PI-Link: A Ground-Truth Dataset of Links Between Pull-Requests and Issues in GitHub

Zakarea Alshara, Anas Shatnawi, Hamzeh Eyal-Salman, Abdelhak-Djamel Seriai, Maad Shatnawi

IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2022.0092316

PI-Link: A Ground-Truth Dataset of Links between Pull-Requests and Issues in GitHub

ZAKAREA ALSHARA[1], ANAS SHATNAWI[2], HAMZEH EYAL SALMAN[3], ABDELHAK-DJAMEL SERIAI[4], and MAAD SHATNAWI[5]

[1]Department of Software Engineering, Jordan University of Science and Technology, P.O.Box 3030, Irbid 22110, Jordan (e-mail: zmalshara@just.edu.jo)
[2] DRIT, Berger-Levrault, Montpellier, France (e-mail: anas.shatnawi@berger-levrault.com)
[3] Department of Software Engineering, Mutah University, Al-Karak 61710, Jordan (e-mail: hamzehmu@mutah.edu.jo)
[4] LIRMM Lab, University of Montpellier, 34000 Montpellier, France (e-mail: seriai@lirmm.fr)
[5] Department of Electrical Engineering Technology, Higher Colleges of Technology, Abu Dhabi, UAE (e-mail: mshatnawi@hct.ac.ae)

Corresponding author: Zakarea Alshara (e-mail: zmalshara@just.edu.jo).

ABSTRACT GitHub hosts Git repositories and provides issues-tracking services to provide a better collaboration environment for software developers. Issues and Pull-Requests are frequently used in GitHub to discuss and review the software requirements (new features, bugs, etc.) and software solutions (source code, test cases, etc.) respectively. The links between Issues and their corresponding Pull-Requests comprise valuable information to keep tracking current development as well as documenting knowledge for future development. Considering a large number of links, such information can be used to train machine learning models for several purposes such as feature location, bug prediction and localization, recommendation systems and documentation generation. To the best of our knowledge, no dataset has been proposed as a ground-truth of links between Issues and Pull-Requests. In this paper, we propose, PI-Link, a new significant and reliable ground-truth dataset composed of 50369 links that explicitly connect 34732 Issues with 50369 Pull-Requests. These links are automatically extracted from all (907,139) Android projects in GitHub created between January 1, 2011 and January 1, 2021. To better organize and store the collected data, we propose a metamodel based on the concepts of Issues and Pull Requests. Moreover, we analyze the relationships between Issues and their linked Pull Requests based on four features related to their titles, bodies, labels and comments. The selected features are analyzed in terms of their lengths and similarities based on three lexical and one semantic similarity metrics. The results showed promising similarities between Issues and their linked PRs at the lexical and semantic levels. In addition, some feature similarities are sensitive to the text length, whereas other feature similarities are sensitive to the term frequency.

INDEX TERMS Android, GitHub, ground-truth dataset, Issue, link, Pull-Request.

I. INTRODUCTION

GitHub is a hosting platform for software development and version control that supports the concept of collaborative social coding. It provides a friendly environment for contributors to communicate, collaborate, and promote software development. Contributors can easily track bugs, request new features, manage tasks, and handle continuous integration.

Issues and Pull-Request (PR) are common software artefacts that are used throughout all software development lifecycles in GitHub. The Issue lets contributors track the development work, and it is managed by the GitHub issue-tracking system. The PRs lets other contributors review and discuss the changes before they are submitted. Issues and PRs contain much information about software development and evolution, but from different perspectives: Issues capture the software requirements (e.g. features, bugs), while PRs reflect the software development solutions (e.g., source code, test cases).

The links between PRs and Issues are vital and useful because they link two development activities and can be leveraged to learn from past experiences and predict future solutions in different software engineering tasks. For example, in the feature location task, PR-Issue links are employed to identify mappings between features and their

corresponding concrete implementation in the software artefacts [1]–[3]. In bug prediction, PR-Issue links are useful for training data to feed and build bug prediction models [4]–[6]. In bug localization, PR-Issue links are used to construct localization models for finding potential buggy portions of source code [7]–[9]. In bug assignment, PR-Issue links could be used to build recommendation models for assigning bug reports to relevant developers [10]–[12].

In GitHub, PR-Issue links are manually established by contributors either implicitly by adding the Issue identifiers in PR logs, or explicitly by adding the links using the GitHub metadata. Moreover, it is manually hard to find and distinguish such links between Issues and their associated PRs for different development and maintenance activities because such a task is challenging, time-consuming, and error-prone task, even for senior developers. Also, all PRs-Issues links are miss-documented as a collection but it is individually documented (implicitly and explicitly) and scattered over hundreds or thousands of PRs and Issues of given repositories.

The links between Issues and their corresponding Pull-Requests comprise valuable information to keep tracking current development as well as documenting knowledge for future development. Considering a large number of links, such information can be used to train machine learning models for several purposes such as feature location, bug prediction and localization, recommendation systems and documentation generation. To the best of our knowledge, no dataset has been proposed as a ground-truth of links between Issues and Pull-Requests.

In this paper, we propose, PI-Link, a new significant and reliable ground-truth dataset composed of 50369 PR-Issue links extracted from 5742 Android projects on GitHub. To do so, we develop a GitHub scraper tool that automatically extracts all explicit PR-Issue links (genuine and examined) from all (907,139) Android GitHub projects created between 2011 and 2021. Moreover, we propose a metamodel to better organize and store the collected PR-Issue links that contain valuable detailed information about Issues and PRs of each link. This ground-truth dataset can be exploited to evaluate the effectiveness of different future works on this subjects. For example, recovering the missing Issue-PR links, build recommender systems, etc. To make our dataset accessible for the community, we publish it on *Kaggle* [1] [2]: a well-known dataset repository.

To better understand the content of our dataset, we analyze the relationships between Issues and their linked PRs based on four features related to their titles, bodies, labels and comments. The selected features are analyzed in terms of their lengths (i.e., number of tokens) and similarities. For the similarity analysis, we rely on four similarity metrics that are frequently used in NLP. These are Jaccard, Cosine, and Levenshtine as lexical similarity metrics and BERT as a semantic similarity metric. The results show that some feature similarities are sensitive to the text length, whereas other feature similarities are sensitive to the term frequency.

The rest of the paper is structured as follows. Section II introduces the principle of GitHub flow and PR-Issue link. Section III presents the approach to extract our ground-truth dataset. Section IV illustrates our ground-truth metamodel. The analysis of our dataset is presented in Section V. The feature selection and the analysis results are introduced in Section VI and Section 13 respectively. Threats to validity are presented in Section VIII respectively. The use cases of our dataset are introduced in Section IX. Section X discusses the most closely related work. Finally, the conclusion and future works are introduced in Section XI.

II. THE ANATOMY OF GITHUB

A. GITHUB FLOW

GitHub is a software development and version control hosting repository based on the Git version control system [13]. In addition to the distributed version control system provided by Git, GitHub provides other features like access control, bug tracking, task management, and continuous integration. GitHub is commonly used to host open-source software development projects, with over 100 million developers working on over 352 million repositories (more than 41 million public repositories [14]) by November 2022 [15] [16].

GitHub flow is a distributed branch-based workflow designed to work with Git and GitHub. The GitHub flow is helpful for software development teams to manage each phase of software development easily and make deployments regularly. Branch-based workflow helps developers to work on isolated development without affecting other branches in the project (repository). In GitHub, each repository has one default main branch, usually named Master, and other branches added by developers. Usually, the Master branch should always be deployable. Therefore, branches allow developers to contribute without affecting continuous development and integration. For example, if a new feature is needed to be added to the main branch, developers create a new branch from the main one to add the needed feature. Once the feature is ready, it is merged back into the main branch. Therefore, it helps developers to work simultaneously on the same repository without affecting the development process.

The GitHub flow is illustrated in Figure 1. First, a new branch is created to develop new features, fix bugs, or do some enhancements. The new branch is a safe container to make and revert changes to solve any changes mistakes. Then, developers push their changes (adding, editing, and deleting files) as commits on the new branch. Each commit has a descriptive message to help the owner and future contributors understand the commit changes (e.g. add localization feature, fix a typo, enhance performance). After the changes are completed, the developers create a PR of its commits. The PR has a summary (descriptive title and body) of the changes and solved problems. Moreover, it keeps records of changes to the code. If the PR handles an Issue

[1] https://www.kaggle.com/datasets/zakareaalshara/android-closed-issues-20110101-20210101-clean

[2] API command: kaggle datasets download -d zakareaalshara/dataset

created by others, the PR should be linked with that Issue to show the working progress. Next, collaborators discuss (i.e. adding comments) and review the PR to make a decision (e.g. approve, reject, or enhance). Reviewers can comment (i.e. leave questions, comments, and suggestions) on the PR or on specific commits. PR review is effective; hence, it is mandatory by some repositories before PRs merge to the main branch. Finally, once the PR is approved and tested, developers merge the PR into the main branch to reflect the changes update. GitHub allows deploying from a branch for final testing before merging with the main branch. When the PR is merged, its state is automatically transferred from open to closed.

