Detecting Duplicate Effort in GitHub Contributions

James Galbraith and Des Greer[a]
School of Electronics, Electrical Engineering and Computer Science, Queen's University Belfast, Belfast, BT71NN, U.K.

Keywords: GitHub, Duplicate Detection, Issues, Pull Requests.

Abstract: The pull-based development model allows collaborators to develop and propose changes to a codebase. However, pull requests can often offer duplicate functionality and therefore duplicate effort. Users can also request changes via issues, the text of which could provide clues, useful in determining duplicate pull requests. This research investigates combining pull requests with issues with a view to better detecting duplicate pull requests. The paper reviews existing related work and then extends this by investigating the use of natural language processing (NLP) on combined issues and pull requests in order to detect duplicates. Using data taken from 15 popular GitHub repositories, an NLP model was trained to predict duplicates by comparing the title and description of issues and pull requests. An evaluation of this model shows that duplicates can be detected with an accuracy of 93.9% and recall rate of 90.5%, while an exploratory study shows that the volume of duplicates detected can be increased dramatically by combining issues and pull requests into a single dataset. These results show a significant improvement on previous studies and demonstrate the value in detecting duplicates from issues and pull requests combined.

1 INTRODUCTION

The introduction of the pull-based development model (Gousios et al., 2014; Gousios and Zaidman, 2014) in GitHub has facilitated the development of open-source projects through the contributions of community developers (Jiang et al., 2017) and has fundamentally changed the way in which projects evolve. The open-source model allows anyone to copy and redistribute the software solution, while the pull-based development model allows anyone to contribute changes to a project. The community developers who provide these changes need not have any particular relationship to, or experience with, the project in question. These developers contribute to projects for a number of reasons (Jiang et al., 2017; McClean et al., 2021). When someone would like to contribute a new feature or bug fix to a project, they do so by forking the project and making the changes locally. These changes are then submitted in the form of a pull request to be reviewed and merged. This process of review is carried out by a core team of reviewers who decide to accept or reject the changes. Each pull request comes with a title and description detailing what the code submitted in the pull request aims to achieve.

Users of an open-source project can also report bugs and contribute ideas and suggestions by raising an issue. Issues, like pull requests, contain a title and description detailing the bug or feature in question, but issues do not provide any solution in terms of code. One way of thinking about this is that issues may be raised by users of a project to detail bugs that they have found, or new features that they would like to see, and who do not have the technical capability, or will, to carry out such work themselves. Pull requests, by contrast, may provide the solution to an issue, but they may also provide a solution to a bug or a feature request that has not already been raised as an issue. Simply, pull requests contain code changes, and issues do not.

Issues are used in open-source software hosting platforms to track bug reports, and propose new features. They also provide a medium through which users and maintainers of a product can communicate, provide feedback, and track the progress of work. Pull requests, by contrast, are used to manage the merging of new code into a codebase. Pull requests display the differences between the source code and the new code being proposed, and they allow reviewers to provide feedback on code changes.

As the number of users using an open-source project increases, so, too, the volume of issues and pull requests being submitted often increase. One

[a] https://orcid.org/0000-0001-6367-9274

Galbraith, J. and Greer, D.
Detecting Duplicate Effort in GitHub Contributions.
DOI: 10.5220/0013289000003928
In *Proceedings of the 20th International Conference on Evaluation of Novel Approaches to Software Engineering (ENASE 2025)*, pages 520-529
ISBN: 978-989-758-742-9; ISSN: 2184-4895

study from 2016 (Gousios et al., 2016) found that 135,000 repositories hosted in GitHub collectively received more than 600,000 pull requests in a single month. Many GitHub repositories can have thousands of open issues and pull requests at any one time. At the time of writing, Flutter has more than 12,000 open issues[1], while Tensorflow has almost 2,000 open issues[2]. This makes it extremely difficult for users, contributors, and reviewers of a project to keep track of all of the changes that are being made to the project. Thus, due to the open, distributed, and uncoordinated nature of pull-based development (Gousios et al., 2016; Gousios et al., 2015) when used at scale, a challenge presents itself where duplicate work can be carried out in terms of development for pull requests, reporting issues, and also in reviewers reviewing submitted pull requests and issues. This can drain reviewers' resources. Since they have no method of immediately identifying duplicate issues and pull requests, these duplicates must undergo the same review process as all other issues and pull requests. Often, the review process will require some back and forth communication between the reviewer and the reporter to gather more information, so the reviewer is not just wasting time through the code review, but also through their interactions with the reporter. Some work has been done previously on creating a dataset of duplicate pull requests (Yu et al., 2018) and some on automatically detecting duplicate pull requests using machine learning (Li et al., 2017), while Runeson et al. explored the possibility of automatically detecting duplicate defect reports (Runeson et al., 2007) from a database of defect reports used by Sony Ericsson. These papers do not, however, consider the possibility of collating pull requests and issues into a single dataset. The machine learning model created by Li et al. (Li et al., 2017) uses natural language processing to calculate the textual similarity between the title and description of pull requests in order to find duplicates. Issues also contain a title and description serving the same function as those of pull requests, which leaves the question of why these cannot be considered as a single entity when training, implementing, and evaluating a machine learning model. Thus, this paper will seek to answer the following questions:

RQ1. What effect does combining pull requests and issues have on the accuracy of a natural language processing (NLP) model, used to predict duplicates, when compared to pull requests only?

