network

CERCS: P170 Computer science, numerical analysis, systems, control

Tüübituletus neljandat järku loogikavalemitele

Lühikokkuvõte:
Arendajad kasutavad tarkvara arendusprotsessi luues probleemide raporteerimist, mis võivad kirjeldada koodi vigu, soovitud uut funktsionaalsust või muudatusettepanekuid. Projekti kasvades aja jooksul raporteeritud probleemide hulk kasvab ja mõnda probleemi esitatakse erinevate kasutajate poolt mitu korda. Selle vältimiseks on välja pakutud mitmeid automatiseeritud lähenemisi korduvate probleemi raportite leidmiseks. Peamiselt on need lähenemised baseerunud informatsiooni leidmise tehnikatel. Käesolev uurimistöö keskendub semantiliselt samaväärse teksti tuvastamisele ja valdkonna viimastele edusammudele korduvate probleemi raportite leidmiseks. Antud teemal on publitseeritud mitmeid artikleid ja selle uurimistöö peamine väljakutse on olemasolevaid lähenemisi jäljendada ja võrrelda tulemusi töös välja pakutud uue lahendusega. Osa tööst on seotud probleemihalduse andmetest vajaliku info leidmise ja eraldamisega. Käesoleva uurimustöö raames välja arendatud lahendus läheneb küsimuste paari samaväärseks või mitte klassifitseerimisele kui loomuliku keele töötluse probleemile. Uurimistöös kasutatakse Github'ist võetud avaandmeid, mida on kasutatud mitmetes projektides, mis muudab tulemuste võrdlemise teiste töödega lihtsamaks. Küsimuste paari semantilise võrdsuse üle otsustamiseks kasutati erinevaid mudeleid - alustati lihtsamatest ja samm-sammult jõuti keerukamate mudeliteni. Iga mudelit rakendati samadele andmetele ja kõiki nende tulemusi võrreldi omavahel.

Võtmesõnad:
Github, korduva sisuga küsimus, loomuliku keele töötlus, Transformer mudel, närvivõrgud

CERCS:P170 Arvutiteadus, arvutusmeetodid, süsteemid, juhtimine (automaatjuhtimisteooria)

Contents

1 Introduction

Issue trackers such as Jira[1] are an exceptional success as they simplify the interaction between users and developers. Popular software projects may receive a considerable number of issues to respond to, and developers need to maintain issue reports. During the maintenance of issue report repositories, one common task is to remove duplicates. A point that must be considered during the removal of duplicates is that many of the issue reports may have the same answer or solution. Therefore, having a system that can help developers filter semantically equivalent issue reports needs an application that understands the text and compares them together.
For such a task, several approaches can be taken to solve it. As machine learning algorithms are flourishing in natural language processing, scientists take advantage of different techniques to tackle this problem. For example, topic modeling[5], discriminative methods[31], and meta-attributes[9].
In this thesis, transformer models are used for detecting duplicated issue reports. Transformer models are good candidates for this task [26]. The design of transformer models is in the way to work with a sequence of data. As texts are sequences of data, transformer models can be a perfect choice. In this study, transformers are used for embedding, which is converting long texts to fixed-size vectors. By this process, similar vectors corresponding to duplicated issue reports are similar and can be detected.
Available datasets include different features. Each feature can be helpful for the categorization of the data. A publicly available dataset, BugRepo [2] is used to conduct several experiments with transformer models. The experiments included the use of pre-trained models, the training of models, and the choice of the best algorithm for tex similarity measurement between two embedded texts. Furthermore, the most appropriate cleaning steps are using pre-trained transformer models and training a transformer architecture model.

1.1 Problem statement

The problem is to detect whether two of the issue reports are duplicated. In order to achieve this task, we are required to understand if two of the issue reports are semantically equivalent.
To do this task, after choosing suitable text similarity metrics and cleaning steps, we select six of the best pre-trained transformer models. We do embedding process using these models, and by the text-similarity metrics, we can see how is the performance of this work. The next step is to train a transformer model which uses the train.csv file in the dataset. Then, we can measure the performance of the trained model and the selected

[1] https://www.atlassian.com/software/jira
[2] https://github.com/logpai/bugrepo

text similarity metrics. In this context, the research questions are as follows:

RQ1: What is the performance of available pre-trained transformer models when detecting duplicate issue reports?
RQ2: What is the performance of training models made of transformer models when detecting duplicate issue reports?
RQ3: How different cleaning steps can change the performance?

For the first research question(RQ1), we compare the performance of six available pre-trained transformer models. Since among many pre-trained transformer models, it is more suitable to use models that work better with larger texts such as RoBERTa. Because the input data we have is a large text, it is better to use suitable models. The pre-trained transformer models currently available use the issue report dataset to convert the content of the issue reports to fixed-sized vectors. Then, the similarity of these vectors is compared using the chosen text similarity measures to determine duplicated issue reports. We want to compare which model works better to find duplicate issue reports.
For the second research question(RQ2), we train a model using the dataset instead of utilizing pre-trained models. As in RQ1, we use the model to observe if the performance is better than previously used models. The performance of the models were compared using recall rate[27] since there are a number of related works that used this metrics for their models.[27] [9] [32]

The third research question(RQ3) is about testing whether some information that we usually remove in natural language processing tasks might be helpful to and enhance the performance in this task. We wonder if keeping links and parenthesis could help the performance in this task. In order to answer RQ3, we trained different transformer models using different versions of the dataset, which passed different data cleaning steps and compared their performance.

1.2 Thesis outline

In chapter 2, we introduce related works that were done which are related to this work. Among these, some of the works have been done on the same dataset. In chapter 3, required background knowledge used in this thesis provided. After this, In chapter 4, we will explain the work we have done in detail step by step. By providing the steps, we can reach the chapter chapter 5 that included all tabtwo les and figures of the results explained in its previous chapter. Then, by reaching this chapter, we discuss the research question by the result we achieved in chapter chapter 6.

2 Background

2.1 Text similarity methods

Text similarity has to determine how *close* two strings of text in the lexical term and their semantics [3]. In this study, we focus on the similarity of texts in terms of their semantics. There are plenty of text similarity methods for our purpose. The algorithms lie in several categories. The categories we used in this study are listed as follows[4]:

- **Token based**: Looking at units(tokens) of two strings(i.e. words, n-grams etc)
- **Sequence based**: Looking at different subsequences of two strings
- **Hybrid**: Edit based + Token based

2.2 Transformers

Transformers are deep learning models that use attention to measure the influence of different parts of the input data.[35]

For understanding transformers, it is essential to get familiar with attention. The attention mechanism looks at an input sequence and decides which other parts of the sequence are important. For instance, in texts, it reminds important keywords of texts in memory in order to provide context.[5]

A transformer model handles variable-sized input exploitation stacks of self-attention layers rather than RNNs or CNNs. This general design includes a variety of advantages:

- It makes no assumptions regarding the temporal/spatial relationships across the information. this can be ideal for process a collection of objects (for example, StarCraft units).
- Layer outputs will be calculated in parallel, rather than a series like an RNN. Distant items will have an effect on every other's output while not passing through several RNN-steps, or convolution layers (see Scene Memory transformer for example).
- It will learn long-range dependencies. this can be a challenge in several sequence tasks. [6]

[3]https://medium.com/@adriensieg/text-similarities-da019229c894
[4]https://activewizards.com/blog/comparison-of-the-text-distance-metrics/
[5]https://medium.com/inside-machine-learning/what-is-a-transformer-d07dd1fbec04
[6]https://www.tensorflow.org/tutorials/text/transformer

2.3 Pre-trained transformer models

Pre-trained transformer models are models created previously to solve a similar problem. Instead of building a model from scratch to solve a similar problem, it is possible to get used to that model trained on other similar problems as a starting point.[7] There are some advantages and disadvantages to using pre-trained models.
Advantages of the pre-trained models are:

- They are optimized to find duplicate texts, a similar sequence of strings and questions which made the work easier and can help the performance
- The are trained with large datasets, which makes it immune to underfit
- Because they are trained, it makes the process so much faster because it does not need the training step.

Furthermore, disadvantages of the pre-trained models are:

- The optimization made for a specific dataset and may not work for other datasets in different domains.
- There is no way to improve the model using the dataset we obtained.
- Using the training file is useless here, and many data will be unused.

[7] https://www.analyticsvidhya.com/blog/2017/06/transfer-learning-the-art-of-fine-tuning-a-pre-trained-model/

3 Related works

In this chapter, we present a brief review and description of works that have been studied in this field. Some works focus on different approaches to find semantically equivalent works, and some others are to detect duplicated issue reports.

3.1 Detecting semantically equivalent texts

There have been quite a few research works on detecting duplicate data to detect duplicate long texts, such as issue reports. Many works have been done on the Quora dataset, and the experience of using this dataset is rich.

The method used a convolutional neural network(CNN), long short term memory networks (LSTMs), and a hybrid model of CNN and LSTM layers. Their best model is the LSTM network that achieved an accuracy of 81.07% and an F1 score of 75.7%. In that work, GloVe word vector of 200 dimensions trained using a large amount of Twitter[8] words was used in the experiments.