B. PR-ISSUE LINKS IN GITHUB

Issues and PRs are software artefacts that are not supported by Git but by GitHub. On the one hand, the primary purpose of using Issues is to track development work on GitHub, which is usually stored in issue-tracking systems. It can be used to track new ideas, bugs, tasks, and other important information. On the other hand, PRs are used to propose and collaborate on changes with other reviewers and contributors, which is usually stored in version control systems. Issue and PR contain much information about software evolution, but from different perspectives: Issues describe the problem domain (e.g. requirements and bugs), while PRs reflect the solution domain (developing requirements, fixing bugs). The links between PRs and issues are important and valuable because they link the problems with their solutions. Therefore, It can be used to better understand the changes and feed experience and knowledge in various software engineering tasks. For example, in feature location, PR-Issue links are used for mappings between features and their implementing source code [17]. Moreover, In bug localization, PR-Issue links are helpful in building localization models [18], [19].

On GitHub, permitted contributors can link Issues to PRs on the same repository. Consequently, collaborators can see the working progress on these linked Issues and PRs. The PR-Issue links might be residing in the same or different repositories. When linked PRs are merged, the states of their linked Issues are automatically transferred from open to closed (solved). Contributors can link Issues to PRs either implicitly or explicitly [20]. In the implicit manner, PRs are linked to issues by using keywords supported by GitHub (i.e. close(s, d), fix(es, ed), resolve(s, d)) in the PR's description or in its commit message (see Figure 2). However, this manner may not be a natural workflow for every team and has many consequences. First, editing or removing implicit PR-Issue links is tedious; the owner must edit the PR description to edit or remove the link's keywords. Second, these keywords could be used in a commit message. The related Issue will be closed when the commit is merged into the branch, but the PR that has the commit will not be linked to that Issue. Consequently, the main link between the problem and its solution will be missing.

In the explicit manner, PRs and Issues are manually linked from either the PR sidebar or the Issue sidebar (see Figure 3). So, contributors and reviewers can immediately see the status of development work on the linked Issues and the associated PRs. This manner guarantees the main link between the problem and its solution. Moreover, the PR-Issue links can be easily edited or removed by any permitted contributor. Consequently, using this manner is considered better than using the implicit one.

GitHub provides many APIs to automatically extract many information (e.g. Issues, PRs, commits, etc.) regarding GitHub repositories [21], [22]. One of these pieces of information is the PR-Issue links. However, these APIs could detect explicit relations but not implicit ones. The implicit linked PRs with their Issues need more text investigations and human efforts to ensure these relations because it is built using the keywords. Consequently, in this paper, we aim to extract explicit PR-Issue links.

III. DATASET EXTRACTION APPROACH

To automatically collect our dataset, we developed a GitHub scraper tool named *PR-Issue-Scraper* using python. It is published in GitHub[5]. *PR-Issue-Scraper* extracts explicit PR-Issue links using GitHub GraphQL API [22]. We design our tool to deal with the GraphQL request limitation; hence GitHub GraphQL API rate limit is 5,000 points per hour [23]. We ran *PR-Issue-Scraper* to target all Android repositories on GitHub in the time period between the 1st of January 2011 to the 1st of January 2021. The run takes around 30 days (24/7) to archive our target.

The architecture of *PR-Issue-Scraper* is composed of three main components: *Scraper*, *Request-meter* and *Validator*, as presented in Figure 4. The *Scraper* component is responsible for managing queries requested from GitHub to collect data using GraphQL APIs. The *Request-meter* component controls the *Scraper* component to be within the request limit range (i.e., 5,000 points [6] per hour). The *Validator* component aims to verify the results of the *Scraper* and remove noise data, e.g. Issue-Issue links, PR-PR links.

For better understanding, Algorithm 1 illustrates the detailed process of *PR-Issue-Scraper*. In the beginning, all Android repositories placed on GitHub were queried using GitHub GraphQL API. To retrieve only Android projects, we assume that an Android project should have the "Android" keyword in its description and should be implemented using Java or Kotlin programming languages. However, many non-Android projects could have previous assumptions. Therefore, all repositories returned from the query are filtered to eliminate the non-Android projects. To do so, all repositories are investigated to check whether they have an AndroidManifest.xml file or not. This comes from the fact that any android project contains an AndroidManifest.xml file. After that, for

[3]https://github.com/AdAway/AdAway/pull/784
[4]https://github.com/AdAway/AdAway/issues/781
[5]https://github.com/zakarea/PR-Issue-Scraper
[6]Every piece of information (e.g. a title, a label) acquired from GitHub GraphQL costs one point.

This article has been accepted for publication in IEEE Access. This is the author's version which has not been fully edited and content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2022.3232982

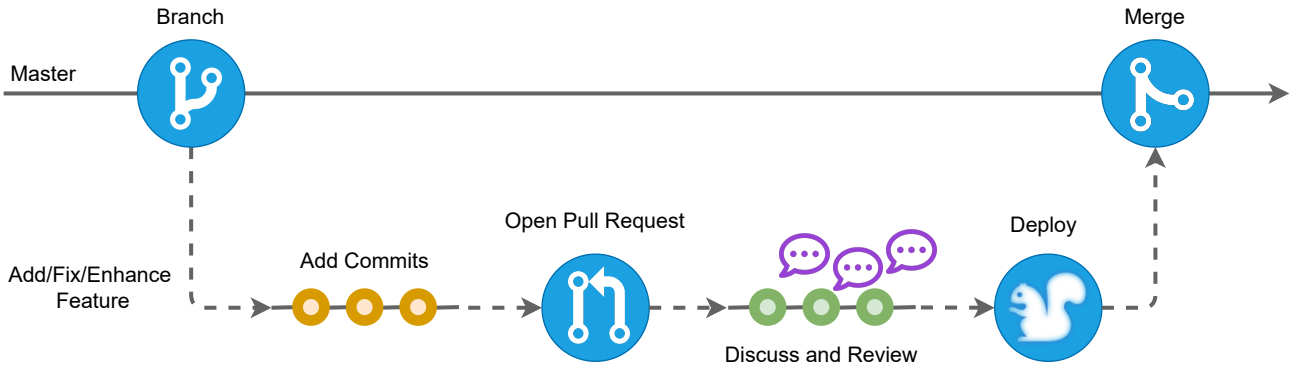

FIGURE 1: Branch-based workflow in GitHub.

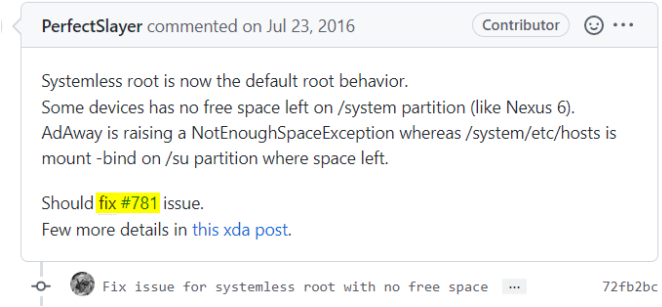

FIGURE 2: A PR linked to an Issue implicitly.[3]

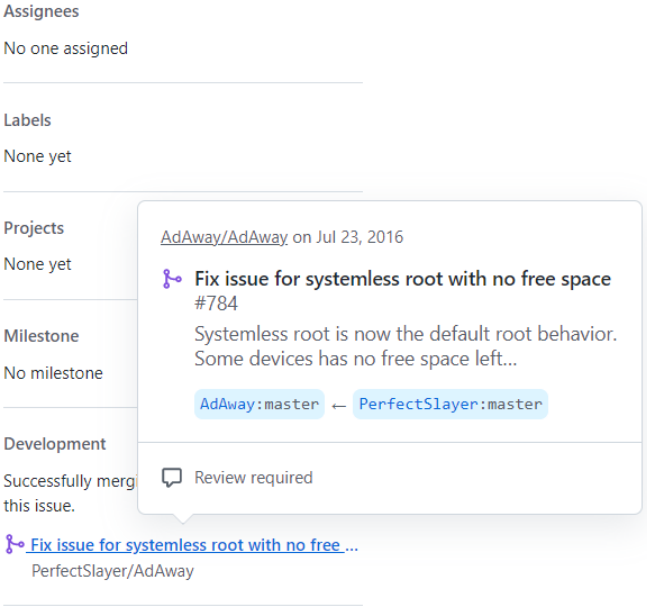

FIGURE 3: A PR linked to an Issue explicitly.[4]

each Android project, the closed Issues are queried. We targeted closed Issues to guarantee that they are solved and could be linked with a PR. As open Issues do not have links to PRs, we exclude them from our dataset. Next, all closed Issues are inspected to check their explicit links with other software artefacts. The explicit links can be found in the Issue sidebars (c.f. Figure 3). We collect data from Issue sidebars using the *timelineitems* interface provided by GraphQL APIs. The returned explicit links connect the current Issue with PRs, Issues or merge requests. As we aim to collect PR-Issue links, we exclude the other ones. As a result, only Issues that have PR-Issue links are returned in our dataset.