RQ2. When considering open and unresolved issues and pull requests, can we find a greater volume of duplicates than when considering pull requests only?

When considering these research questions, we will adopt the following null hypotheses, respectively:

$H_0$1: & The accuracy of a given NLP model will not *increase* when evaluated on the combined dataset compared to pull requests only.

$H_0$2: & The *volume* of duplicates found when considering issues and pull requests combined will not be significantly *greater* than when considering pull requests only.

To test these hypotheses, we will adopt an approach similar to that of Li et al. (Li et al., 2017). First, a console application will be developed to gather a dataset of known duplicate issues and duplicate pull requests from a number of popular open-source GitHub repositories, making use of GitHub's public API to do so. This application will use regular expressions to search the comments of closed issues and pull requests for strings identifying one issue or pull request as a duplicate of another. This data will then be used to train and evaluate a machine learning model. The model will consider the title and description, and use natural language processing to calculate the textual similarity between the items in our dataset, and thus group these items where they are likely to be duplicates of each other. This model will also then be used to calculate the number of predicted duplicates, from a snapshot of unresolved issues and pull requests, relative to the size of the dataset.

This approach differs from that of Li et al. (Li et al., 2017) in several ways. Our model will use a different regex pattern for finding comments identifying duplicates, to allow for greater variety in the format of such comments, following on from the findings from Li et al. (Li et al., 2018). This program will also consider a wider range of projects. However, critically we will be combining pull requests and issues into a single dataset when training our NLP model. Given the nature of the data collected, it may be useful to explore the differences between pull requests only, and then issues and pull requests combined, when detecting duplicates in public projects and also in evaluating our machine learning model. We will also explore the differences when the model is evaluated on each project individually.

2 RELATED WORK

All the papers reviewed in the following section have been taken from Google Scholar, IEEE Xplore, Science Direct, Springer, and Scopus. The following search strings composed of key words were used to

[1]https://github.com/flutter/flutter/issues

[2]https://github.com/tensorflow/tensorflow/issues

search these sites:
"GitHub" AND "Issues" AND ("Pull requests" OR "Pull-requests")
"GitHub" AND ("Issues" OR "Pull requests" OR "Pull-requests") AND "Duplicate"

This will exclude articles centred around other source code hosting platforms. As an extension to these search strings, articles that have not been peer reviewed will be excluded, along with opinion pieces and articles which have not been published in a known and credible journal or conference paper. This will provide a systematic approach to selecting papers for review, and facilitate a study of the practices associated with pull-based development and open-source software firstly, followed by a review of the work carried out thus far in finding and reducing duplicate contributions.

2.1 Pull-Based Development

Through GitHub's public REST API, given the name and owner of a public repository, researchers can collect vast amounts of data related to the issues, pull requests, comments, contributors, and so on, connected to said repository. This has, in part, enabled studies to be carried out on the processes and methodologies used to manage the continuous development of software solutions, and how human resources, in terms of contributors and integrators, can collaborate to facilitate this development. An early study by Kalliamvakou et al. (Kalliamvakou et al., 2014) investigating the perils of mining GitHub for data found that almost 40% of all pull requests do not appear as merged even though they have been. A more recent study by Wessel et al. (Wessel et al., 2023) to investigate the effects of GitHub Actions on the pull request process found that the adoption of GitHub Actions leads to more comments on accepted pull requests and less comments in rejected pull requests. GitHub Actions is a continuous integration and continuous delivery platform that enables the automation of testing and deployment workflows. From their sample of 5,000 popular GitHub repositories, 1,489 of them used GitHub Actions. These findings may pose the possibility of a bias towards merged pull requests if we analyse the comments on pull requests and issues.

2.2 Duplicate Contributions

Zhang et al. (Zhang et al., 2018) conducted a study in 2018 on competing pull requests, wherein competing pull requests were defined as those that aimed to "change the same piece of code" and were open for overlapping periods of time. In 45 out of the 60 repositories explored, over 31% of pull requests belonged to competing pull requests. In 20 of the repositories, there were more than 100 groups of competing pull requests, each of which were submitted by more than five developers. However, it should be noted that competing pull requests are not strictly the same as duplicate pull requests, although in some cases competing pull requests may also be duplicate pull requests.