Some of the researchers tried to attack this problem with deep learning approaches. For instance, in [3], they utilize a Siamese network structure [23]. In this network structure, each query processes into an individual branch with unique parameters and weights. Firstly, queries are modified into vectors. Secondly, they converted into word embedding using pre-trained Glove [15]. This embedding will be used in the encoding layer. It is needed to modify the matrix, which included the words to feature vectors. After this step, feature vectors are connected using the method proposed by Bowman et al. [28]. In the end, feature vectors go into a multi-layer perceptron for the final output. The authors have experimented with three different encoding methods. This method was CNN based on what Yoon [17] and Bogdanova et al. [12] proposed. The encoding method had a feature vector of size 328 as an output. Another encoding method used bidirectional LSTM on what was proposed by Wang et al. [38]. The encoding method built 10N features where N is the number of words in a sentence. The third encoding method was a hybrid of the two methods. This method applies the bidirectional LSTM and the CNN method one after the other to get a feature vector. The final step consisted of merging the feature vectors of branches of the Siamese neural network. For this, they used a multi-layer perceptron. By doing so, it grows by 2% in performance compared to the merge of feature vectors. In Stanford report[36], a method proposed which used Siamese GRU neural network. It used the network to encode sentences and compare them using different distance measurements. This approach had a few necessary steps. The first step is data processing. To do so, it involved tokenizing the sentences Stanford Tokenizer [9]. This step additionally concerned modifying every question to a fixed length

[8] https://nlp.stanford.edu/projects/glove/

[9] https://nlp.stanford.edu/software/tokenizer.html

for permitting batch computation using matrix operations. The second step involves sentence encoding. They used each recurrent neural network(RNN) and gated recurrent unit (GRU) during this step. What they did was initialize the word embedding to the GloVe vectors[15]. After this, they set the distance measure[16] that determined whether two encoded sentences are semantically equivalent. There have been two approaches for this step: calculative distances between the sentence vectors and running logistical regression to create the prediction. In the paper, cosine similarity, euclidian distance, and weighted Manhattan distance were used to test the performance. The matter here is that it is troublesome to grasp the natural distance measure encoded by the neural network. Researchers replaced the distance function with a neural network to tackle this issue and exploit it to the current neural network to leave distance performance. They provided a row concatenated vector as input to the neural network and conjointly experimented using one layer and two-layer within the neural network. The paper utilized data augmentation as an Associate in Nursing approach to reduce overfitting. They also did a hyperparameter search by tuning the dimensions of the neural network hidden layer. Therefore, the standardized length of the input sentences opens up to a raised performance.[7]

In the determining literature entailment of questions in the Quora, Dataset [34], authors have used word ordering and word alignment employing a long-short-term-memory(LSTM) recurrent neural network[13], and therefore the decomposable attention model severally and tried to mix them into the LSTM attention model to attain their best accuracy of 81.4%. Their approach concerned implementing numerous models planned by multiple papers made to see the sentence implication on the SNLI dataset. [10] Seven a number of these models are the Bag of words model, RNN with GRU and LSTM cell, LSTM paying attention, complex attention model. a number of the challenges they baby-faced in implementing these models were the difficulty with memory as a result of the hugeness of the dataset and conjointly problems with overfitting that they tried to tackle by introducing drop out and regularization. Doing a sentence analysis showed that completely different models have their pros and cons in several sentence pairs. However, sentences similar grammatically with words out of vocabulary were higher classified with word-by-word and two-way-word-by-word attention models. On the opposite hand, LSTM attention model performed well to classify sentences with words tangentially connected. However, in cases wherever words within the sentences have a particular order, the complex attention model [2] achieves higher performance. This paper tried to mix the GRU/LSTM model with the complex attention model to achieve from the advantage of each and are available up with higher models with higher accuracy such as LSTM with Word by Word Attention, and LSTM with two means Word by Word Attention. In the relevant literature, "Detection of Duplicates in Quora and Twitter Corpus" [29], the work has experimented with six classifiers. They used a

[10]https://nlp.stanford.edu/projects/snli/

straightforward approach to extract six simple features: word counts, familiar words, and term frequencies(TF-IDF)[25] on question pairs to coach their models. During this work, the most straightforward accuracy reportable is 72.2% and 71.9% obtained from binary classifiers random forest and KNN, severally [7].

3.2 Detecting duplicated issue reports

Murphy G et al. [21], which is preparatory works done on the subject, planned a machine learning-based approach to automatically assigning bug reports to developers. In their work, they treated the problem as a text categorization task. Description of the problem reports was used as an information source and described as a bag of words representation supported term frequency and used the Naïve Bayes algorithmic program to train a model on this representation which will classify issue reports among the developers and automatically assigned to them. This approach is used on issue reports collected from the Eclipse project and were able to accomplish accuracy for up to 30. Ahsan sn et al. [4] is another related work that used machine learning to classify bug reports for automatic assignments. Automatic bug triage system using latent semantic classification and support vector machine. Similar to work by murphy G et al. [21], they relied entirely on the outline of the problem reports as an information source. However, the source different in two things. First, they used TF-IDF weighing-based VSM representation for the issue reports, and that they additionally applied dimensionality reduction and latent semantic categorization strategies for feature choice. The opposite distinction is that they used the SVM algorithmic program for classification. During this approach, they were able to reach up to 44.4% accuracy on the Mozilla project. Nasim S et al. [22] used the frequency of every alphabet rather than terms within the bug outline as options for eleven completely different classification algorithms to predict the developer to be allotted. They used the Eclipse JDT project for his or her experiments. They experimented with their approach using the labeled issue reports solely and using all collected issue reports. The most effective result they achieved was using solely the labeled issue reports' tag information, which gave 62 with the J48 decision tree algorithm. However, not all issue report descriptions contain tags; from the issue reports they collected, half the issue reports contained tags in their description, creating it arduous to suppose tags entirely. Using all collected issue reports, the most effective accuracy they were able to win was solely 32nd. Not all research works on the issue report assignment problem relied on machine learning algorithms. For instance, Tamrawi A et al. [33] proposed a fuzzy sets-based approach. In their approach, for each technical term in bug reports, they kept a record of a fuzzy set of the developer's relevancy. The term supported the issue reports fixed by the developers antecedently. For a brand new issue report, they graded the developers based on their membership score to the fuzzy set of the new issue report, calculated based on the fuzzy set theory [18]. They allotted the issue

report to the developer with the very best membership score. Using this approach, they achieved the most accuracy of 37.81% and the top 5 accuracies of on average with issue reports collected from the Eclipse project. To improve the representation of the textual description of issue reports, recent work by Mani S et al. [19] has applied an additional advanced illustration using deep learning. They used a deep bidirectional recurrent neural network to determine the meaning of the description of the issue reports in an unsupervised manner. Applying this sort of representation on issue reports by using Naïve Bayes, SVM, and cosine distance-based classifiers for predicting developers reported enhancements in the TF-IDF-based bag of word representation accuracy. However, their best results can still be better. [11]

Other than these works, some works used the same or similar datasets, that each of them had its limitations. The papers provided in Table 1 with some related information.

Table 1. Summary of some related works

Name	**Year**	**Methods**	**Dataset**	**Size**	**Limitations**
A Contextual Approach towards More Accurate Duplicate Bug Report Detection [6]	2015	Exploits domain knowledge: preprocess to gather several word lists to extract the context implicit in each report	Android Open Source Project	37236 bug reports	For each repository, it should have a separate preprocess-ing and take so long
Detecting Semanti-cally Equivalent Questions in Online User Forums [8]	2015	CNN model	Wikipedia and AskUbuntu	1.6B tokens and 121M tokens	Word embedding should be pretrained on in-domain data

Learning Hybrid Representations to Retrieve Semantically Equivalent Questions [30]	2015	BOW-CNN combines a BOW presentation with a distributed vector made by a CNN	Two Stack Exchange communities (AskUbuntu and English)		Not the best result on short texts using BOW-CNN
Combining Word Embedding with Information Retrieval to Recommend Similar Bug Reports [37]	2016	propose the first similar bug recommendation system named NextBug	BugRepo	763,729 bug reports	Threats to external validity should be reduced The parameters of the word embedding technique should be optimized.
LWE: LDA Refined Word Embeddings for Duplicate Bug Report Detection [10]	2018	Combining LDA and word embedding	BugRepo	565,668 issues	Do not use any signals from the tagged duplicate reports

Deep Word Embedding Network for Duplicate Bug Report Detection in Software Repositories[9]	2018	Combine word embeddings, TF-IDF and meta-attributes	BugRepo	565,668 issues	The two-step training process of training embeddings and deep neural network

4 Methodology

There are several steps involved in this project. In this section, listing the steps will be followed by a detailed explanation of each step about the work that has been done. The first steps consists of preparing the data and building the models. Then, the hyperparameters are optimized and the results evaluated.
Step 1: Manipulating Data
Step 2: Building models
Step 3: Optimizing hyperparameters
Step 4: Evaluating results

Figure 1 shows the order of the steps. Firstly, by cleaning and pre-processing the dataset, we will achieve three different versions of the datasets. The datasets D1, D2, and D3 passed through different pre-trained models and trained models. As shown in the figure, it is clear how each step of the work will contribute to answering one of the research questions.