To deal with the rate limit of the GraphQL APIs, each query is examined before it is submitted to the GraphQL API using the *RateLimitMeeterListener* function. *RateLimitMeeterListener* verifies whether the current query exceeds the limit rate or not. In case it exceeds, *RateLimitMeeterListener* divides the query to make it fit with the rate limit.

IV. DATASET MODEL

To better organize and store the collected data by our *PR-Issue-Scraper*, we propose a metamodel of our ground-truth dataset. This metamodel is described in Figure 5. The metamodel is composed of *Repository*, *Issue*, *PullRequest*, *Comment*, *Label*, and *Commit*. In the following, we describe these metamodel elements.

A. REPOSITORY

The repository is a container for the project's files and its revision history. Contributors can discuss and manage their software development within the repository. Each repository has a unique name over all GitHub repositories.

B. ISSUE

Issues are used to track ideas, tasks, bugs or feedback on GitHub. Issues tackle challenges in the requirements/specifications, workflow, design, implementation, or even production in software development. Each Issue has a unique *number* based on its repository. This unique *number* could be used as a reference to its Issue (e.g. could be used to establish an implicit link by a PR). The issue *url* is a web-based url. The *title* and *body* are textual data to describe the Issue. The creation and closing Issue time-stamps are

Algorithm 1: PR-Issue-Scraper Algorithm.

Data: $Points \leftarrow RateLimitMeeterListener(Points)$
Result: $\forall(PR \leftrightarrow Issue) \in Android\ \exists\ GitHub$
$AndroidRepos \leftarrow GraphQL.query(\text{“}keyword : Android\text{”}, \text{“}lang : Java|Kotlin\text{”})$;
$AndroidRepos \leftarrow FilterAndroidRepos(AndroidRepos)$;
$AndroidIssues \leftarrow GraphQL.query(\text{“}type : issue\text{”}, \text{“}is : closed\text{”})$;
$PRIssueLinks \leftarrow getPRIssueLinks(AndroidIssues)$;
while $N \neq 0$ **do**
  if N *is even* **then**
    $X \leftarrow X \times X$;
    $N \leftarrow \frac{N}{2}$ *[r]This is a comment
  else
    if N *is odd* **then**
      $y \leftarrow y \times X$;
      $N \leftarrow N - 1$;
    end if
  end if
end while
Function `FilterAndroidRepos(`$Repos$`)`**:**
  $AndroidRepos$;
  foreach $repo \in Repos$ **do**
    if $\text{“}AndroidManifest.xml\text{”} \in repo$ **then**
      $AndroidRepos \longleftarrow repo$;
    end if
  end foreach
  return $AndroidRepos$;
End Function
Function `getPRIssueLinks(`*Issues*`)`**:**
  $Links$;
  foreach $issue \in Issues$ **do**
    foreach $link \in issue$ **do**
      if $link \equiv PR \leftrightarrow Issue$ **then**
        $Links \longleftarrow link$;
      end if
    end foreach
  end foreach
  return $Links$;
End Function
Function `RateLimitMeeterListener(`$Points$`)`**:**
  $Action$;
  if $\text{“}Points\text{”} \geq GraphQL.query(\text{“}limit\text{”})$ **then**
    $Points -= GraphQL.query(\text{“}limit\text{”})$;
  else
    $Action \longleftarrow DIVIDE - QUERY$;
  end if
  return $Points$;
End Function

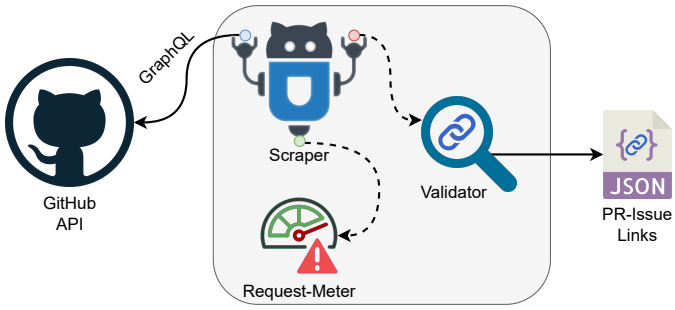

FIGURE 4: The PR-Issue-Scraper architecture.

represented by *createdAt* and *closedAt*, respectively. Each Issue must reside in one repository. Moreover, it may have *Comments* and *Labels*. *Comments* are inserted by collaborators to discuss many things related to the Issue. For example, the comments could explain the optimal way to solve the Issue or the responsible developers. *Labels* are selected from the Issue creator to give Issue types. For example, if an Issue talks about an existing security bug, its labels could be “bug" and “security". Finally, the relationship between the Issue and PR could be one of the three scenarios:

- In the first scenario, one PR is linked with one Issue. For example, a bug Issue is solved only by one PR.
- In the second scenario, many PRs are linked to the same Issue. For example, a bug Issue could be caused by two parts of the implementation, and each part is the responsibility of different development teams. In this case, each team creates their PR. Then PRs will be merged and linked to their Issue.
- In the third scenario, one PR is linked to many Issues. For example, a bug Issue and an improvement Issue are related to the same problem and could be solved by the same development team. In this case, the corresponding PR will be linked to both Issues.

C. PULL-REQUEST

The PullRequest is a container of changes that have been done by developers. It allows reviewers to discuss and review the changes with collaborators and make decisions before they are merged into the main branch. Similar to the *Issue*, *PullRequest* has a unique *number* that could be used as a reference to its PR (e.g. could be used to establish a link with Issues). It also has a web-based *url*, and textual data represented by a *title* and *body* to describe the PR. The creation and closing time-stamps are represented by *createdAt* and *closedAt*, respectively. A PR may also have *comments* and *labels*. In addition to Issue, *PullRequest* could have code *Commits*.

D. COMMENT

It includes a piece of textual information provided by reviewers and developers.

E. LABEL

Each label has a name and a textual description. Labels are very useful for classifying Issues and PRs. The most popular labels on GitHub are *bug, enhancement, duplicate, documentation,* and *wontfix*.

F. COMMIT

The *Commit* has a *url* that is used to access many features like file changes and differences between versions. Moreover, each *Commit* should have a textual message that briefly describes the changes.

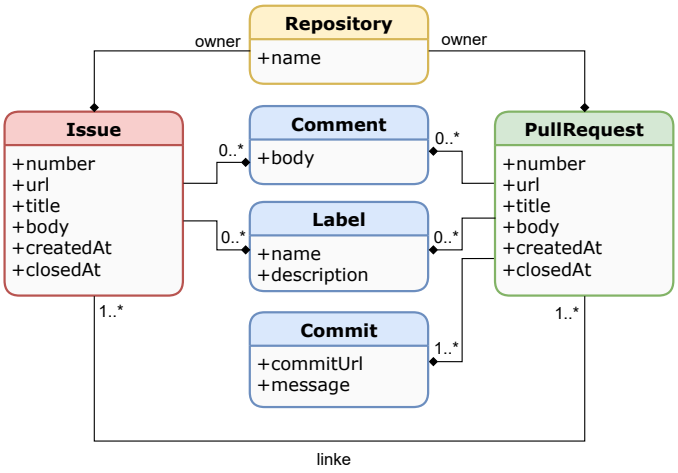

FIGURE 5: The metamodel of the dataset.

V. DATASET ANALYSIS

PR-Issue-Scrapper identifies 907,139 Android projects in GitHub. Among these projects, it detects 5742 Android projects that have explicit PR-Issue links. Table 1 provides statistical descriptions of the ground-truth dataset in terms of the total numbers in the whole dataset, the min, max, mean and standard deviation (Std) at the level of projects.

TABLE 1: Ground-truth dataset statistics

	Total	Min	Max	Mean	Std
Issue	34732	1	974	6.04	30.13
PR	50369	1	1570	8.77	44.78
Issue comment	96366	0	7185	16.78	109.51
PR comment	133865	0	3206	23.31	101.9
Issue label	22235	0	1087	3.87	26.96
PR label	8882	0	1192	1.54	23.01
commit	90740	0	3206	15.82	73.19

A. RESULTS OF ISSUES AND PRS

The results show that the dataset contains 34732 Issues linked to 50369 PRs. The difference in the numbers of PRs and Issues is due to the multiple cardinality relationships between PRs and Issues. For a better illustration of their distributions,

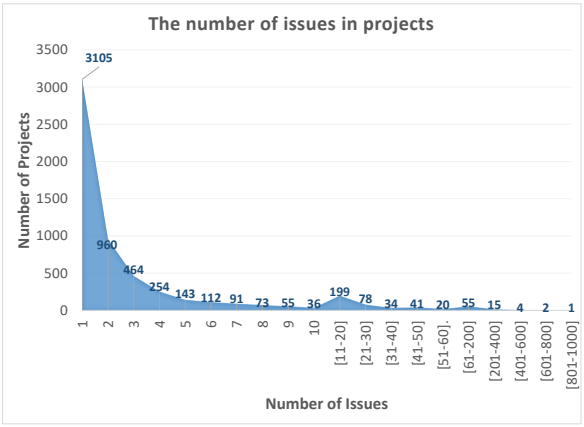

FIGURE 6: The distribution of Issue count over projects.