Much of the research conducted around duplicate contributions concerns two or more pull requests that are decided to add the same functionality, rather than those which simply change the same articles of code. This often manifests itself as pull requests which fix the same bug, or add the same new piece of functionality. A study by Li et al. (Li et al., 2020) found that duplicate pull requests lead to the use of redundant resources. Another study (Zhou et al., 2019) found that as much as 51% of rejected pull requests in some repositories were rejected due to redundant development (i.e. they were duplicates of other pull requests). Although the mean was only 3.4%, they also found a high variance between the repositories considered. This shows the potential value in exploring our research questions on a project-to-project basis. Another study by Li et al. (Li et al., 2017) explored the possibility of using textual similarity between the titles and descriptions of pull requests to detect duplicates. When this model was evaluated on three GitHub projects, 55.3% - 71.0% of the duplicates were found using a combination of title similarity and description similarity. This evaluation has only used data from three repositories however, which does not allow for the potential discrepancies between repositories in terms of reporting bugs, feature requests, etc. Many open source projects provide contribution guidelines (Li et al., 2022), while GitHub also provides guidelines for contributions. Therefore, the results of this study (Li et al., 2017) cannot account for these differences. Wang et al. proposed an approach (Wang et al., 2019) in which they used nine features of pull requests to train a machine learning model to detect duplicates. Their model was trained on data from 12 GitHub repositories and evaluated on data from 14 different GitHub repositories and demonstrated a recall rate of 88.7%.

A study (Yu et al., 2018) by Yu et al. has suggested a method for collecting a dataset of duplicates consisting of automatic identification followed by manual verification. A program will use regex patterns to search the comments for links between pull requests and extract the links to said pull requests. Each pair of pull requests were then manually verified to remove possible false positives. This paper provides a good

starting point for collecting a dataset of duplicates; however, this manual verification step would appear to be superfluous. The article does not detail the number of false positives found, but given the contents of the data collected, combined with GitHub's template for reporting such data[3], the prediction is that the probability of false positives being detected is significantly small, provided that the program works as expected. As shown, the comments left by contributors follow particular patterns depending on the relationship they are trying to convey (Li et al., 2018), and hence manual verification becomes redundant. This extracted dataset is publicly available and contains duplicate pull requests from 26 popular open-source projects in GitHub.

2.3 Combining Issues and Pull Requests

Again, studies (Yu et al., 2018; Li et al., 2017; Zhang et al., 2018; Li et al., 2020) only consider pull requests that may be duplicates of other pull requests. They do not consider the impact of combining datasets of issues and pull requests, when performing empirical studies and evaluating their models. Some work has been done on matching pull requests to commits, but there are no studies exploring the automatic detection of duplicate issues, or the potential value in detecting duplicates from a combined dataset of issues and pull requests.

When pull requests are submitted to GitHub, they are sometimes submitted as a solution to an already existing issue. In this scenario, the pull request will contain an explicit link to the issue that it is resolving. In the event where two users identify the same bug in a software product, one user raises it as an issue, while the other user forks the project and submits a pull request resolving the bug unaware that an issue has been raised concerning the same bug, then there will be no link between the issue and pull request. By predicting duplicates from a combined dataset of issues and pull requests, the program will be able to identify pull requests and issues which deal with the same bug but are not known to do so. This can be used to *prevent* duplicate pull requests.

These ideas are predicated on the assumption that the title and description of issues serve the same purpose as the title and description of pull requests. This seems like a reasonable assumption to make, and one which we will seek to verify in our empirical study, as part of **RQ2**. However, one potential problem might be how the level of detail presented differs between issues and pull requests. Any user who submits a pull request must have some level of technical ability along with some understanding of the codebase since they are making changes to the code. A user who submits an issue, though, need not necessarily have any level of technical ability or understanding. They only really need to be able to explain the bug they have found with a system, possibly along with steps to recreate the bug, or the feature which they would like added. This poses the possibility that pull requests may contain much more technical jargon, compared to issues which may use simpler language to provide a higher-level overview. If this is the case, where the descriptions of pull requests provide much lower-level detail compared to issues, then this could reduce the effectiveness of any natural language processing model when comparing the text between pull requests and issues, and would advance the case for null hypothesis $\mathbf{H_0 1}$.

[3]https://docs.github.com/en/issues/tracking-your-work-with-issues/marking-issues-or-pull-requests-as-a-duplicate

3 DATA COLLECTION

3.1 Projects Studied

15 open-source projects, hosted in GitHub, have been selected to be studied, and from which we will gather our data. The projects selected are: angular, angular.js, ansible, bootstrap, elasticsearch, electron, flutter, freeCodeCamp, kubernetes, node, opencv, rails, react, rust, and tensorflow. These projects have been selected partly by design. As a starting point, projects with less than 5000 *closed* issues or less than 5000 *closed* pull requests were not considered. This was to ensure that there was sufficient data available, and to maximise the probability of finding duplicate issues and pull requests. Following this, projects with less than 1000 contributors were also not considered. Working off the assumption that a large number of issues and pull requests, coupled with a large number of contributors, makes it difficult for users to keep track of what work has been completed and hence increasing the likelihood that duplicate efforts will be submitted, the minimum contributor limit was hence added. Projects where the spoken language was any other than English were also not considered. Some effort was made to ensure that a range of programming languages were represented in the projects selected to remove the possibility that our machine learning algorithm would place either too much or too little value on language-specific keywords. To this end, 12 programming languages have been represented in the 15 open-source projects chosen, as shown in Table 1.

Some of the projects selected therefore have been

Table 1: Project Characteristics.