4.1 Step 1: Manipulating Data

4.1.1 Dataset

The dataset used in the experiment is known as BugRepo[11]. This dataset is a collection, publicly available, of issue reports for research purposes. It is mainly built to facilitate NLP-based research in software engineering. The main reason to choose this dataset was the various works on this dataset related to finding duplicate issue reports. [9] [10] [37]

What other important factor can be seen is the available percentage of duplicates. It is essential to know beforehand because it significantly impacts the evaluation metrics we use, so it should be considered. There are different software projects included in this dataset, but we work on four different projects with the most number of issues for this thesis. The targeted projects are Mozilla Core, Firefox, Eclipse Platform, and JDTsteps. There is several additional information provided in the dataset, such as the number of issues, duplicates, and percentage of duplicates. Table 3 shows the first rows of the dataset. A brief detail of each project is provided in Table 2.

We only consider the field title and description of the issue reports in this study.

The Bugrepo dataset is available as a GitHub project. There is a separate directory for each project in the Github project. In each directory, there are three files: Two of

[11]https://github.com/logpai/bugrepo
[12]https://github.com/logpai/bugrepo

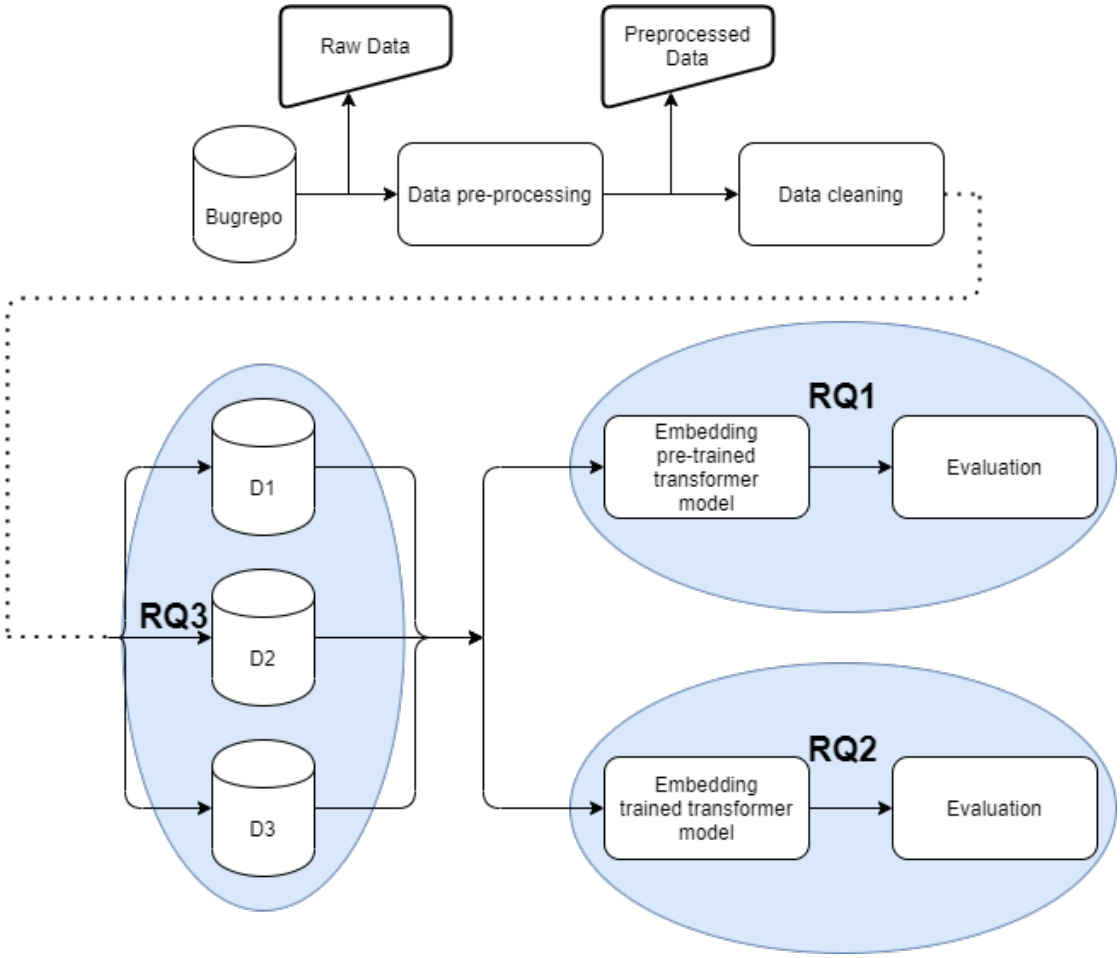

Figure 1. Diagram of the study in this thesis

Table 2. Details of the software projects in the bugrepo repository

12

Project	#Components	#Issues	#Duplicates	%Duplicates
Mozilla Core	130	205,069	44,691	21.8%
Firefox	52	115,814	35,814	30.9%
JDT	6	45,296	7,688	17.0%
Eclipse Platform	21	85,156	14,404	16.9%

Table 3. Header of the JDT dataset

Issue ID	Priority	Component	Duplicated issue	Title	Description	Status	Resolution	Version	Created time	Resolved time
1518	P1	Debug		Icons needed \for actions (1GI5UXW)	JGS (8/8/01 5:20:19 PM); We need enabled; disabled ...	VERIFIED	FIXED	2.0	2001-10-10 22:14:00 -0400	2001-10-18 11:51:14 -0400
1519	P3	Debug		README: Hit count not reset (1GET20Y)	JGS (6/4/01 11:43:47 AM); Set a breakpoint in InfiniteLoop ...	RESOLVED	WORKS FORME	2.0	2001-10-10 22:14:00 -0400	2001-11-28 13:42:46 -0500
1520	P3	Debug		Use styled text in console (1G9S1YF)	DW (2/26/01 5:56:36 PM); Could allow users to use styled ...	CLOSED	WONTFIX	2.0	2001-10-10 22:14:00 -0400	2002-06-26 11:32:05 -0400
1521	P3	Debug		StringBuffer representation (1GE3BFA)	JGS (5/21/01 2:09:10 PM); To be consistent; should we ...	VERIFIED	FIXED	2.0	2001-10-10 22:14:00 -0400	2002-02-05 10:37:09 -0500
1522	P2	Debug	1663.0	Feature: use #toString to display variable values (1G1Y25J)	DW (9/27/00 11:12:31 AM); Currently; the type of an object...	RESOLVED	DUPLICATE	2.0	2001-10-10 22:14:00 -0400	2001-10-17 10:17:47 -0400
1523	P3	Debug	1555.0	Breakpoint in an invalid location (1G4F8P8)	DS (11/14/00 2:59:11 PM); You can set a breakpoint that ...	RESOLVED	DUPLICATE	2.0	2001-10-10 22:14:00 -0400	2002-09-10 17:00:45 -0400
1524	P2	Debug		Feature: Displaying Instruction pointer (1G3A7CG)	DW (10/23/00 11:08:55 AM); We currently show the ...	VERIFIED	FIXED	2.0	2001-10-10 22:14:00 -0400	2002-11-22 09:33:05 -0500
1525	P2	Debug		Feature: Locks and Monitors (1G3A7ZH)	DW (10/23/00 11:22:46 AM); Support to show which ...	VERIFIED	FIXED	2.0	2001-10-10 22:14:00 -0400	2002-09-30 12:24:15 -0400
1526	P3	Debug	1633.0	Snippet evaluation should support imports (1G47213)	I want to write code like this in a snippet file; ; import ...	RESOLVED	DUPLICATE	2.0	2001-10-10 22:14:00 -0400	2002-03-18 15:23:50 -0500
1527	P3	Debug		{[}scrapbook{]} Snippet editor color snippet output in java style (1G4730W)	Coloring the snippet output in java style results in funny result...	RESOLVED	WONTFIX	2.0	2001-10-10 22:14:00 -0400	2009-08-30 02:22:18 -0400

them are ***train.csv*** and ***test.csv***. The other file contains all the issue reports of the projects along with different fields that include the issue ID, title, and description. It can be either zip or CSV format file.

In the train and test files, each record of data includes an issue ID, and the second column includes all of the issue IDs which are duplicated with the corresponding issue ID.

As mentioned, each project divided the train and test split itself. This is useful because related works help us make comparisons of our results with the results from previous studies. Having train/test files allows us to train models with the same data that others did. The size of each project's training file is set by the total size of the issue report project dataset. Another point that makes this split suitable to work with is that the duplicated ratio does not differ much, so the training will not be over-fitted or under-fitted. A summary of the size of each project provided in Table 4

Table 4. Details about the size of train and test datasets for the project targeted in this thesis

13

Project	Total (+/-)	Train (+/-)	Test (+/-)
Mozilla Core	205,069 (54,237/150,832)	164,055 (50,122/113,933)	41,014 (4,115/36,899)
Firefox	115,814 (34,262/81,552)	92,651 (30,026/62625)	23,163 (4,236/18,927)
JDT	45,296 (10,127/35,169)	36,236 (8,859/27,377)	9,060 (1,268/7,792)
Eclipse Platform	85,156 (19,845/65,311)	68,124 (17,518/50,606)	17,032 (2,327/14,705)

4.1.2 Data cleaning

There are some existing libraries and packages to clean data. For example, NumPy [14] [14] provides several modules for conducting general text data cleaning. These modules will remove all the words that are not in a predefined list of common vocabulary. In this study, which focuses on issue reports, many words can be detected as out-of-vocabulary(OOV), but they have additional information in reality. An example of this kind of text is the name of classes and files, which the name of these libraries are not in the commonly used vocabulary. It can be detected as OOV and not be considered. Therefore, the standard packages for text data cleaning cannot be used, and we define context-specific data cleaning steps. The steps are:

1. Convert to lowercase: we do not want to differentiate between capitalized words and non capitalized ones.
2. Remove of Numbers and Punctuations: They do not have any information to detect whether two texts are duplicated.
3. Remove stopwords: Such as and, or, the. Deleting them makes it easier for a network to detect the semantic of a text.
4. Remove links: In this study, it will be checked whether this action helps the performance.
5. Remove parenthesis, brackets, and their contents: In this study, it will be checked whether this action helps the performance.