Figures 6 and Figures 7 show the distributions of Issues and PRs over projects, respectively. These distributions show that we have 3105 and 3101 projects having only a single Issue or PR, respectively. However, these Issues and PRs only represent 8% (3105/34732) of Issues and 6% (3101/50369) of PRs of our dataset. On the other hand, the dataset includes some projects that have large numbers of Issues and PRs. For example, we have a project that includes 974 Issues and 1570 PRs. These represent 2.8% (974/34732) and 3.1% (1570/50369) of Issues and PRs of our dataset, respectively. Considering another example from the data centralization, we have 51 projects that have between 100 and 200 PRs. In total, their PRs are 7022 which makes 13.9% (7022/50369) of all PRs. Referring to the mean values, the average number of Issues and PRs are 6.04 Issues and 8.77 PRs, respectively. As a result, our dataset is representative for different types of PR-Issue links in terms of their co-occurrences together within the same projects starting from singular occurrences up to a large number of co-occurrences (e.g., 947 co-occurrence Issues). This helps to make our dataset not biased.

Concerning the multiple cardinality relationships, the results show that roughly 32% $(1 - (6.04 \div 8.77))$ of Issues are linked to multiple PRs. This means that the solutions to these Issues have been proposed by multiple contributors at multiple levels.

B. RESULTS OF COMMENTS RELATED TO ISSUES AND PRS

The Issues and PRs of our dataset have 96366 and 133865 comments in total, respectively. We find that 1296 Issues (represents 3.7% (1296/34732) of Issues) and 8 PRs do not have any comments. The reason is that the PRs usually go through a review and discussion process represented by comments in the PRs, but this process is not mandatory to have happened for Issues. Another reason is that the corresponding

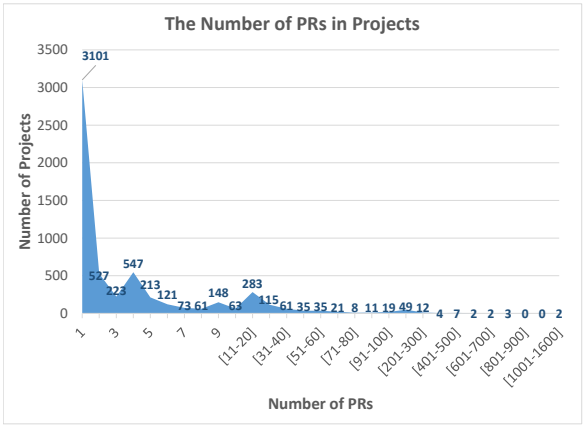

FIGURE 7: The distribution of PR count over projects.

contributors have the privilege of skipping the review and discussion stage because they are trusted enough by the community (e.g., having enough expertise). On the other hand, we identify that some projects have a large number of comments, e.g., 7185 comments related to Issues and 3206 ones related to PRs. This means that the corresponding Issues and PRs have got long discussions between the contributors to define their solutions. On average, projects have 40.09 comments (i.e., 16.78 comments for Issues and 23.31 comments for PRs). Based on these results, we observe that the number of comments per project is varied based on their standard deviation values which are 109.51 and 101.9, respectively. Indeed, these comments could be valuable for the research community to extract knowledge, e.g., about how to solve similar Issues.

C. RESULTS OF LABELS RELATED TO ISSUES AND PRS

The total numbers of labels are 22235 and 8882 for Issues and PRs, respectively. As it is noted, we have $\approx$ 2.5 more labels for Issues than PRs. The reason is related to the fact that contributors maybe not have a complete awareness of the problems at the time of creating Issues. Thus, they could use more labels (i.e., keywords) for describing the types of the corresponding Issues. On the other hand, contributors of PRs have a complete awareness of the types of problems as they have already solved them. Thus, they use a few labels to briefly describe the types of these PRs. On average, Issues have 3.87 labels while PRs have only 1.54 ones. Based on the standard deviation values, we note that Issues and PRs have variability in terms of the number of labels.

Moreover, we find that %8.1 (2830/34732) of Issues and %9.2 (4636/50369) of PRs do not have any label. Therefore, it could be a future work to learn a machine learning model for predicting the missing labels using the labelled Issues and PRs in our dataset.

D. RESULTS OF COMMITS RELATED TO ISSUES AND PRS

The total number of commits attached to the identified PRs is 90740. We note that only 11 PRs do not have any commits. For instance, an Issue demands executing test cases. In this case, the results of these test cases are attached in the body of the corresponding PR, without changing the repository content (i.e., do not need to add a commit). However, these can be neglected as they do not impact the quality of our dataset compared to 90740 commits. On the other hand, for other projects, we find a huge number of commits. For instance, we identify 3206 commits in one project. On average, we have 15.82 commits connected to PRs per project. However, we have a large variation in terms of the number of commits per project based on the standard deviation (73.19). This means that the size of solutions to Issues is varied. For example, some Issues need a single commit to be solved, while some others could need a large number of commits done probably by different contributors (e.g., a development team).

VI. FEATURE SELECTION

In our dataset, we identify four main features that can be used to distinguish between PR-Issue links. These features are *Title*, *Body*, *Comment* and *Label* of Issues and PRs. We select these features because they include much information in terms of the textual representation of their Issues and PRs. For instance, Titles provide short-term descriptions. Bodies include long-term descriptions. Comments represent the communication between the contributors. Labels refer to the types of Issues and PRs.

In this section, we present the text size for each feature to better understand their information amounts. We calculate the text size in terms of word count. To do so, we first pre-processed the text by cleaning up text (e.g. removing unwanted words and punctuation marks). Then we count the number of words represented by each feature text.

A. FEATURE PRE-PROCESSING

We apply the following standard pre-processing steps to clean the dataset.

1) **Tokenization:** we tokenize (split) each feature text to a list of words. The tokenization strategy considers space as a separator and discards punctuation marks and unnecessary symbols. To do so, we use the "spaCy" library to end up with useful words.
2) **Stopword Removal:** in this step, we remove all stop-words that frequently occur in the word list but do not carry any meaning (e.g. the). We use "NLTK " (Natural Language Toolkit) to remove all stop-words.
3) **Stemming:** it is a process of reducing words to their root form. We use one of the most popular stemming algorithms, "Porter stemmer", provided by NLTK library. Moreover, we convert the entire text into lower case characters

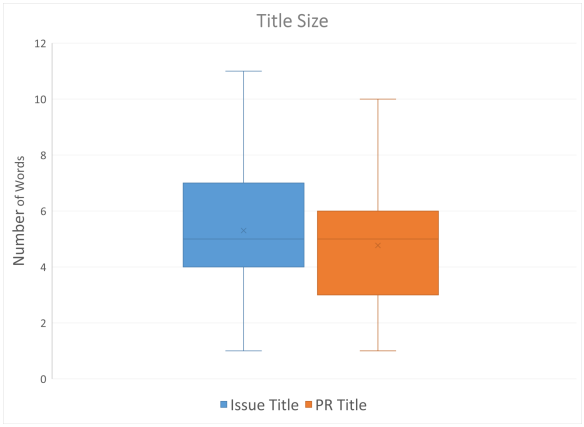

FIGURE 8: The number of words counted from Issue and PR titles.

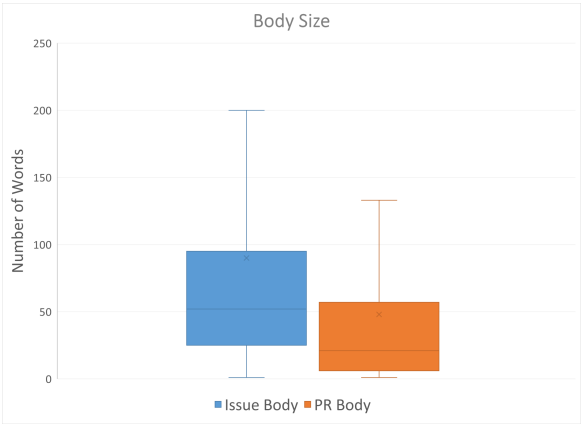

FIGURE 9: The number of words counted from Issue and PR bodies.

B. WORD-BASED MEASUREMENT RESULT

1) Title Word Count

Figure 8 shows the distribution of the Title sizes in terms of the number of words for Issues and PRs in our dataset. We can observe that the Issue Titles are slightly bigger than the PR Titles, but almost both have the same median value. The Title sizes range from one to 11 words and from one to 10 words for Issues and PRs, respectively. The Title sizes consider short compared with Body sizes.

2) Body Word Count

Figure 9 illustrates the distribution of the Body word sizes for Issues and PRs in our dataset. We can see that the Body sizes of the Issues are relatively vast bigger than the Body sizes of the PRs. The median value is 52 in the case of Issue, whereas the median value is 21 in the case of PR. The reason behind this size variation between Issue and PR is that the contributors need much more words to explain better a new feature or bug (Issue) while needing fewer words to explain its solution (PR).