Project	Pulls?	Issues?	Language
Angular	y	y	TS
Angular.js	y	n	JS
Ansible	y	y	Python
Bootstrap	y	y	JS, CSS
Elasticsearch	y	y	Java
Electron	y	y	JS, Python, C++, C
Flutter	y	y	Dart, C, C++
FreeCodeCamp	y	y	TS, React
Kubernetes	y	y	Go
Node	y	y	JS, Python, C++, C
OpenCV	y	y	Java, Python, C++
Rails	y	y	Ruby, JS
React	y	y	JS
Rust	y	y	Rust
TensorFlow	y	y	Python, C++

taken from the research previously carried out on this subject (Yu et al., 2018; Li et al., 2017). Beyond that, the other projects were selected at random from a list of the top 100 starred open-source projects in GitHub[4], provided they met the criteria in the preceding paragraph. A breakdown of the numbers of issues, pull requests, and contributors from the projects selected can be seen in Table 2. Note that while some effort was expended to try to maximise the numbers of duplicates found in the selected projects, the data collection method was designed to ensure no false positives were identified, and thus could not guarantee that duplicates would be found in all of the projects. Thus in the case of Angular.js no issues were found.

Table 2: Issue and PR Statistics from studied projects.

Statistic	Mean	Median	Min	Max
Open Issues	2571	1391	271	12165
Closed Issues	27156	21894	7901	79838
Open PRs	336	214	61	820
Closed PRs	36083	32138	7926	77660
Contributors	2889	1907	1249	5540

3.2 Method

In GitHub, there is no mechanism to check for duplicate pieces of work. There is no mechanism, external to an issue/pull request, by which duplicates can be marked. Duplicates must be marked by leaving a comment on the issue or pull request in question noting the issue or pull request which it is a duplicate of, and guidelines have been provided for this[5]. To find duplicates in the projects chosen, we will therefore need to dig through the comments for each issue and pull request and check if the format of the comment matches the format expected for marking a duplicate. GitHub has made available a public REST API which can be used to retrieve data from GitHub projects (Kalliamvakou et al., 2016; Mombach and Valente, 2018). Simply by knowing the name and owner of a repository, we can work through a series of layers to retrieve all its issues and pull requests, and subsequently retrieve all comments for each issue/pull request. When an issue or pull request is marked as a duplicate, it will also be closed, and so we will only be looking at closed issues and pull requests when searching for duplicates.

Since there is no requirement to follow the guidelines on marking duplicates, variations arise in the format of the comments to identify duplicates (Li et al., 2017; Gousios and Zaidman, 2014). As shown in previous research, duplicates are sometimes marked by leaving a comment of the format *"Duplicate of #11111"*. In this context, the number is a unique identifier which can be used to identify the issue or pull request in question. On GitHub's website, this number also contains a hyperlink to said issue or pull request. When this data is retrieved through the REST API the number is lost, and only the hyperlink remains (on the latest version of the API). Therefore, we need not check for comments of the format *"Duplicate of #11111"* as has been done previously. GitHub limits the number of requests which can be made to its REST API by a single user within a specified period of time. As a result of this, we were only able to make 5000 requests to the API in each hour (one hour after the first request is made, the limit would be reset). Given that some of the projects to be studied have many tens of thousands of closed issues and pull requests, it would be impossible to retrieve and process all issues and pull requests for each project in a reasonable time frame. Therefore only the 5000 most recently closed issues and 5000 most recently closed pull requests were considered for each project.

To collect the data needed, a console application was created and hard-coded with the name and owner of the projects selected for study. When the issues and pull requests were retrieved for each project, and the comments for each issue/pull request were retrieved, regex was used to search the comments for messages

[4]https://github.com/EvanLi/Github-Ranking/blob/master/Top100/Top-100-stars.md

[5]https://docs.github.com/en/issues/tracking-your-work-with-issues/marking-issues-or-pull-requests-as-a-duplicate

which match one of the patterns known to be used to mark duplicates. As per previous research (Yu et al., 2018; Li et al., 2017; Li et al., 2018), comments can take several forms including *"duplicate of..."*, *"closed by..."*, and *"addressed in..."*. Taking this into consideration, the following regex pattern has been constructed to find such strings in pieces of text:

(dup|addressed|close|duplicate|duplicated|closed) (w+)?(by|of|in) (http)

A separate regex pattern, following the same format, was then used to extract the URLs of the duplicates from the text. From each URL, the GitHub API was used once again to retrieve the full details of the item, and thus, a pair of duplicates had been identified. In the event where an issue or pull request is a duplicate of multiple other issues/pull requests, the second regex pattern can allow for this to extract multiple URLs, provided the URLs are separated by commas. From a string such as *"Duplicate of https://github.com/..., https://github.com/..., https://github.com/..."*, each of the URLs would be extracted and processed and hence a tuple of duplicates would be recorded. When the console application had run its course and all processing was complete, the resultant data was stored in a database.