In order to answer RQ3, we decide to check whether steps 4 and 5 in the data cleaning steps enhance the performance or not. In other words, we want to see whether **links** or **parenthesis, brackets and their contents** add information to the text, or they will be considered as a noise and hurt the performance. There is a reason why we chose these two steps. It can be observed that in issue reports, the number of parentheses, brackets, and links is considerable. We would like to see if they add more information to understanding the semantics of the text.
As a result of applying the selected data cleaning steps, we created three different versions of the original dataset:

- Cleaned data 1: No links and parenthesis in the body of the issue report are kept. Thus, steps one to five are applied.
- Cleaned data 2: All the parenthesis in the body of the issue report are kept. Thus, steps one to four are applied.

[13] https://github.com/logpai/bugrepo
[14] https://numpy.org/

- Cleaned data 3: All the links in the body of the issue report are kept. Thus, steps one to five except four are applied.

Thus, we have three different versions of cleaned data, where all the content of the issue reports have the format in Figure 1.

Issue ID | cleaned Title | cleaned Description

Figure 2. Format of the cleaned data

4.1.3 Data pre-processing

After cleaning the data in three different ways. We apply several manipulation steps to make the datasets suitable as input to the transformer models since the transformer models require an input of the form of texts and not IDs. Therefore, we pre-process ***train.csv*** and ***test.csv*** as follows:

We process each file as follows: Since the transformer models (pre-trained and trained) require the input to be in the form of only one string, we first concatenate the title and description of each issue report into only one field. Figure 3 shows an example of this transformation. As it can be seen, in the last step of pre-processing, we should substitute issue ID with their corresponding texts. These steps were applied to the three datasets (D1, D2, D3) that were described in section 4.1.2.

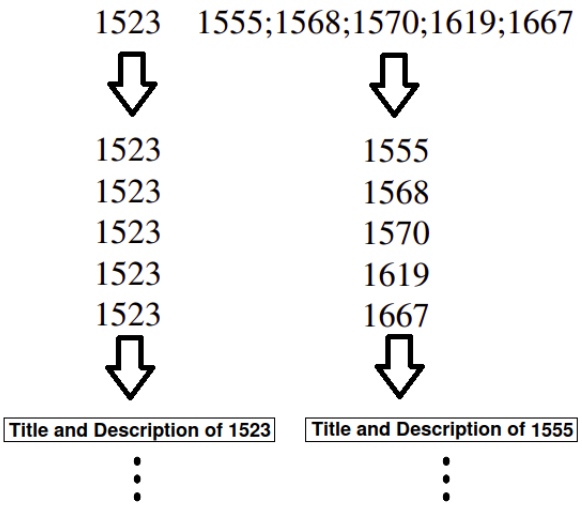

Figure 3. Manipulation of train.csv and test.csv

At last, there is a point that should be addressed. To feed the data into different networks, we should add pairs of non-duplicated and the training file of the dataset. Up to this point, we have duplicated pairs of issue reports. First, we add a column to the existing duplicated pairs with the value 1, which shows that these pairs are duplicated. Then, we add all pairs of issue reports which are not duplicated in the dataset with the value 0 in the duplicated column.

As we have three different versions of cleaned data, we will have three different versions of datasets named D1, D2, and D3 corresponding to cleaned data 1, cleaned data 2, and cleaned data 3, respectively. The cleaning steps, as mentioned earlier, applied to them. Also, each dataset has a file for train data and another one as test data. A sample example of training file of the datasets is shown in Figure 4.

Title+Description of issue report	Title+Description of issue report	Duplicated?
Issue #1	Duplicated issue #2	1
Issue #3	Unrelated issue #4	0

Figure 4. Sample of train file of datasets D1, D2, or D3

4.2 Step 2: Building models

In this step, we build several models using transformers. Transformers are a type of neural network used in neural machine translation, mainly involving tasks that transform an input sequence to an output sequence. [15] [35] In this study, the input sequence is the text of issue reports and the output sequence is a fixed-size vector of float numbers. This vector embeds the information so that the output vectors of two similar texts have a small distance from each other.

In order to determine duplicated issue reports, our approach requires a transformer model. Then, a text similarity algorithm that takes two embedding vectors is required to see whether two reports are duplicated or not. The format of this work is visualized in Figure 5.

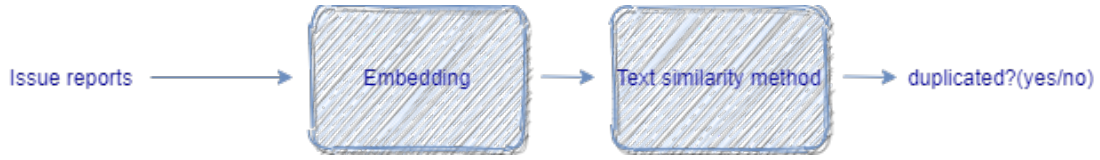

Figure 5. Overview of the building model

In the following, in section 4.2.2, we will use choose pre-trained models and utilize them to get the result. After doing this step, we will train transformer models in section 4.2.3 for each project.

4.2.1 Libraries used for implementation

To build the transformer model, Python provides various libraries with many features, so we use Python for implementation. Among different available libraries, we use the SentenceTransformer framework[16]. In the following, a brief explanation of what is sentence transformer library, how it works, and why we chose it is provided.

SentenceTransformers[17] is a Python framework for cutting-edge sentence and text embeddings. There are different structures to build sentence and text embeddings in this library. Moreover, it has quite a few pre-trained models for different purposes.[24] Because all of their interfaces are the same, it is easy to implement various libraries quickly. A comparison of different models based on one library with a single interface is more reliable. Not using libraries, implementing from scratch can cause some minor changes in the performance due to the difference in self-implementation, not because of the models in their natures. The library allows us to choose among different models,

[15] https://www.kdnuggets.com/2019/10/research-guide-transformers.html
[16] https://www.sbert.net/index.html
[17] https://www.sbert.net/index.html

and it eases the implementation. In a step of our work, we will implement a Sentence-Transformers model, and this work can be quickly done. It provides most of the building blocks that one can put together to enhance embeddings for a specific task, such as the task this thesis is attacking.

In order to be consistent with the implementation, it is better to use all of the text-similarity method candidates from one library. We used strsimpy library[18] which provided an implementation of the methods we wanted to use in our experiment.

4.2.2 Pre-trained transformer models

This section will choose pre-trained transformer models among available models and utilize them to embed the targeted texts. Table 5 shows a summary of the pre-trained models available in the SentenceTransformer library. The column "Training data" shows the corresponding model was trained with which dataset. There are Paraphrase Data, STSb [19] and SNLI[20] datasets which these model trained with them. STSb is a selection of the English datasets used in the STS tasks organized in the context of SemEval between 2012 and 2017.[21] Also, SNLI is a collection of 570k human-written English sentence pairs which manually labeled for balanced classification.[22] The performance column provides the performance of models on SNLI benchmark using spearman correlation[1]. The last column of the table is speed. Its value is the speed of the embedding process. The values are the number of processed sentences per second on a V100 GPU for each model.

Table 5. Summary of pre-trained models in SentenceTransformer library[23]

Model name	Base Model	Training Data	Performance	Speed
stsb-roberta-large	roberta-large	NLI+STSb	86,39	830
stsb-roberta-base	roberta-base	NLI+STSb	85,44	2300
stsb-bert-large	bert-large-uncased	NLI+STSb	85,29	830
stsb-distilbert-base	distilbert-base-uncased	NLI+STSb	85,16	4000
paraphrase-distilroberta-base-v1	distilroberta-base	Paraphrase Data	81,81	4000
allenai-specter	allenai-specter	SciDocs	N/A	N/A

In total, we selected six pre-trained models from the library. Three models were chosen because of their performance in Table 5. The model distilbert-base-nli-stsb-quora-ranking is optimized to find duplicate questions, which can also be helpful for such a task. The model, paraphrase-distilroberta-base-v1, was chosen because it had a different

[18] https://pypi.org/project/strsimpy/

[19] https://paperswithcode.com/sota/semantic-textual-similarity-on-sts-benchmark

[20] https://nlp.stanford.edu/projects/snli/

[21] https://huggingface.co/datasets/stsb$_multi_mt$

[22] https://nlp.stanford.edu/projects/snli/

perspective to build this model and may enhance the model's performance. Its purpose was to paraphrase mining. It is a task that aims to find texts with similar meanings in a large corpus. This model works fine in a large corpus, and because descriptions in reports can get large, the model should be able to handle long texts well. Table 6 summarizes the characteristics of the selected pre-trained models. Some characteristics made these chosen models good candidates for this study.