3) Comment Word Count

Figure 10 illustrates the distribution of the Comment word sizes for Issues and PRs in our dataset. We can observe that the Comment sizes of the Issues are relatively bigger and much spread than the Comment sizes of the PRs. The median values are relatively close to each other, 46 and 39 for Issue and PR, respectively. As same as the median values, the mean values are also close to each other. Hence the average values of words are 145 and 140 for all Comments in Issue and PR, respectively. The reason behind this size variation between the Issue's Comments and the PR's Comments is that, in many cases, we found a single Issue linked with more than one PR. Therefore, The contributors comment in one place

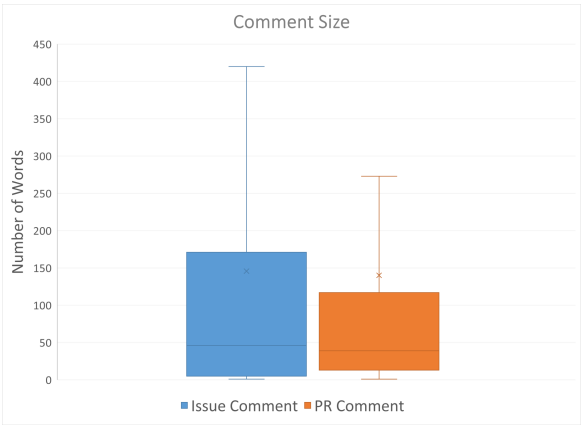

FIGURE 10: The number of words counted from Issue and PR comments.

in case of Issue, whereas commenting in many places in case of many PRs.

4) Label Word Count

Figure 11 shows the distribution of the Label sizes for Issues and PRs in our dataset. As same as the Title sizes, the Label sizes are small compared with the Body and the Comment Sizes. The Label size range from one to 6 and from one to 3 for labelled Issue and PR, respectively. Figure 12 represents the top 20 Labels occurrences that occurred in more than 1000 times for all Issues and PRs in our dataset. We can obviously observe that the *"bug"* label has the biggest share (13518 times) among the other Labels. The next two Labels *"enhancement"*, and *"closed"* occurred more than

5000 times, and the next two Labels *"android"*, and *"feature"* occurred more than 3000 times. Finally, the rest Labels occurred between 1000 and 2000 times.

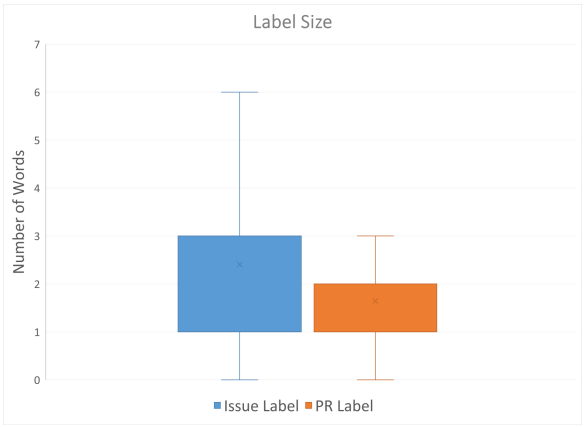

FIGURE 11: The number of words counted from Issue and PR labels.

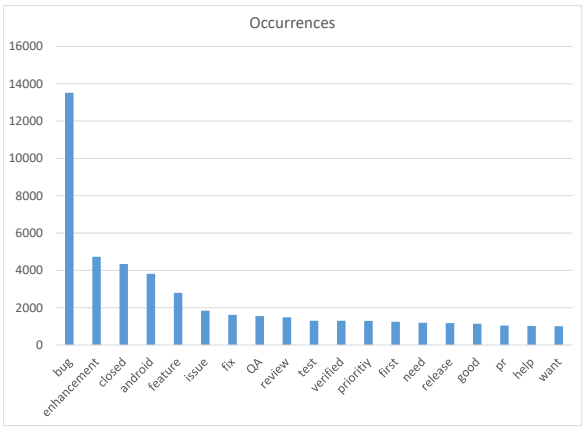

FIGURE 12: The number of labels frequency.

VII. FEATURE ANALYSIS

To analyze the relationships between Issues and their linked PRs, our feature analysis is based on measuring the similarity between features of Issues and PRs belonging to the same PR-Issue link. For a given PR-Issue link, we calculate the similarity between the Issue title and the PR title, the Issue body and the PR body, the Issue comment and the PR comment, and the Issue label and the PR label. To do so, we rely on four text similarity metrics that are commonly used in NLP. As lexical metrics, we select *Jaccard*, *Cosine* and *Levenshtein* metrics that are based on different measurement techniques. As a semantic metric, we rely on the *BERT* similarity metric. We select these metrics because they are commonly used in the literature. Each similarity metric could describe the data similarity from a different point of view. For example, Jaccard focuses on similar words regardless their frequency unlike Cosine. Levenshtein measures the similarity at the level of characters unlike the other metrics that rely on words. This means that Levenshtein is able to identify similar words in case of abbreviations and typos. Unlike the other lexical similarity metrics, BERT measures the semantic (meaning) similarity between texts.

In the remaining of this section, we present how to measure the similarity based on these four metrics. Then, we discuss the results of comparing features of Issues and PRs based on the four metrics.

A. SIMILARITY METRIC

1) Jaccard

Jaccard metric, also known as Jaccard Similarity Coefficient, is used to measure the similarity and diversity of two texts based on the lexical structure of their contents. The Jaccard similarity score between two texts is calculated based on the ratio between the number of unique common words and the total number of unique words in both texts. In other words, Jaccard similarity Jaccard(T_1, T_2) measures the similarity between text T_1 and text T_2 by dividing the number of unique common words in T1 and T2 on the total number of unique words in T_1 and T_2. Jaccard similarity scores are formally obtained based on Equation 1:

$$Jaccard(T_1, T_2) = 1 - \frac{words(T_1) \cap words(T_2)}{words(T_1) \cup words(T_2)} \quad (1)$$

where $words(T)$ denotes the set of distinct words in text T. Jaccard similarity scores range from 0 to 1 where 0 means no similarity and 1 refers to complete similarity.

2) Cosine

Cosine similarity measures the lexical similarity between two texts based on their term-frequency vectors (embeddings). A term-frequency vector of a given text is represented by attributes that record the frequency of distinct words in that text. Cosine similarity measures the cosine angle between the two vectors by computing the dot product of the vectors divided by the product of their lengths. Cosine similarity scores are formally obtained based on Equation 2:

$$Cosine(T_1, T_2) = cos(\theta) = \frac{\overrightarrow{T_1}.\overrightarrow{T_2}}{\|\overrightarrow{T_1}\|\|\overrightarrow{T_2}\|} \quad (2)$$

where $\overrightarrow{T_1}$ and $\overrightarrow{T_2}$ denote to the vector of text T_1 and text T_2, respectively.

For example, two identical texts represented by two symmetrical vectors have a Cosine similarity of 1, two orthogonal vectors have a Cosine similarity of 0, and two opposite

vectors have a Cosine similarity of -1. However, the Cosine similarity is usually used in positive space, where the outcome scores are neatly bounded in $[0, 1]$. A score closer to 0 implies less similarity, whereas a score closer to 1 implies more similarity.

3) Levenshtein

Levenshtein similarity measures the lexical similarity between text T_1 and text T_2 based on the minimum number of characters that need to be edited to make T_1 identical to T_2 and the size of T_1 and T_2. An edit is defined by either an insertion, a deletion, or a replacement of a character. The lower number of edits required, the more the texts are similar to each other.

For two Texts T_1 and T_2 with character lengths $\|T_1\|$ and $\|T_2\|$, the Levenshtein distance is defined as follows:

$$Levenshtein(T_1, T_2) = 1 - \frac{minEdit(T_1, T_2)}{max(\|T_1\|, \|T_2\|)} \quad (3)$$

The scores of Levenshtein similarity range from 0 to 1, where 0 refers to no similarity and 1 denotes to complete similarity.

4) BERT

BERT refers to Bidirectional Encoder Representations from Transformers. It is one of the most popular deep-learning models for NLP. BERT is a pre-trained language-based transformer model that is developed by Jacob et al. in Google [24]. The BERT transformer model can be used to embed (i.e. vector representations) the semantics of texts. Therefore, it can be used hand in hand with a similarity measurement to measure the semantics between two texts. In this paper, we used the BERT transformer that works based on *"bert-base-nli-mean-tokens"* pre-trained model to measure the semantic similarity score between two texts using the Cosine measurement. The BERT transformer transforms text T_1 and text T_2 to vector $\overrightarrow{T_1}$ and $\overrightarrow{T_2}$, respectively. Then, the similarity score is obtained using Equation 2.

B. SIMILARITY RESULTS

The results of the four feature analyses based on the four similarity metrics are shown in Figure 13. It illustrates the box-whisker plots of Jaccard, Cosine, Levenshtein, and BERT similarity scores for Title, Body, Comment, and Label of Issues and their linked PRs in our dataset.

1) Title Similarity Results

The results show that Levenshtein and BERT have a similar distribution as their box-plots are almost symmetric and identical. Based on their results, the titles of Issues and their linked PRs have similarity scores of 55.3% and 56.3% on average, respectively. As Levenshtein similarity is sensitive to the lengths of the texts, its results show that titles of the Issues and their linked PRs have akin lengths for most of the PR-Issue links in our dataset.