To answer the research questions, a natural language processing model must be trained such that given two pieces of text, it will predict them to be either duplicates or non-duplicates. To train such a model, we would need data containing sets of both duplicates and non-duplicates, but the data gathered to this point consisted only of sets of duplicates. To gather sets of non-duplicates, the existing sets of duplicates were simply shuffled to generate sets of non-duplicates. Since there was a possibility, as allowed for by our console application, that sets of duplicates containing more than two items were collected, some effort was required to ensure that the non-duplicates were truly non-duplicate, and not simply an inversion of an existing set of duplicates. The result of all of this was a dataset containing 393 sets of duplicate issues, 312 sets of non-duplicate issues, 510 sets of duplicate pull requests, and 509 sets of non-duplicate pull requests. To ensure a more even balance in the datasets, some pull requests were also taken from the dataset produced by Yu et al. (Yu et al., 2018). These figures are not intended to be reflective of the proportion of duplicates which exist in the projects, either identified (closed) or unidentified (open and unresolved).

4 EXPERIMENTS

In considering the research questions previously laid out, we might summarise them to ask, is it possible to predict duplicates from a combined dataset of pull requests and issues, and if so, is there any value in it. **RQ1** is concerned only with the dataset collected, used as a source of truth, from which we might train and evaluate a machine learning model. **RQ2**, by contrast, concerns itself with how this machine learning model might be deployed and used to explore the extent to which duplicates exist in open-source GitHub projects, and how it's deployment might be used to identify these automatically.

4.1 Setup

Once sufficient data had been collected, a natural language processing model would be trained to identify duplicates from two pieces of text. To do this, the NLP model would convert each piece of text to a sequence of vectors, so that the two sequences could be compared. Following on from previous research (Ma et al., 2019; Muennighoff et al., 2022; Devlin et al., 2018), it was decided that a MiniLM (Wang et al., 2020; Reimers and Gurevych, 2019) based model should be used. A pre-trained model would be selected, and fine-tuned using our own data. The Massive Text Embedding Benchmark (MTEB) leaderboard (Muennighoff et al., 2022) was, in part, used to find a suitable model. Many of the models listed in this leaderboard however would only accept text input up to a few hundred tokens (equivalent to a few hundred words), which would be a problem since the title and descriptions of some of the issues and pull requests in our dataset contained (an estimated) tens of thousands of tokens. Some of the models are able to handle larger pieces of text, however this often came at the cost of unreasonably large and complicated models, which required substantially more computing resources to use. Another way around this issue is to split the text up into chunks, generate a sequence of embeddings for each chunk, and then take an average across all the embeddings to get a single sequence which would be representative of the entire piece of text. To do this, we split the text into chunks of 250 characters, while also ensuring words do not get split across chunks. Thus, by applying these techniques, a single sequence of embeddings could be generated to represent the semantics of each piece of text.

4.2 Predicting Duplicates

To predict duplicates from the data, some measure would be needed to compare each of the two sequences of embeddings and generate a score of similarity between them. As per Li et al. (Li et al., 2017), we will be using a cosine similarity measure (Manning and Schutze, 1999) to do this. This will give a single score between 0 and 1 representing how similar the two pieces of text are, where 1 is identical and 0 is completely unlike. To classify duplicates from this, a threshold therefore needs to be applied whereby two pieces of text whose similarity score meets the threshold are duplicates. Varying duplicate threshold values were tested and used as appropriate.

4.3 Model Evaluation

To explore **RQ1**, the sentence-transformers (Reimers and Gurevych, 2019) model all-MiniLM-L6-v2[6], as in the MTEB leaderboard[7], was selected as a starting point and a base model. This model was fine-tuned on our data using an 80:20 train/test split. The model was fine-tuned on both issues and pull requests. The test data consists then of a dataset containing only pull requests, and a dataset containing both issues and pull requests. In evaluating the model, we will explore the accuracy of the model when varying the threshold value by which duplicates are determined, and compare and contrast the results when doing this for each of the two datasets. To disprove the null hypothesis, we would expect the accuracy to increase for the combined dataset. This experiment will then be repeated on a per-project basis to investigate how the results vary between projects and whether the techniques used are equally applicable to all projects.

4.4 Empirical Study

To answer **RQ2** we will take a snapshot of all the issues and pull requests open and unresolved in the projects studied, and use our fine-tuned model to predict sets of duplicates from among these, and investigate what proportion of pull requests are (predicted) duplicates, in comparison with the proportion of (predicted) duplicates found with the issues and pull requests combined. The alternative hypothesis is that the duplicates, as a proportion of the whole, should increase when the pull requests and issues are combined. To make the investigation more manageable, only 525 issues and 525 pull requests have been considered as part of this experiment, with an even distribution from each project. In comparing the results for a dataset of pull requests only to a dataset of issues and pull requests combined, the latter dataset will be larger and therefore would be expected to contain more duplicates, which is why the volume of duplicates found will be represented relative to the size of the dataset, otherwise the results will lose their value. This experiment will also be repeated on a per-project basis to explore how the number of duplicates found vary by project, and whether the impact of combining issues and pull requests into a single dataset has an equal impact across projects. Since an equal number of issues and pull requests have been taken from each project, it would be expected that the relative increase in duplicates found would be similar across projects.