Table 6. Characteristics of the chosen models that makes them suitable for this study

Model	Characteristics
stsb-bert-large	Optimized for Semantic Textual Similarity (STS) Trained on SNLI+MultiNLI Fine-tuned on the STS benchmark train set
stsb-roberta-base	Optimized for Semantic Textual Similarity (STS) Trained on SNLI+MultiNLI Fine-tuned on the STS benchmark train set
stsb-roberta-large	Optimized for Semantic Textual Similarity (STS) Trained on SNLI+MultiNLI Fine-tuned on the STS benchmark train set
distilbert-base-nli-stsb-quora-ranking	Extended of distilbert-base-nli-stsb-mean-tokens model and trained it with OnlineContrastiveLoss and with MultipleNegativesRankingLoss on the Quora Duplicate questions dataset
paraphrase-distilroberta-base-v1	Trained on Millions of paraphrase examples. Create extremely good results for various similarity and retrieval tasks
allenai-specter	Document-level Representation Learning using Citation-informed Transformers

What should be noted here is that there is no need to train any model as models are already trained. So, in this case, we do not need training data, and we should use models to build embedding vectors on the test data and then measure the performance.

Therefore, by choosing the pre-trained transformer models, we can use them for embedding. In seection 4.1, we made three versions of datasets. We pass test data through all pre-trained models and evaluate them by their embedded vectors for each project for each dataset.

In summary, we have selected six pre-trained models. Also, we made three datasets which means, we will have 18 embedding transformer models for each project. As we have four different projects, we achieve 72 different embedding transformer models. The result of the models is available in chapter 5.

4.2.3 Trained models

Although pre-trained models can perform well, another way to enhance the performance is to train a model as it includes the data from the train set so the network can understand the semantics of the text better. It is because the domain of test data is the same as training. Hopefully, the SentenceTransformer framework provides this option as well effortlessly.

Figure 5 shows the steps required to train transformer models with data. After training the transformer model, by putting a title and description as an input, it creates embeddings and pass it through a pooling model; we have an embedding output vector related to its embedding.

For training a transformer model, we choose RoBERTa as the base transformer structure for embedding. As shown in Figure 5.

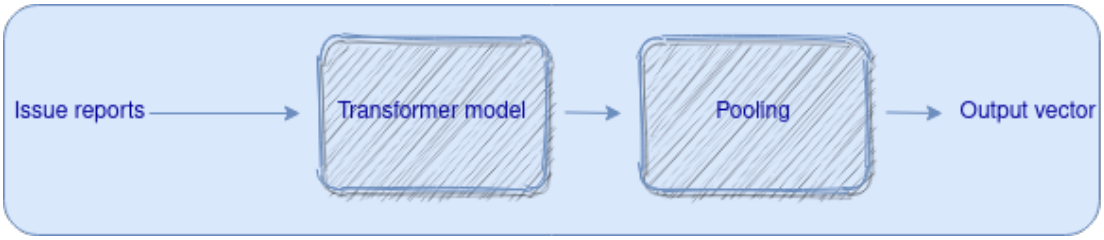

Figure 6. Embedding box part for training in Figure 5

For this model, we choose **RoBERTa** as a transformer structure for embedding. As shown in the Figure 6, there is a pooling step. It is a filter that is applied to the output of the previous layer's maps. The size of the pooling or filter is smaller than the size of the map. For our training, we use **mean pool** and the max_seq_length set as 256, after

training the models for each dataset with the training data. We will pass the test data to the model for evaluation. Fitting a model using RoBERTa requires to select a loss function. We used cosine similarity as the loss function for our experiments since text similarity based on cosine showed the best performance during the baseline calculation. Table 9 shows a comparison of the different text similarity metrics. RoBERTa also requires an input made of sets of sentences as input, where each set has to be labeled according to its similarity (see Listing 4). Thus, we set the labels using the original information in the dataset.
By setting the structure of our model, the input issue reports can be passed to the network for embedding. We have three versions of datasets, and for each dataset, we train a different model using the same setting.

The baseline architecture that we will build has the same structure as visualized in Figure 5. The difference is the embedding model. Instead of transformer models used in previous sections, we use a well-known embedding model Word2Vec[20]. As Word2Vec is pre-trained, we need to pass the test data to the model for the result. After the embedding process, it will pass through text similarity metrics which will be discussed in 4.4.1.

4.3 Step 3: Optimizing hyperparameters

In this section, we explain the parameter optimization that was conducted to increase different models' performance.

4.3.1 Pre-trained transformer models

Table 7 shows the list of available parameters for pre-trained transformer models. Parameters are available to set.

There is not much flexibility for hyperparameter optimization when using pre-trained models; we only adjust the parameter batch_size to make the process quicker. We doubled the size of batch size. The default value for batch size is 32, and we set it to 64. We did not set the batch size much larger because too much large batch size may make the network incapable of converging. Other parameters do not involve the network's performance, and we set them as their default values.

4.3.2 Trained models

Table 8 shows the parameters available for optimization when training transformer models.

Among the available parameters, we explore different values for labels. we set the label 1 for semantically equivalent pairs and 0 for other pairs; for the sake of conciseness, we write them as (1,0). The difference between the labels is significant, so it is easier for

Table 7. Available hyperparameters for pre-trained transformer models to set

Parameters	**Explanation**
sentences	the sentences to embed
batch_size	the batch size used for the computation
show_progress_bar	Output a progress bar when encode sentences
output_value	Default sentence_embedding, to get sentence embeddings. Can be set to token_embeddings to get wordpiece token embeddings.
convert_to_numpy	If true, the output is a list of numpy vectors. Else, it is a list of pytorch tensors.
convert_to_tensor	If true, you get one large tensor as return. Overwrites any setting from convert_to_num
device	Which torch.device to use for the computation
normalize_embeddings	If set to true, returned vectors will have length 1. In that case, the faster dot-product (util.dot_score) instead of cosine similarity can be used.

Table 8. Available hyperparameters for trained transformer models to set

Parameters	**Explanation**
train_objectives	Tuples of (DataLoader, LossFunction) Pass more than one for multi-task learning
evaluator	An evaluator (sentence_transformers.evaluation) evaluates the model performance during training on held-out dev data It is used to determine the best model that is saved to disc
epochs	Number of epochs for training
steps_per_epoch	Number of training steps per epoch. If set to None (default), one epoch is equal the DataLoader size from train_objectives
scheduler	Learning rate scheduler. Available schedulers: constantlr, warmupconstant, warmuplinear, warmupcosine, warmupcosinewithhardrestarts
warmup_steps	Behavior depends on the scheduler For WarmupLinear (default), the learning rate is increased from o up to the maximal learning rate. After these many training steps, the learning rate is decreased linearly back to zero.
optimizer_class	Optimizer
optimizer_params	Optimizer parameters
weight_decay	By default, a list of tensors is returned. If convert_to_tensor, a stacked tensor is returned. If convert_to_numpy, a numpy matrix is returned.
evaluation_steps	If > 0, evaluate the model using evaluator after each number of training steps
output_path	Storage path for the model and evaluation files
save_best_model	If true, the best model (according to evaluator) is stored at output_path
max_grad_norm	Used for gradient normalization.
use_amp	Use Automatic Mixed Precision (AMP). Only for Pytorch $>= 1.6.0$
callback	Callback function that is invoked after each evaluation. It must accept the following three parameters in this order: score, epoch, steps
show_progress_bar	If True, output a tqdm progress bar
labels	value set for each data tuples for training

the model to distinguish. It means that the model does not need to make much effort to embed it in the best possible way. To make the model distinguish on complex pairs, it should embed the most critical information in issue reports. We explored different value pairs for labels as candidates. The candidates we explored were (0.8, 0.2) and (0.6, 0.4), and the exploration is done with brute force.
As explained in the previous paragraph, we are making the model more distinguishable in its embedding. To achieve this goal, the model needs more epoch to train itself. We explored the number of epochs as 5 to see how it will change the model. The results will be in chapter 5.

4.4 Step 4: Evaluating results

In this section, we are going to explain the evaluation steps and details. Firstly, we find a text similarity measurement method. Then, we define a metric to provide the results. Steps of the work provided in Figure 7.

4.4.1 Baseline comparison

To evaluate the perfomarnce of the models, we first create a baseline. The baseline has mainly two purposes:

- To compare the performance of the transformer models against the performance of the baseline
- To determine the best text similarity measure that can be used as a loss function during the training of transformers

To create the baseline, we use a simple embedding model instead of a transformer. The algorithm used is the standard algorithm Word2Vec. Then, we use different text similarity algorithms on the test data for each project to see which text similarity method works the best. Among the results, the one that comes the best will be the text-similarity method to use during the training of the transformers (see Section 4.2.3). For its implementation, we used NumPy as a primary package for scientific computation in Python. It helped the computing process by fast operations on many tasks such as Word2Vec.

The text similarity measures considered during the baseline are Cosine similarity, Longest common subsequence similarity, and Monge-Elkan for token-based, sequence-based, and hybrid methods. These metrics were chosen based on a categorization provided by strsimpy library.

For each category of text similarity metrics, we chose an algorithm that supports normalized. The algorithm's complexity is lower according to the strsimpy library to compare one of each category of methods with the other methods.

4.4.2 Evaluating the models

To evaluate, we should define a performance metric such as accuracy, recall, or precision. We use **recall rate** in this study. Recall rate(RR) is defined as the accuracy of the top list of scores that semantically equivalent detector found for an actual top list. [27] The size of the top list is defined as the parameter of recall rate and is written as RR-N, which N is the recall rate parameter. Recall rate is used as the main performance metric in this study because it is used in similar works [27][10][9] and helps us to compare the results. The text similarity measures considered during the baseline are Cosine similarity, Longest

common subsequence similarity, and Monge-Elkan for token-based, sequence-based, and hybrid methods.