Jaccard has a similar distribution as Levenshtein and BERT but with lower scores. Although Q1, Q2 and Q3 have different values, the number of elements (i.e., similarity scores) between Q1-Q2 and Q2-Q3 are almost the same. The small difference in the position of the Jaccard box-plot compared to Levenshtein and BERT ones is because Jaccard ignores similarities coming from word frequency. As it is a small difference, we do not have many words that have a frequency.

The similarity scores obtained by Cosine have a completely different distribution compared to the other similarity metrics. Based on its results, 25% of PR-Issue links do not have any similarity (i.e., zero similarity) between their titles. Moreover, most of the remaining similarity scores are spread between zero and 42.9%. The reason behind that returns to the nature of how Cosine similarity is calculated. As the sizes of titles of Issue and PR are small (c.f., Figure 8, where the average title size is five words), we have less probability of identifying co-located frequency words. This negatively affects the Cosine similarity scores.

2) Body Similarity Results

The results show a kind of agreement in the scores of Jaccard for about 50% of bodies (i.e., a short box-plot). However, these scores are less than the similarity scores of Levenshtein and BERT. This means that bodies have many frequent words. The diversity in the Levenshtein and BERT scores denotes to the variability in the frequency of words over the different bodies. This is justified by the long box-plots of Levenshtein and BERT.

The similarity scores obtained by Levenshtein are a little bit higher than the ones obtained by BERT. The median similarity scores are 33% and 29.7% for Levenshtein and BERT, respectively. The reason behind the aforementioned results is due to the nature of the input texts and how Levenshtein and BERT scores are computed. For example, we have high Levenshtein and BERT scores as the textual representation of bodies for Issues and their linked PRs have similar words in terms of spelling and counts and a similar meaning.

The similarity scores obtained by Cosine are poor. Based on their results, 25% of bodies do not have any similarities. Furthermore, the next 50% of bodies have between zero and 20%.

3) Comment Similarity Results

The similarity scores obtained for Comment are very close to that obtained for Body. The similarity scores obtained by Levenshtein and BERT are higher than the scores obtained by Jaccard and Cosine from the comments of Issues and their linked PRs. And the similarity scores obtained by Levenshtein are varied spread than the similarity scores obtained by BERT. However, the median similarity scores are 24.3% and 22.9% for Levenshtein and BERT, respectively. Moreover, they have very close similarity scores on average of 26% and 25.7% for Levenshtein and BERT, respectively.

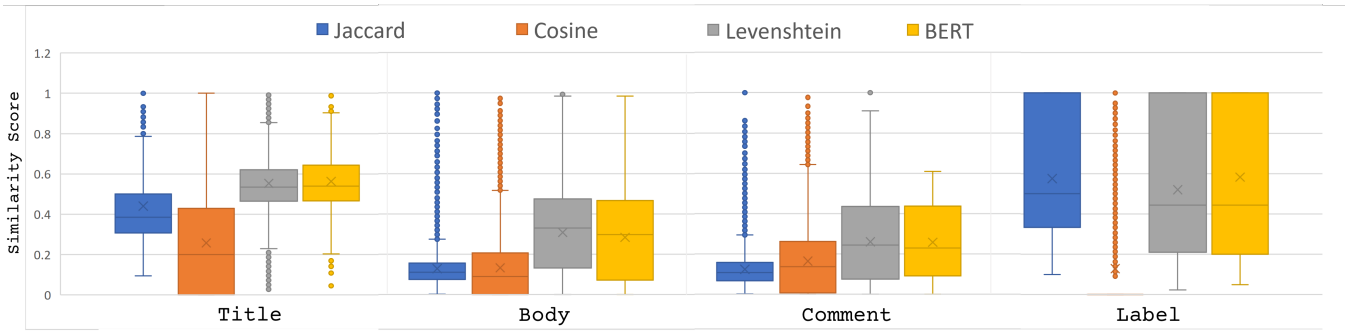

FIGURE 13: The similarity scores between the features of Issues and the features of their linked PRs.

The similarity scores obtained by Jaccard and Cosine are poor. Based on their results, the average similarity scores between the comments on Issues and their linked PRs are 12.4% and 16.4%, respectively. Moreover, more than 75% of their similarity scores are spread between zero and 26%. The most remaining similarity scores are spread between 26% and 30%, whereas a few score cases spread between 30% and 66.6%. This is due to the fact that the contributors that comment on the Issues may have different roles than the contributors who comment on the PRs. For example, Issue comments could describe a new feature's specifications, whereas the comments on PRs related to that Issue describe the implementation evaluation for that feature.

4) Label Similarity Results

The similarity scores obtained for the Label are very different to that obtained from other features. The similarity scores obtained from the labels of Issues and their linked PRs by Jaccard are the highest. Moreover, the similarity scores obtained by Levenshtein and BERT are high too. The median similarity scores are 50%, 44.4% and 44.4% by Jaccard, Levenshtein and BERT, respectively. Moreover, they have very close average similarity scores of 57.5%, 52% and 58.3% by Jaccard, Levenshtein and BERT, respectively. This is due to the repetition of using the labels (see Figure 12). There are many labels that are well-known by social collaborators and used frequently to label Issues and PRs.

We can observe that the similarity scores obtained by Cosine are not normally distributed, where most similarity scores are considered as out-layers. This is because of the short number of labels for Issues and their linked PRs. After investigation, we found that many Issues and PRs have one or two labels (see Figure 11), and this negatively affects the scores obtained by Cosine.

VIII. THREATS TO VALIDITY

During this research work, we identified a set of concerns as threats to validity.

A. INTERNAL THREATS

- In GitHub, many (sub)projects may be hosted in the same repository (called monorepo). If the monorepo includes non-Android projects in addition to an Android project, the proposed scraper tool fetches all projects together, and stores them in the dataset as target projects. Therefore, some PR-Issue links in the dataset are not from Android projects. However, after a focused inspection, this case does not frequently occur where only 6% of all PR-Issue links are from non-Android projects.
- In our dataset, there are many projects with only one PR-Issue link which give an impression that such links are the reason behind the promising obtained results. However, these links form just 8% of the dataset's links which will not impact the obtained results. Also, either single and multiple PR-Issue link projects are from the same subject (Android projects), which means that textual information of single PR-Issue links have similar vocabularies with others.

B. EXTERNAL THREATS

- The ground-truth dataset is collected around only Android applications in GitHub. In fact, we limit ourselves to Android applications because domain-based dataset support establishing a recommender system in that subject (Android) in contrast to the general dataset. Moreover, the domain-based dataset is to guarantee a bias-free dataset for their uses (e.g. especially in Machine Learning) because it shares a lot of similar vocabularies.
- In this paper, the proposed scrapper tool is used to collect PR-Issue links only from Android applications. This gives the first-glance impression that the proposed scrapper tool can not be generalize to other types of applications. In fact, the proposed scraper tool works with any type of application hosted on GitHub.

IX. USE CASES

Among other use cases, we discuss in this section three prominent use cases to show how practitioners can use our dataset to perform other research.

A. IDENTIFY MISSING PR-ISSUE LINK

In GitHub, PR-Issue links are usually manually established by contributors (developers), specifically in large projects. However, many incidents show that contributors might forget

or be lazy to establish these links. There are only a tiny share of PR-Issue links are established, but a large portion of links are missed in the development history. However, manually recovering PR-Issue links from evolutionary development history is a challenging, time-consuming, and error-prone task, even for expert developers.

For this challenge, our dataset can be used with ML and deep-learning approaches to extract the missing PR-Issue links. It can be used to feed the learning models and make them able to predicate the missing links. For example, our dataset can be used to determine the appropriate features that are useful to build a PR-Issue clustering model that aims to group PRs and Issues that should be linked with each other.

B. IDENTIFY DOCUMENTATION-CODE TRACEABILITY LINK

Recovering the traceability links between documentation and source code, especially in legacy systems, is important for various software engineering tasks. It is mandatory for software comprehension, software maintenance and analysis, systematic software reverse, and software collaboration. The major challenge is to develop a methodology to recover these types of links which are rarely explicit. However, manually recovering Documentation-Code links from large software projects is a boring, challenging, time-consuming, and error-prone task.

In this case, our dataset can help to define the relationships between the code and its documentation. It can be used to learn from ground-truth Documentation-Code links to recover the missing ones. We can consider that the textual description of our PR-Issue links represents the documentation and the code represented by commits. Therefore, we can select the appropriate features that can be used for recovering commit code and their textual documentation.

C. RECOMMENDATION SYSTEM SOCIAL CODING PLATFORMS

On a practical level, our dataset can be used by social coding platforms (e.g. GitHub) to recommend PR-Issue links for current software development. Our dataset can be helped to develop integrated tools for these social coding platforms that can assist the developer to easily establish these links during development. This kind of tool will help to reduce the number of missing links that contain much valuable information about software development and the evolution of current development, and document experiences and knowledge for future development.