5 RESULTS

5.1 Model Evaluation

The accuracy measured across various duplicate thresholds for each of pull requests only, and issues and pull requests combined, can be seen in Fig. 1. The results for the two datasets follow the same pattern, with both peaking at the same point, and the combined dataset returning better results across most duplicate thresholds. The combined dataset achieved a peak accuracy of 93.9%, which occurred when the threshold was set at 0.38. The pull requests dataset achieved a peak accuracy of 90.6%, which also occurred when the threshold was set at 0.38. The accuracy of the model was on average 4.25 percentage points higher when using the combined dataset. A recall rate of 90.5% was observed for the combined dataset with the duplicate threshold set at 0.38, while the dataset of pull requests only displayed a recall rate of 83.2% for the same duplicate threshold. The results from both of our datasets are a significant improvement on those from Li et al. (Li et al., 2017) (who considered only pull requests) in that our model can predict duplicates with a significantly greater accuracy and recall rate.

Fig. 2 shows the accuracy in predicting duplicates from the combined dataset compared to pull requests only, when evaluated for each repository individually. In doing this, the threshold for duplicates was set to 0.38 which was found to produce the greatest overall accuracy. In 11 out of the 15 projects, the accuracy was higher when evaluated on the combined dataset. The results are reasonably consistent between the two datasets, and also between projects. The highest accu-

[6]https://huggingface.co/sentence-transformers/all-MiniLM-L6-v2

[7]https://huggingface.co/spaces/mteb/leaderboard

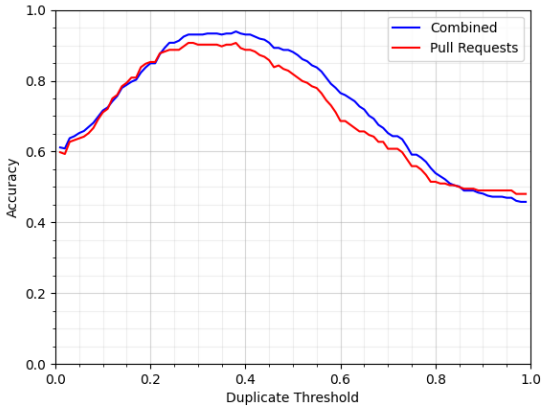

Figure 1: Accuracies for Varying Thresholds.

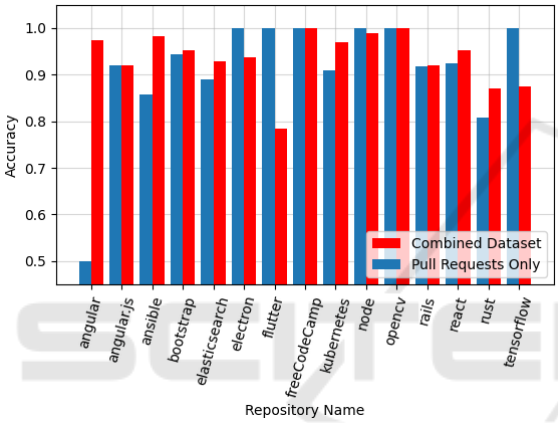

Figure 2: Accuracy evaluated for each repository.

racy is 100%, which occurs for multiple projects and for both datasets. Angular returned an accuracy of only 50% when considering pull requests only, which is significantly less than its accuracy for the combined dataset, meanwhile opencv returned 100% accuracy across both datasets.

5.2 Duplicate Prediction

Fig. 3 shows the volume of (predicted) sets of duplicates found, relative to the size of the input data (number of issues and pull requests considered), across several duplicate thresholds. The two datasets largely follow the same trend, albeit with clear separation between the results for each. For a sensitivity threshold of 0.38 (for which we achieved the greatest accuracy in the previous section), there was a significant difference between the two datasets. Fig. 4 shows the volume of duplicates found at this threshold for each on a per-project basis. The median of the differences was 7.84 with a 95% confidence interval ranging from 8.81 to 6.43 (Wilcoxon signed-rank

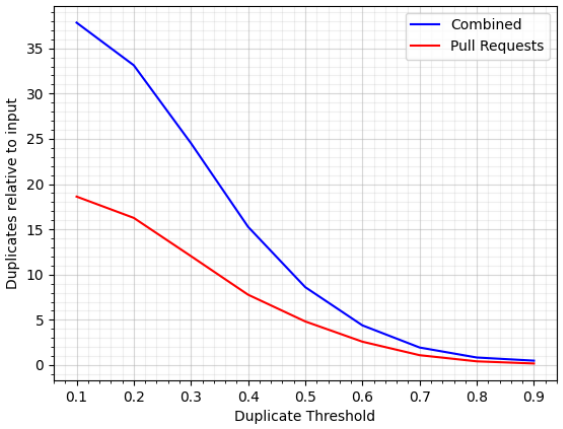

Figure 3: Sets of Duplicates Predicted Relative to Input Size.

test V=0, $p \ll 0.001$). This implies that the dataset comprised of issues and pulls produces a consistently higher volume of duplicates relative to the size of the input over the pulls-only dataset. Indeed, in all of the projects studied, the volume of duplicates found were much greater for the combined dataset compared to the dataset of pull requests only. The greatest observed percentage difference between the two datasets was for Angular.js where using the combined dataset improved the detection by a factor of 2.8.