By having the embedding achieved by previous steps, we are going through the test file. In the test file, we have a list of records in which, for an issue report, there is a list of semantically equivalent issue reports. For each issue report in the test file, we iterate in the whole issue report list and find the text-similarity score for pairs consisting of the targeted issue report. Then, we find a top list of semantically equivalent issue reports. The size of the list depends on the parameter. Then, we calculate the recall rate.

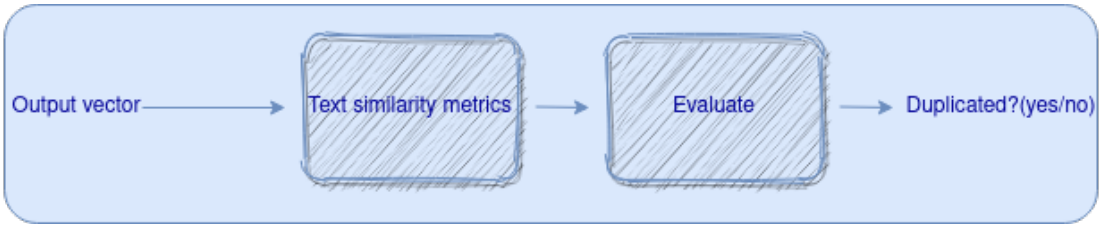

Figure 7. Text similarity method box part in Figure 5

In this study, we use RR-1, RR-5, and RR-20. In the next chapter, we will provide the performance of each model explained in previous steps and compare their results.

5 Results

5.1 Baseline result

Table 9 shows the result of baseline calculation. Also, we set our baseline as the model used with Word2Vec as embedding and the chosen text similarity metrics for each project.

The results range from 0.023 to 0.113 for RR-1, 0.076 to 0.177 for RR-5, and 0.101 to 0.213 for RR-20. The best result is given by cosine similarity with values between 0.1 and 0.213. As it can be seen, the performance of these simple methods is not high. The reason is the basis of their comparison between the two texts. They do not include the semantic of the sentence. They worked with words or even vowels that are not important because two different reports may point to the same issue with a different sequence of words. These algorithms may not understand their similarity.

In Eclipse, Firefox, and JDT projects, cosine similarity performed the best. Cosine similarity followed by Monge-Elkan. Cosine similarity performed slightly better than Monge-Elkan in these projects. Other methods had a considerable distance from all other methods, which can be seen in Table 9.
In the JDT project, Monge-Elkan was followed by cosine similarity. Monge-Elkan performed slightly better than cosine similarity as provided in Table 9.

Table 9. Performance of different text similarity score on different projects using Word2Vec embedding

Method	Approach	Project	RR-1	RR-5	RR-20
Cosine similarity	**Token-based**	**Mozilla Core**	0.112	0.166	0.194
		Firefox	0.1	0.156	0.204
		JDT	0.104	0.163	0.198
		Eclipse Platform	0.113	**0.177**	**0.213**
Longest common subsequence similarity	**Sequence-based**	**Mozilla Core**	0.038	0.076	0.101
		Firefox	0.035	0.088	0.12
		JDT	0.023	0.102	0.124
		Eclipse Platform	0.043	0.093	0.131
Monge-Elkan	**Hybrid**	**Mozilla Core**	0.121	0.172	0.197
		Firefox	0.12	0.152	0.193
		JDT	**0.12**	0.152	0.193
		Eclipse Platform	0.112	0.172	0.205

5.2 RQ1: Pre-trained transformer models

The next step is to see the result of the pre-trained models trained with a large different dataset to detect duplication semantically. By implementing them, we reached the results that are provided in Table 10. The results range from 0.09 to 0.139 for RR-1, 0.096 to 0.214 for RR-5, and 0.1 to 0.277 for RR-20.

Table 10. Performance of pre-trained transformer models on the projects

Model	Project	Dataset	RR-1	RR-5	RR-20
stsb-bert-large	**Mozilla Core**	**D1**	0.107	0.161	0.225
		D2	0.109	0.169	0.231
		D3	0.101	0.153	0.194
	Firefox	**D1**	0.112	0.185	0.233
		D2	0.126	0.202	0.267
		D3	0.105	0.17	0.224
	JDT	**D1**	0.106	0.162	0.212
		D2	0.109	0.173	0.23
		D3	0.092	0.153	0.188
	Eclipse Platform	**D1**	0.115	0.185	0.243
		D2	0.116	0.196	0.256
		D3	0.111	0.16	0.221
stsb-roberta-base	**Mozilla Core**	**D1**	0.109	0.173	0.224
		D2	0.121	0.189	0.238
		D3	0.106	0.163	0.209
	Firefox	**D1**	0.103	0.173	0.241
		D2	0.105	0.193	0.249
		D3	0.09	0.153	0.217
	JDT	**D1**	0.104	0.173	0.223
		D2	0.113	0.18	0.247
		D3	0.09	0.157	0.223
	Eclipse Platform	**D1**	0.113	0.177	0.233
		D2	0.118	0.182	0.263
		D3	0.104	0.176	0.226
stsb-roberta-large	**Mozilla Core**	**D1**	0.112	0.186	0.231
		D2	0.119	0.197	0.251
		D3	0.102	0.163	0.216
	Firefox	**D1**	0.107	0.182	0.246
		D2	0.123	0.2	0.268
		D3	0.099	0.175	0.243
	JDT	**D1**	0.108	0.106	0.114
		D2	0.121	0.114	0.12

		D3	0.092	0.096	0.1
	Eclipse Platform	**D1**	0.118	0.193	0.247
		D2	0.13	**0.214**	0.277
		D3	0.101	0.17	0.216
distilbert-base-nli-stsb-quora-ranking	**Mozilla Core**	**D1**	0.103	0.152	0.202
		D2	0.113	0.157	0.215
		D3	0.093	0.149	0.191
	Firefox	**D1**	0.106	0.154	0.224
		D2	0.118	0.157	0.251
		D3	0.095	0.14	0.205
	JDT	**D1**	0.105	0.171	0.226
		D2	0.116	0.194	0.254
		D3	0.103	0.17	0.194
	Eclipse Platform	**D1**	0.109	0.169	0.214
		D2	0.118	0.19	0.229
		D3	0.108	0.156	0.185
paraphrase-distilroberta-base-v1	**Mozilla Core**	**D1**	0.115	0.186	0.238
		D2	0.122	0.192	0.256
		D3	0.102	0.176	0.221
	Firefox	**D1**	0.117	0.192	0.251
		D2	0.125	0.206	0.263
		D3	0.106	0.181	0.222
	JDT	**D1**	0.111	0.184	0.236
		D2	0.122	0.186	0.26
		D3	0.095	0.168	0.228
	Eclipse Platform	**D1**	0.124	0.194	0.256
		D2	**0.139**	0.198	**0.277**
		D3	0.109	0.166	0.24
allenai-specter	**Mozilla Core**	**D1**	0.106	0.153	0.198
		D2	0.11	0.167	0.216
		D3	0.104	0.145	0.198
	Firefox	**D1**	0.104	0.156	0.221
		D2	0.113	0.175	0.244
		D3	0.09	0.139	0.213
	JDT	**D1**	0.106	0.177	0.231
		D2	0.114	0.203	0.255
		D3	0.103	0.158	0.221
	Eclipse Platform	**D1**	0.107	0.171	0.211
		D2	0.112	0.171	0.239

		D3	0.096	0.156	0.208

5.3 RQ2: Trained transformer models

The result for trained transformer models shown in Table 11. Moreover, we explained hyperparameter optimization in the previous chapter. To find the best parameters to tune the model, we should compare the results for the model used fine-tuned parameter. In Table 11, the result for different trained transformer models is provided.

5.4 RQ3: Pre-trained and trained transformer models on datasets by keeping parenthesis or links

This section presents the result for pre-trained and trained transformer models provided in Table 11 for all projects. The results range from 0.108 to 0.159 for RR-1, 0.192 to 0.237 for RR-5, and 0.243 to 0.306 for RR-20.

Table 11. Performance of trained models on the projects

Model	Dataset	Project	RR-1	RR-5	RR-20
Trained model - label (1,0) - Default parameters	**D1**	**Mozilla Core**	0.114	0.207	0.265
		Firefox	0.116	0.196	0.263
		JDT	0.136	0.192	0.243
		Eclipse Platform	0.155	0.203	0.273
	D2	**Mozilla Core**	0.117	0.215	0.285
		Firefox	0.119	0.214	0.303
		JDT	0.148	0.219	0.275
		Eclipse Platform	0.155	0.234	0.288
	D3	**Mozilla Core**	0.110	0.197	0.261
		Firefox	0.105	0.202	0.286
		JDT	0.137	0.208	0.237
		Eclipse Platform	0.128	0.220	0.268
Trained model - label (0.8,0.2)	**D1**	**Mozilla Core**	0.117	0.212	0.284
		Firefox	0.118	0.213	0.298
		JDT	0.153	0.224	0.276
		Eclipse Platform	0.143	0.212	0.285
	D2	**Mozilla Core**	0.119	0.217	0.288
		Firefox	0.122	0.216	0.306
		JDT	0.155	0.226	0.279
		Eclipse Platform	0.159	0.237	0.291
	D3	**Mozilla Core**	0.113	0.209	0.267
		Firefox	0.108	0.209	0.291
		JDT	0.142	0.218	0.245
		Eclipse Platform	0.133	0.223	0.274
Trained model - label (0.6,0.4)	**D1**	**Mozilla Core**	0.116	0.209	0.281
		Firefox	0.118	0.209	0.293
		JDT	0.153	0.219	0.273
		Eclipse Platform	0.136	0.208	0.277
	D2	**Mozilla Core**	0.114	0.215	0.281
		Firefox	0.118	0.208	0.297
		JDT	0.149	0.222	0.272
		Eclipse Platform	0.153	0.232	0.286
	D3	**Mozilla Core**	0.112	0.206	0.263
		Firefox	0.105	0.205	0.288
		JDT	0.14	0.217	0.242
		Eclipse Platform	0.129	0.216	0.271

5.5 Comparison with the related work

We need to understand how other works have been done. In the following, we compared the evaluation of recent works on such datasets with different methods. Overall, our result did not outstand the state-of-the-art result.