X. RELATED WORK

Links can be recovered from various software artefacts like requirements, source code and documentation [25]. Many approaches are proposed to recover the links between software artefacts based on bug tracking (e.g., Bugzilla) and version control (e.g., Git) systems. There are many platforms that store these links between different software artefacts like GitHub and *Bitbucket*. These platforms record much information like PR-Issue links, Issue-Commit links, Bug-Commit links, and Documentation-Code links. Consequently, many research and tools have been proposed to identify this type of information. To the best of our knowledge, it has yet to be proposed a ground-truth dataset for PR-Issue links.

In this section, we review the most closely related work in two aspects: the ground truth datasets for the links between software artefacts, and the applications and tools proposed based on these kinds of ground truth datasets.

A. GROUND-TRUTH DATASET

A ground-truth dataset for linking bugs with their bug-fixes code (commits) is proposed by Bachmann et al. [26]. The dataset was extracted from *The Apache HTTP web server* project manually based on heuristic methods. It contains 256 bugs that are linked with 472 commits. However, the dataset is considered small and it was extracted only from one project. Therefore, the features that are extracted from this dataset have not been evaluated on other Bug-Commit links placed on other projects. Bird et al. [27] expand the dataset by adding Bug-Commit links from another two Android projects, *ZXing* and *OpenIntents*. They also manually read change logs and the corresponding code commits to establish links between bugs and their linked commits. The added Android projects result from 236 new Bug-Commit links. However, the dataset still considers small and confined to a few sample projects.

Mehdi et al. presented a dataset for bots' comments in GitHub Issue and PR based on a manual analysis [28]. The dataset represents 527 bots accounts with at most 100 comments for each account from 5,000 distinct Github accounts. A diverse dataset of Java Bugs is proposed by [18]. The dataset comprises 1,158 bugs and patches classified into eight categories extracted from eight large and popular GitHub Java projects. However, the extraction process of the dataset from GitHub projects is evaluated manually by a few people (at most three persons) that are not involved in these projects. Therefore, it could have faulty information due to the evaluation decisions.

Yue et al. [29] proposed a dataset of duplicate PRs in GitHub using a semi-automatic approach. The data contains 2,323 pairs of duplicate PRs collected from 26 popular open-source projects hosted on GitHub. Moreover, it includes duplicate relations between PRs, the meta-data and reviews of duplicate PRs. Another dataset for duplicate repositories on GitHub was proposed by Spinellis et al. [30]. The dataset presents 30 thousand duplicate, cloned, and forked GitHub projects.

Fry et al. presented a dataset for real and alias developer IDs [31]. The data shows that 14.8 million alias developer IDs extracted from 38 million belong to 5.4 million different developers (2 per developer). This data might be helpful in analyzing developer behaviour. However, the real and alias developer dataset are classified based on some assumptions that could not be valid for some portions of the dataset.

All of the aforementioned ground-truth datasets are extracted manually and based on heuristic methods. However, manually extracted datasets are time-consuming, costly, and error-prone. Moreover, the links between software artefacts are extracted from projects by another person who is not involved in the development of these projects. They are assumed to be links based on heuristic methods but not based on explicit notation provided by projects' contributors. Therefore, some of these links could be faulty. However, our ground-truth dataset is automatically extracted and based on explicit notation by the involved contributors themselves. This makes the links represented by our dataset Reliable and error-free.

B. GROUND-TRUTH DATASET APPLICATION

A full-automation tool named *ReLink* was proposed by Wu et al. [32]. *ReLink* recovered Bug-Commit links based on three heuristics: the bug and its commit resolving time should be close to each other; the similarity of the bug and its commit reports; and the bug and its commit contributor should be the same. The *ReLink* results have been evaluated based on the Bug-Commit links ground-truth dataset (mentioned in the previous subsection) that is proposed by [26] and [27]. Further improvements of *ReLink* were proposed by Bissyande et al. [33] and tested with the same ground-truth dataset.

Several approaches have been proposed to recover Issue-commit links using machine learning techniques. A heuristic approach named *MLink* proposed by Nguyen et al. [34] used code heuristics to recover Issue-Commit links by analyzing the similarity of code fragments depicted in Issue reports and commits. *MLink* also used the ground-truth dataset that is proposed by [26] to measure the accuracy of the results.

Another approach proposed by Le et al. [35] introduced *RCLinker*, a clustering tool to predict Issue-Commit links based on the metadata and the textual similarity for Issues and commits. For this purpose, the authors collected 609 Issues-Commits links dataset that was extracted from six Apache software projects. All these projects use JIRA as the issue-tracking system and Git as the version control. Many approaches have been proposed that use the same *RCLinker* dataset. Sun et al. proposed *FRLink* in [36] and *PULink* in [37] to recover Issue-commit links. *FRLink* filtered out irrelevant source code related to an Issue and added non-source documents as a feature to recover missing similarity links compared with other approaches. *PULink* used positive links (the corresponding commit fixes the Issue) and unlabeled links (links from commits that are not found in the positive links) to achieve better recovery performance. Moreover, Likewise, Rath et al. [38] recovered Issue-commit links by using different metadata and textual similarity features. For evaluation, they extracted their own dataset from different domains that utilized Git and JIRA. The dataset contains 4,182 Issue-commit links that are manually extracted from six projects.

XI. CONCLUSION AND FUTURE WORK

In this paper, we proposed a novel approach to extract a ground-truth dataset for the links between PRs and Issues in collaborative software development repositories on GitHub. We developed the PR-Issue-Scraper tool that extracted 50369 PRs explicitly linked to 34732 Issues from 5742 Android projects on GitHub. We select four features, Title, Body, Comment, and Label, to better understand the relationships between the Issues and their linked PRs. After that, we analyzed the selected features by using four textual similarity metrics frequently used in NLP. We compute the lexical and semantic similarity between each feature pair of Issue and their linked PRs using Jaccard, Cosine, Levenshtine, and BERT. The results show that some feature similarities are sensitive to the text length, whereas other feature similarities are sensitive to the term frequency.

In future work, we intend to use our ground-truth dataset to recover the missing PR-Issue links. We are planning to build a clustering model that can recover these missing links based on the appropriate features. Finally, we are also planning to extend our dataset to include other domain-based projects.

REFERENCES

[1] B. Andam, A. Burger, T. Berger, and M. R. V. Chaudron, "Florida: Feature location dashboard for extracting and visualizing feature traces," in *Proceedings of the Eleventh International Workshop on Variability Modelling of Software-Intensive Systems*, ser. VAMOS '17. New York, NY, USA: Association for Computing Machinery, 2017, p. 100–107. [Online]. Available: https://doi.org/10.1145/3023956.3023967

[2] F. Beck, B. Dit, J. Velasco-Madden, D. Weiskopf, and D. Poshyvanyk, "Rethinking user interfaces for feature location," in *2015 IEEE 23rd International Conference on Program Comprehension*, 2015, pp. 151–162.

[3] B. Bassett and N. A. Kraft, "Structural information based term weighting in text retrieval for feature location," in *2013 21st International Conference on Program Comprehension (ICPC)*, 2013, pp. 133–141.

[4] M. Borg, O. Svensson, K. Berg, and D. Hansson, "Szz unleashed: An open implementation of the szz algorithm - featuring example usage in a study of just-in-time bug prediction for the jenkins project," in *Proceedings of the 3rd ACM SIGSOFT International Workshop on Machine Learning Techniques for Software Quality Evaluation*, ser. MaLTeSQuE 2019. New York, NY, USA: Association for Computing Machinery, 2019, p. 7–12. [Online]. Available: https://doi.org/10.1145/3340482.3342742

[5] K. Herzig, S. Just, and A. Zeller, "It's not a bug, it's a feature: How misclassification impacts bug prediction," in *2013 35th International Conference on Software Engineering (ICSE)*, 2013, pp. 392–401.

[6] Y. Zhou and A. Sharma, "Automated identification of security issues from commit messages and bug reports," in *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, ser. ESEC/FSE 2017. New York, NY, USA: Association for Computing Machinery, 2017, p. 914–919. [Online]. Available: https://doi.org/10.1145/3106237.3117771

[7] Z. Yang, J. Shi, S. Wang, and D. Lo, "Incbl: Incremental bug localization," in *2021 36th IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2021, pp. 1223–1226.

[8] S. H. Tan and Z. Li, "Collaborative bug finding for android apps," in *Proceedings of the ACM/IEEE 42nd International Conference on Software Engineering*, ser. ICSE '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 1335–1347. [Online]. Available: https://doi.org/10.1145/3377811.3380349

[9] K. C. Youm, J. Ahn, J. Kim, and E. Lee, "Bug localization based on code change histories and bug reports," in *2015 Asia-Pacific Software Engineering Conference (APSEC)*, 2015, pp. 190–197.

[10] A. Sajedi-Badashian and E. Stroulia, "Guidelines for evaluating bug-assignment research," *Journal of Software: Evolution and Process*, vol. 32, no. 9, p. e2250, 2020.