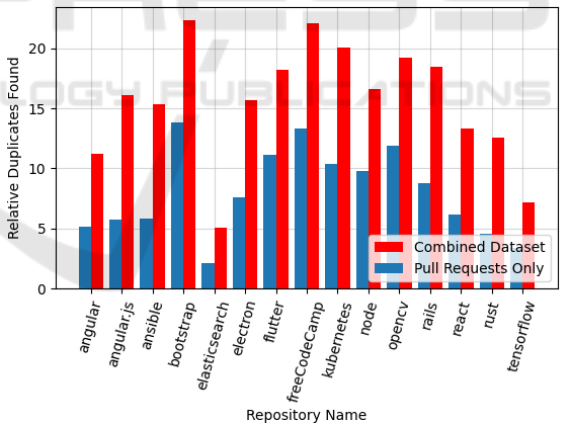

Figure 4: Relative duplicates found for each project.

5.3 Threats to Validity

One threat to the internal validity of this work lies in the method of identifying duplicates from resolved issues and pull requests. The regex pattern searches the comments of issues and pull requests for messages of the format *'duplicate of https://github.com/...'*. If a comment were to be found which *asks*, rather than asserts, whether a particular contribution is a duplicate of another, then this could be incorrectly identified as a duplicate. Although the console application

searched for duplicates in the same number of issues and pull requests from each project, the numbers of duplicates found were not evenly distributed among projects. This might therefore be a threat to the validity of the per-project results. Given the relatively small sample size of 15 projects, a post-hoc power analysis (Cohen, 2013) was conducted to evaluate statistical power. The observed effect size between combined dataset versus the pulls-only dataset was 7.84, and the resulting power of the test was 1, indicating that the study had an very high probability of detecting a true effect if one exists. The splitting of the text into chunks of 250 characters when training and testing the NLP model may also pose a threat to the validity of this work since sections of meaning may be split between chunks and hence lost. This might reduce the effectiveness of a NLP model trained on these chunks.

Affecting the external validity, while efforts have been made to ensure that a range of programming languages, technologies, and application domains were represented in the projects studied, there remains some potential that these results are not entirely reflective of the wider GitHub community. Furthermore, only open source GitHub projects were studied, so we cannot claim validity outside of this. For example in a smaller co-located agile team there may be a better inherent understanding of the relationship between issues and pulls (Taylor et al., 2006).

6 CONCLUSION

The aim of this paper is to assess the feasibility of automatically detecting duplicates in open-source GitHub projects using natural language processing techniques, and the potential benefits of applying these techniques on issues and pull requests alike. A Sentence Transformers model was evaluated on a dataset of known duplicate issues and pull requests and found it to be 93.9% accurate in detecting duplicates. The accuracy was also found to be 4.25 percentage points higher when considering issues and pull requests together, compared to pull requests only, thus supporting the alternative hypothesis that *combining issues and pull requests into a single dataset will show an increase in the accuracy of our NLP model*.This verifies the assumption that the title and description of issues serve the same purpose as the title and description of pull requests, and that the level of technical detail is similar between both. This model was then used to predict duplicates from a snapshot of unresolved issues and pull requests, and found that the volume of duplicates found when considering issues and pull requests together was consistently higher compared to when considering pull requests only, which supports the alternative hypothesis that *by combining issues and pull requests into a single dataset the volume of duplicates found will increase significantly*.

These results demonstrate the practical value of automatically detecting duplicates using semantic similarity techniques in that not only can we detect duplicates with great accuracy, but by considering issues and pull requests together we can detect a significantly greater volume of duplicates than we would if we were only considering pull requests. Our recall rate of 90.5% was 20.0 percentage points higher than that demonstrated by Li et al. (Li et al., 2017), while our model was evaluated on a larger number of GitHub repositories. Our recall rate is also 1.8 percentage points higher than that displayed by Wang et al. (Wang et al., 2019), with their model being evaluated on 14 GitHub repositories. Lazar et al. (Lazar et al., 2014) displayed a recall rate of 100% in detecting duplicate bug reports from three open-source systems, although it is unclear the degree to which the choice of open-source systems studied affects the results. Our model was evaluated on GitHub bug reports where theirs was evaluated on bug reports from other systems.

Extending our study to include other sources of bug reports would strengthen the work and constitutes future work. Further, as opposed to detecting duplicates from issues and pull requests which have already been submitted, one avenue of potential further study lies in detecting duplicate pieces of work at their inception. In the case of pull requests in particular, there may be some value in incorporating a machine learning model into the GitHub forking process through which contributors can check if their ideas or bug fixes have already been added as part of another pull request, thus detecting potential duplicates before development work has begun. In thinking about issues, a machine learning model might be incorporated into the submission process. Such work along with the findings in this paper could inform and then be used to alleviate the current waste of resources caused by duplicate pull requests and the resultant burned on those reviewing open source code.

REFERENCES

Cohen, J. (2013). *Statistical power analysis for the behavioral sciences*. Routledge.

Devlin, J., Chang, M.-W., Lee, K., and Toutanova, K. (2018). Bert: Pre-training of deep bidirectional trans-

formers for language understanding. *arXiv preprint arXiv:1810.04805*.