Table 12. Result of the state-of-the-art related work on Bugrepo [9]

	Project					
	Firefox Project			**Open Office Project**		
Approach	**RR-1**	**RR-5**	**RR-20**	**RR-1**	**RR-5**	**RR-20**
BM25F	0.142	0.256	0.313	0.105	0.239	0.310
LDA	0.067	0.135	0.267	0.128	0.219	0.389
Sureka et al.	0.221	0.356	0.490	0.213	0.359	0.493
Yang et al.	0.112	0.204	0.301	0.075	0.159	0.277
DWEN	**0.254**	**0.517**	**0.702**	**0.219**	**0.559**	**0.770**

6 Discussion

6.1 RQ1: Pre-trained transformer models

The first research question investigates the performance of pre-trained transformer models when detecting duplicate issue reports. To answer this question, we used several pre-trained transformer models to compare their performance for each project. We used pre-trained transformer models on datasets D1, D2, and D3 to see the performance of each project. It can be seen that for almost all of the models, by increasing the recall rate parameter, the recall rate increases. Compared to baseline models, the recall rate increased considerably, which means that pre-trained models are better than baseline.
In all of the projects, it can be observed that **Paraphrase model** has the best performance. The reason can be that the paraphrase model is designed to work for longer texts.
In Eclipse, Firefox, JDT, and Mozilla projects, **Quora model** worked the worst among them because this model has been trained on specific texts. The text structure of Quora is not like the text of issue reports. Although other models training texts are also different, but for this model, the text is informal.
In the Mozilla project, what is specific for this project is that in recall rate 1, their result is somehow the same, and the difference between the best and the worst is not much considerable. This means that finding the best match was hard for all of the projects.

In the Firefox project, the growth of **BERT model** in each parameter is not consistent, and its rank among other models changes in each recall rate becomes better and worse. This behavior is in Table 10.

In JDT, the recall rate decreased by increasing recall rate from 1 to 5. **RoBERTa model** has an unexpected result which made it the worst model for this work with recall rate parameters 5 and 20. A model that works very good other than **paraphrase model** is **allenai-specter**. The result can be because the model is trained to identify similar titles and abstracts of academic papers. The length of title and abstract of academic papers is near the size of the issue reports' title and description. This can be the reason why this model worked well for this purpose. The data is in Table 10.
Overall, the performance of pre-trained models are better than the baseline. This means that pre-trained models worked better than Word2Vec as an embedding method. However, compared to the state-of-the-art result which shown in Table 12 the result was worse.

6.2 RQ2: Trained transformer models

The first research question investigates the performance of training transformer models when detecting duplicate issue reports. To answer this question, we trained several transformer models and tune them to improve the performance. Compared to the previous implementation result, the performance of the trained model increased considerably, which means that they worked better than pre-trained models. This is evident because

they trained on the dataset related to this work, not on a general or different dataset that trained for other text structures, such as the formality, the purpose of the texts. The discussion about the parenthesis and links are provided in the latter sections.
For the Eclipse project, a surprising observation can be because fine-tuning made the work worse for recall rate 1. An important point that should be noted to make this question answered is that fine-tuning, in this case, made the performance better for higher parameters, although it made the work worse for this one. The data is in Table 11.
In Firefox, JDT, and Mozilla models, the difference between models using one parameter is not huge but still countable. They improved by fine-tuning and in all of the recall rate parameters. This can be checked in Table 11.
Overall, the performance of trained models was better than pre-trained models and the baseline for all of the datasets in all of the projects. However, the result did not excess the state-of-the-art result which shown in Table 12.

6.3 RQ3: How different data cleaning steps change the performance?

The first research question investigates effect of different cleaning steps on performance. To answer this question, we created three different versions of dataset of each project. Then, we compare the results of models based on all versions of dataset.

6.3.1 How keeping parenthesis change the performance?

According to Table 11 in all of the trained ones and most previous models, we can see a similar pattern. By considering the difference between the performance of models using two different versions of the dataset, we can say that using D2 helped the performance.

We think that the reason is because of the format of the texts. Parenthesis and brackets are used in two cases.

- First of the description that used to include a code for the issue report
- In the middle of the report to include some more explanation

Having these in mind can help the performance in two ways. First, the parenthesis for the issue report's code distinguishes title and description, which can help the model understand the text better. Moreover, by explaining a technical problem, the actual semantic of a sentence can be revealed for the model more conveniently.
Overall, the performance after keeping links in the dataset got better than working with the version of the dataset which do not keep the parenthesis. This result can be seen in all of the projects in Table 11.

6.3.2 How keeping links change the performance?

According to Table 11, by keeping the links and replacing their dots with space, the performance did not change much. The text of the links is read as words in the machine, and because their words do not contribute to the semantics of the text, they are just making some noises that hurt the performance.
Overall, the performance after keeping links in the dataset did not differed much. In some cases, there was an improvement and some cases the performance got worse without any specific pattern. This means that the difference between the results are noise and there is no specific relation between keeping the links and the performance of transformer models.

6.4 Limitations

There are some limitations in this study that limited the result.
We only used one model structure. By changing the model structure used in the network, the performance could be enhanced.
Moreover, we had different possible ways to clean the data. There were several steps for cleaning the data, but we explored only two steps of cleaning. By exploring different steps of cleaning data, the performance may increase.
Furthermore, the dataset included different features, but we utilized the title and description of the dataset. Taking other features into account may help the performance.
In addition, We worked on 4 projects, but the result cannot be generalized for different projects.
Lastly, it is good to analyze the semantics of the networks manually to determine in which situation each network gets the result wrong. By studying this, we may find a solution to fix some types of mistakes in the network and increase the performance.

6.5 Future works

We think some improvements can be made in future works. In this dataset, we only used title and description. Nevertheless, some other works can be added. For instance, we think that partitioning data into chunks of data by their dates may be helpful. At different times, some different issues may be posted into the repository management. Also, there is another option which is to categorize the data by components column. About the components column, it is needed to point out that what this column's information is. Each issue is categorized into different components. Some of them are about the **bugs** found in the program, some other ones are related to the **suggestions** that users give to developers to add as a feature to the program. These differences can be categorized as various components. By categorizing issue reports by the components, it is easier to find semantically equivalent.

7 Conclusion

This study has shown how transformer models can be used to implement a system of semantically equivalent issue reports. Most previous works on this dataset have implemented primary machine learning techniques and tried deep learning techniques for word embedding in some recent works. In this study, we used different transformer models to perform word embedding and used their embedding to detect whether two issue reports are semantically equivalent.
By evaluating the model in terms of recall rate and comparing it with a baseline, we have shown transformer models excess the result of baseline. Also, we provided the result of previous works.
Most previous works have also focused on building the model. In this study, we have tried to include different data cleaning steps. We explored to understand whether modifying some steps of data cleaning may influence the performance.
Overall, the models we evaluated have an excellent performance to apply in issue tracking software. The software can help developers find semantically equivalent issue reports to track issue reports more conveniently.

References

[1] *Spearman Rank Correlation Coefficient*, pages 502–505. Springer New York, New York, NY, 2008.

[2] D. D. A. Parikh, O. Täckström and J. Uszkoreit. A decomposable attention model for natural language inference. *Proc. 2016 Conf. Empir. Methods Nat. Lang. Process.*, pages 2249—-2255, 2016.

[3] T. Addair. Duplicate question pair detection with deep learning. 2017.

[4] W. F. Ahsan SN, Ferzund J. Automatic software bug triage system (bts) based on latent semantic indexing and support vector machine. *Fourth International Conference on Software Engineering Advances IEEE*, pages 216–221, 2009.

[5] T. Akilan, D. Shah, N. Patel, and R. Mehta. Fast detection of duplicate bug reports using lda-based topic modeling and classification. *2020 IEEE International Conference on Systems, Man, and Cybernetics (SMC)*, 2020.

[6] A. Alipour, A. Hindle, and E. Stroulia. 2013.

[7] N. Ansari and R. Sharma. Identifying semantically duplicate questions using data science approach: A quora case study, 2020.

[8] D. Bogdanova, C. D. Santos, L. Barbosa, and B. Zadrozny. Detecting semantically equivalent questions in online user forums. *Proceedings of the Nineteenth Conference on Computational Natural Language Learning*, 2015.

[9] A. Budhiraja, K. Dutta, R. Reddy, and M. Shrivastava. Dwen. *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*, 2018.