[11] P. Bhattacharya, I. Neamtiu, and C. R. Shelton, "Automated, highly-accurate, bug assignment using machine learning and tossing graphs,"

Journal of Systems and Software, vol. 85, no. 10, pp. 2275–2292, 2012, automated Software Evolution. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0164121212001240

[12] T. F. Bissyandé, D. Lo, L. Jiang, L. Réveillere, J. Klein, and Y. Le Traon, "Got issues? who cares about it? a large scale investigation of issue trackers from github," in *2013 IEEE 24th international symposium on software reliability engineering (ISSRE)*. IEEE, 2013, pp. 188–197.

[13] L. Dabbish, C. Stuart, J. Tsay, and J. Herbsleb, "Social coding in github: Transparency and collaboration in an open software repository," in *Proceedings of the ACM 2012 Conference on Computer Supported Cooperative Work*, ser. CSCW '12. New York, NY, USA: Association for Computing Machinery, 2012, p. 1277–1286. [Online]. Available: https://doi.org/10.1145/2145204.2145396

[14] Github Number of Public Repository Search, "https://github.com/search?q=is:public," online, accessed 01/11/2022, November 2022, search result shows more than 41M public repositories.

[15] User search, "GitHub," online, accessed 01/11/2022, November 2022, search result shows more than 100M users.

[16] Github Number of Repository Search, " https://github.com/search," online, accessed 01/11/2022, November 2022, search result shows more than 352M repositories.

[17] B. Dit, M. Revelle, M. Gethers, and D. Poshyvanyk, "Feature location in source code: a taxonomy and survey," *Journal of Software: Evolution and Process*, vol. 25, no. 1, pp. 53–95, 2013. [Online]. Available: https://onlinelibrary.wiley.com/doi/abs/10.1002/smr.567

[18] R. Saha, Y. Lyu, W. Lam, H. Yoshida, and M. Prasad, "Bugs.jar: A large-scale, diverse dataset of real-world java bugs," in *2018 IEEE/ACM 15th International Conference on Mining Software Repositories (MSR)*, 2018, pp. 10–13.

[19] Z. Tóth, P. Gyimesi, and R. Ferenc, "A public bug database of github projects and its application in bug prediction," in *Computational Science and Its Applications – ICCSA 2016*, O. Gervasi, B. Murgante, S. Misra, A. M. A. Rocha, C. M. Torre, D. Taniar, B. O. Apduhan, E. Stankova, and S. Wang, Eds. Cham: Springer International Publishing, 2016, pp. 625–638.

[20] GitHub Inc., "Linking a pull request to an issue," online, accessed 1/10/22, October 2022, https://docs.github.com/en/issues/tracking-your-work-with-issues/linking-a-pull-request-to-an-issue.

[21] GitHub, Inc., "REST API," online, accessed 1/10/22, October 2022, https://docs.github.com/en/rest.

[22] GitHub Inc., "GraphQL API," online, accessed 1/10/22, October 2022, https://docs.github.com/en/graphql.

[23] GitHub Inc. , "GraphQL API," online, accessed 1/10/22, October 2022, https://docs.github.com/en/graphql/overview/resource-limitations.

[24] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "Bert: Pre-training of deep bidirectional transformers for language understanding," *arXiv preprint arXiv:1810.04805*, 2018.

[25] J. Cleland-Huang, O. C. Z. Gotel, J. Huffman Hayes, P. Mäder, and A. Zisman, "Software traceability: Trends and future directions," in *Future of Software Engineering Proceedings*, ser. FOSE 2014. New York, NY, USA: Association for Computing Machinery, 2014, p. 55–69. [Online]. Available: https://doi.org/10.1145/2593882.2593891

[26] A. Bachmann, C. Bird, F. Rahman, P. Devanbu, and A. Bernstein, "The missing links: Bugs and bug-fix commits," in *Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 97–106. [Online]. Available: https://doi.org/10.1145/1882291.1882308

[27] C. Bird, A. Bachmann, F. Rahman, and A. Bernstein, "Linkster: Enabling efficient manual inspection and annotation of mined data," in *Proceedings of the Eighteenth ACM SIGSOFT International Symposium on Foundations of Software Engineering*, ser. FSE '10. New York, NY, USA: Association for Computing Machinery, 2010, p. 369–370. [Online]. Available: https://doi.org/10.1145/1882291.1882352

[28] M. Golzadeh, A. Decan, D. Legay, and T. Mens, "A ground-truth dataset and classification model for detecting bots in github issue and pr comments," *Journal of Systems and Software*, vol. 175, p. 110911, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S016412122100008X

[29] Y. Yu, Z. Li, G. Yin, T. Wang, and H. Wang, "A dataset of duplicate pull-requests in github," in *Proceedings of the 15th International Conference on Mining Software Repositories*, ser. MSR '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 22–25. [Online]. Available: https://doi.org/10.1145/3196398.3196455

[30] D. Spinellis, Z. Kotti, and A. Mockus, "A dataset for github repository deduplication," in *Proceedings of the 17th International Conference on Mining Software Repositories*, ser. MSR '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 523–527. [Online]. Available: https://doi.org/10.1145/3379597.3387496

[31] T. Fry, T. Dey, A. Karnauch, and A. Mockus, "A dataset and an approach for identity resolution of 38 million author ids extracted from 2b git commits," in *Proceedings of the 17th International Conference on Mining Software Repositories*, ser. MSR '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 518–522. [Online]. Available: https://doi.org/10.1145/3379597.3387500

[32] R. Wu, H. Zhang, S. Kim, and S.-C. Cheung, "Relink: Recovering links between bugs and changes," in *Proceedings of the 19th ACM SIGSOFT Symposium and the 13th European Conference on Foundations of Software Engineering*, ser. ESEC/FSE '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 15–25. [Online]. Available: https://doi.org/10.1145/2025113.2025120

[33] T. F. Bissyandé, F. Thung, S. Wang, D. Lo, L. Jiang, and L. Réveillère, "Empirical evaluation of bug linking," in *2013 17th European Conference on Software Maintenance and Reengineering*, 2013, pp. 89–98.

[34] A. T. Nguyen, T. T. Nguyen, H. A. Nguyen, and T. N. Nguyen, "Multi-layered approach for recovering links between bug reports and fixes," in *Proceedings of the ACM SIGSOFT 20th International Symposium on the Foundations of Software Engineering*, 2012, pp. 1–11.

[35] T.-D. B. Le, M. Linares-Vasquez, D. Lo, and D. Poshyvanyk, "Rclinker: Automated linking of issue reports and commits leveraging rich contextual information," in *2015 IEEE 23rd International Conference on Program Comprehension*, 2015, pp. 36–47.

[36] Y. Sun, Q. Wang, and Y. Yang, "Frlink: Improving the recovery of missing issue-commit links by revisiting file relevance," *Information and Software Technology*, vol. 84, pp. 33–47, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950584916303792

[37] Y. Sun, C. Chen, Q. Wang, and B. Boehm, "Improving missing issue-commit link recovery using positive and unlabeled data," in *2017 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE)*, 2017, pp. 147–152.

[38] M. Rath, J. Rendall, J. L. C. Guo, J. Cleland-Huang, and P. Mäder, "Traceability in the wild: Automatically augmenting incomplete trace links," in *Proceedings of the 40th International Conference on Software Engineering*, ser. ICSE '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 834–845. [Online]. Available: https://doi.org/10.1145/3180155.3180207

ZAKAREA ALSHARA received the Ph.D. degree in computer and software engineering from LIRMM and the University of Montpellier, France, in 2016. He is currently an assistant professor in Software Engineering department at Jordan University of Science and Technology. Moreover, he is the team leader of collaboration with the Compact Muon Solenoid (CMS) and the European Organisation for Nuclear Research (CERN) in Geneva, Switzerland. His current research interests include Software maintenance, Software evaluation, Software architecture, and Software modeling. He has published many papers in various international journals, conferences and workshops on these topics.

ANAS SHATNAWI is a senior research engineer at Berger-Levrault, France. He obtained his Ph.D. degree in Computer Science from Laboratory of Computer Science, Robotics, and Microelectronics (LIRMM) of University of Montpellier, France. His research interest is in software reuse, reengineering, reverse engineering and empirical software engineering. He has published many papers in various international journals, conferences and workshops on these topics. Contact him at anas.shatnawi@berger-levrault.com.

HAMZEH EYAL-SALMAN received the Ph.D. degree in software engineering from LIRMM Laboratory and the University of Montpellier, France, in 2015. He is currently an associate professor in Software Engineering department at Mutah University. His current research interests include software product line engineering, software reuse, software maintenance, and feature location. He has published many papers in various international journals, conferences on these topics.

DJAMEL SERIAI He is an associate professor at University of Montpellier and a member of the MaREL team of the LIRMM Laboratory. He is co-head of the Software Engineering Master. SERIAI is a senior software architect with more than 25 years of experience as an engineer, architect, and project manager in software development and maintenance field. and more than 20 years of experience in innovation and related to applied research projects.

MAAD SHATNAWI He is currently an assistant professor in department of Electrical Engineering at Higher Colleges of Technology. His current research interests include Data Mining and Knowledge Discovery, Advanced Machine Learning, Optimization, and Modeling and Simulation. He has published many papers in various international journals, conferences and workshops on these topics.