Gousios, G., Pinzger, M., and Deursen, A. v. (2014). An exploratory study of the pull-based software development model. In *Proceedings of the 36th international conference on software engineering*, pages 345–355.

Gousios, G., Storey, M.-A., and Bacchelli, A. (2016). Work practices and challenges in pull-based development: The contributor's perspective. In *Proceedings of the 38th International Conference on Software Engineering*, pages 285–296.

Gousios, G. and Zaidman, A. (2014). A dataset for pull-based development research. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 368–371.

Gousios, G., Zaidman, A., Storey, M.-A., and Van Deursen, A. (2015). Work practices and challenges in pull-based development: The integrator's perspective. In *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, volume 1, pages 358–368. IEEE.

Jiang, J., Lo, D., He, J., Xia, X., Kochhar, P. S., and Zhang, L. (2017). Why and how developers fork what from whom in github. *Empirical Software Engineering*, 22:547–578.

Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D., and Damian, D. (2016). An in-depth study of the promises and perils of mining github. *Empirical Software Engineering*, 21(4):2035–2071.

Kalliamvakou, E., Gousios, G., Blincoe, K., Singer, L., German, D. M., and Damian, D. (2014). The promises and perils of mining github. In *Proceedings of the 11th working conference on mining software repositories*, pages 92–101.

Lazar, A., Ritchey, S., and Sharif, B. (2014). Improving the accuracy of duplicate bug report detection using textual similarity measures. In *Proceedings of the 11th Working Conference on Mining Software Repositories*, pages 308–311.

Li, L., Ren, Z., Li, X., Zou, W., and Jiang, H. (2018). How are issue units linked? empirical study on the linking behavior in github. In *2018 25th Asia-Pacific Software Engineering Conference (APSEC)*, pages 386–395. IEEE.

Li, Z., Yin, G., Yu, Y., Wang, T., and Wang, H. (2017). Detecting duplicate pull-requests in github. In *Proceedings of the 9th Asia-Pacific symposium on internetware*, pages 1–6.

Li, Z., Yu, Y., Wang, T., Lei, Y., Wang, Y., and Wang, H. (2022). To follow or not to follow: Understanding issue/pull-request templates on github. *IEEE Transactions on Software Engineering*, 49(4):2530–2544.

Li, Z., Yu, Y., Zhou, M., Wang, T., Yin, G., Lan, L., and Wang, H. (2020). Redundancy, context, and preference: An empirical study of duplicate pull requests in oss projects. *IEEE Transactions on Software Engineering*, 48(4):1309–1335.

Ma, X., Wang, Z., Ng, P., Nallapati, R., and Xiang, B. (2019). Universal text representation from bert: An empirical study. *arXiv preprint arXiv:1910.07973*.

Manning, C. and Schutze, H. (1999). *Foundations of statistical natural language processing*. MIT press.

McClean, K., Greer, D., and Jurek-Loughrey, A. (2021). Social network analysis of open source software: A review and categorisation. *Information and Software Technology*, 130:106442.

Mombach, T. and Valente, M. T. (2018). Github rest api vs ghtorrent vs github archive: A comparative study.

Muennighoff, N., Tazi, N., Magne, L., and Reimers, N. (2022). Mteb: Massive text embedding benchmark. *arXiv preprint arXiv:2210.07316*.

Reimers, N. and Gurevych, I. (2019). Sentence-bert: Sentence embeddings using siamese bert-networks. *arXiv preprint arXiv:1908.10084*.

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

Taylor, P. S., Greer, D., Sage, P., Coleman, G., McDaid, K., Lawthers, I., and Corr, R. (2006). Applying an agility/discipline assessment for a small software organisation. In *Product-Focused Software Process Improvement: 7th International Conference, PROFES 2006, Amsterdam, The Netherlands, June 12-14, 2006. Proceedings 7*, pages 290–304. Springer.

Wang, Q., Xu, B., Xia, X., Wang, T., and Li, S. (2019). Duplicate pull request detection: When time matters. In *Proceedings of the 11th Asia-Pacific symposium on internetware*, pages 1–10.

Wang, W., Wei, F., Dong, L., Bao, H., Yang, N., and Zhou, M. (2020). Minilm: Deep self-attention distillation for task-agnostic compression of pre-trained transformers.

Wessel, M., Vargovich, J., Gerosa, M. A., and Treude, C. (2023). Github actions: the impact on the pull request process. *Empirical Software Engineering*, 28(6):1–35.

Yu, Y., Li, Z., Yin, G., Wang, T., and Wang, H. (2018). A dataset of duplicate pull-requests in github. In *Proceedings of the 15th international conference on mining software repositories*, pages 22–25.

Zhang, X., Chen, Y., Gu, Y., Zou, W., Xie, X., Jia, X., and Xuan, J. (2018). How do multiple pull requests change the same code: A study of competing pull requests in github. In *2018 IEEE International Conference on Software Maintenance and Evolution (ICSME)*, pages 228–239. IEEE.

Zhou, S., Vasilescu, B., and Kästner, C. (2019). What the fork: A study of inefficient and efficient forking practices in social coding. In *Proceedings of the 2019 27th ACM joint meeting on european software engineering conference and symposium on the foundations of software engineering*, pages 350–361.