[10] A. Budhiraja, R. Reddy, and M. Shrivastava. Lwe. *Proceedings of the 40th International Conference on Software Engineering: Companion Proceeedings*, 2018.

[11] A. M. Cherinet and E. Scott. Recommending issue reports to developers using machine learning, 2019.

[12] L. B. D. Bogdanova, C. dos Santos and B. Zadrozny. Detecting semantically equivalent questions in online user forums. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, pages 123–131, 2017.

[13] F. Gers. Long short-term memory in recurrent neural networks. 2001.

[14] C. R. Harris, K. J. Millman, S. J. van der Walt, R. Gommers, P. Virtanen, D. Cournapeau, E. Wieser, J. Taylor, S. Berg, N. J. Smith, R. Kern, M. Picus, S. Hoyer, M. H. van Kerkwijk, M. Brett, A. Haldane, J. F. del Río, M. Wiebe, P. Peterson, P. Gérard-Marchant, K. Sheppard, T. Reddy, W. Weckesser, H. Abbasi, C. Gohlke, and T. E. Oliphant. Array programming with NumPy. *Nature*, 585(7825):357–362, Sept. 2020.

[15] R. S. J. Pennington and C. Manning. Glove: Global vectors for word representation. pages 1532–1543, 2014.

[16] J. O. JOSEPHSEN. Similarity measures for text document clustering. *Nord. Med.*, 56(37):1335–1339, 1956.

[17] Y. Kim. Convolutional neural networks for sentence classification. *Proceedings of the 2014 Conference on Empirical Methods in Natural Language Processing (EMNLP)*, 2014.

[18] Y. B. Klir GJ. Fuzzy sets and fuzzy logic: theory and applications. *Upper Saddle River*, 1995.

[19] A. R. D. Mani S, Sankaran A. Exploring the effectiveness of deep learning for bug triaging. *Proceedings of the ACM India Joint International Conference on Data Science and Management of Data ACM*, pages 171–179, 2019.

[20] T. Mikolov, K. Chen, G. Corrado, and J. Dean. Efficient estimation of word representations in vector space, 2013.

[21] C. D. Murphy G. Automatic bug triage using text categorization. *Proceedings of the Sixteenth International Conference on Software Engineering Knowledge Engineering*, 2004.

[22] F. J. Nasim S, Razzaq S. Automated change request triage using alpha frequency matrix. *Frontiers of Information Technology IEEE*, pages 298–302, 2011.

[23] M. Nicosia and A. Moschitti. Accurate sentence matching with hybrid siamese networks. *Annalen der Physik*, pages 2235–2238, 2017.

[24] N. Reimers and I. Gurevych. Sentence-bert: Sentence embeddings using siamese bert-networks. In *Proceedings of the 2019 Conference on Empirical Methods in Natural Language Processing*. Association for Computational Linguistics, 11 2019.

[25] S. Robertson. Understanding inverse document frequency: On theoretical arguments for. *J. Doc.*, 2004.

[26] D. Rothman. *Transformers for natural language processing: build innovative deep neural network architectures for NLP with Python, PyTorch, TensorFlow, BERT, RoBERTa, and more*. Packt Publishing, 2021.

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

[28] A. R. R. G. C. D. M. S. R. Bowman, J. Gauthier and C. Potts. A fast unified model for parsing and sentence understanding. *Proc. 54th Annu. Meet. Assoc. Comput. Linguist. (Volume 1 Long Pap.*, 1:1466–1477, 2016.

[29] N. D. S. Viswanathan and A. Simon. Advances in big data and cloud computing. *Springer Singapore*, 750, 2019.

[30] C. D. Santos, L. Barbosa, D. Bogdanova, and B. Zadrozny. Learning hybrid representations to retrieve semantically equivalent questions. *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 2: Short Papers)*, 2015.

[31] C. Sun, D. Lo, X. Wang, J. Jiang, and S.-C. Khoo. A discriminative model approach for accurate duplicate bug report retrieval. *Proceedings of the 32nd ACM/IEEE International Conference on Software Engineering - ICSE '10*, 2010.

[32] A. Sureka and P. Jalote. Detecting duplicate bug report using character n-gram-based features. In *2010 Asia Pacific Software Engineering Conference*, pages 366–374, 2010.

[33] A.-K. J. N. T. Tamrawi A, Nguyen TT. Fuzzy set-based automatic bug triaging (nier track). *Proceedings of the 33rd International Conference on Software Engineering*, pages 884–887, 2011.

[34] A. Tung and E. Xu. Determining entailment of questions in the quora dataset. *Nord. Med.*, 56(37):1–8, 2017.

[35] e. a. Vaswani, Ashish. Attention is all you need. 2017.

[36] S. S. Y. Homma and C. Yeh. Detecting duplicate questions with deep learning. *30th Conf. Neural Inf. Process. Syst. (NIPS 2016)*, 2016.

[37] X. Yang, D. Lo, X. Xia, L. Bao, and J. Sun. Combining word embedding with information retrieval to recommend similar bug reports. *2016 IEEE 27th International Symposium on Software Reliability Engineering (ISSRE)*, 2016.

[38] W. H. Z. Wang and R. Florian. Bilateral multi-perspective matching for natural language sentences. *IJCAI International Joint Conference on Artificial Intelligence*, 2017.

Appendix

Codes

The codes are at `https://github.com/behradmoeini/find-duplicate-report-issues`

Code samples for pre-trained models

In Listing 1, texts will be encoded to vectors. As we do not have a training step, this encoding is the longest process time.

```
texts  = [...] #a list consist of text of all (title + description)s
#Compute embeddings for all papers
corpus_embeddings = model.encode(texts, convert_to_tensor=True)
```

Listing 1. Encoding issue reports

By running previous codes, in Listing 2, a function is built for querying for new texts. What happens here is to build a list of best matches according to the input texts. By so, we can build this function to list the texts with their scores.

```
def search(text):
  query_embedding = model.encode(text, convert_to_tensor=True)

  search_hits = util.semantic_search(query_embedding,
   corpus_embeddings)
  search_hits = search_hits[0]  #Get the hits for the first query


  result = []
  for hit in search_hits:
    related_text = texts[hit['corpus_id']]
    result.append(related_text['title'], hit['score'])
  return result
```

Listing 2. Finding similar issue reports function

Moreover, what should be noted here is that in this case, there was no need for any text similarity method. However, it is because it is a pre-trained model for academic papers, and the developers have chosen the text-similarity method for it to work in the best way for such a purpose.

The following code can be used for the algorithms: **stsb-bert-large**, **stsb-bert-base**, **stsb-roberta-large**, **distilbert-base-nli-stsb-quora-ranking**, and **paraphrase-distilroberta-base-v1**

The difference between the code below and previous one is that this one used an external text similarity scoring method, which is cosine similarity.

```
from sentence_transformers import SentenceTransformer, util
```

```
model = SentenceTransformer('model_name')

texts  = [...] #a list consist of text of all (title + description)s

#Compute embedding for both lists
embeddings1 = model.encode(sentences1, convert_to_tensor=True)
embeddings2 = model.encode(sentences2, convert_to_tensor=True)

#Compute cosine-similarits
cosine_scores = util.pytorch_cos_sim(embeddings1, embeddings2)

#Output the pairs with their score
for i in range(len(sentences1)):
    print("{} \t\t {} \t\t Score: {:.4f}".format(sentences1[i],
    sentences2[i], cosine_scores[i][i]))
```

Listing 3. Embedding and evaluating

Code samples for the trained model

```
from sentence_transformers import SentenceTransformer, InputExample,
    losses
from torch.utils.data import DataLoader

#Define the model. Either from scratch of by loading a pre-trained
    model
model = SentenceTransformer('distilbert-base-nli-mean-tokens')

#Define your train examples. You need more than just two examples...
train_examples = [InputExample(texts=['My first sentence', 'My second
     sentence'], label=1),
    InputExample(texts=['Another pair', 'Unrelated sentence'], label
    =0)]

#Define your train dataset, the dataloader and the train loss
train_dataloader = DataLoader(train_examples, shuffle=True,
    batch_size=16)
train_loss = losses.CosineSimilarityLoss(model)

#Tune the model
model.fit(train_objectives=[(train_dataloader, train_loss)], epochs
    =10, warmup_steps=100)
```

Listing 4. Training model

II. Licence

Non-exclusive licence to reproduce thesis and make thesis public

I, **Behrad Moeini**,
*(*author's name)

1. herewith grant the University of Tartu a free permit (non-exclusive licence) to

   reproduce, for the purpose of preservation, including for adding to the DSpace digital archives until the expiry of the term of copyright,

   Detecting semantically equivalent issue reports using transformer models,
   *(*title of thesis)

   supervised by Ezequiel Scott.
   *(*supervisor's name)

2. I grant the University of Tartu a permit to make the work specified in p. 1 available to the public via the web environment of the University of Tartu, including via the DSpace digital archives, under the Creative Commons licence CC BY NC ND 3.0, which allows, by giving appropriate credit to the author, to reproduce, distribute the work and communicate it to the public, and prohibits the creation of derivative works and any commercial use of the work until the expiry of the term of copyright.

3. I am aware of the fact that the author retains the rights specified in p. 1 and 2.

4. I certify that granting the non-exclusive licence does not infringe other persons' intellectual property rights or rights arising from the personal data protection legislation.

Behrad Moeini
14/05/2021