On the Influence of Biases in Bug Localization: Evaluation and Benchmark

Ratnadira Widyasari, Stefanus Agus Haryono, Ferdian Thung, Jieke Shi, Constance Tan, Fiona Wee, Jack Phan, and David Lo
School of Computing and Information Systems, Singapore Management University
{ratnadiraw.2020,stefanusah,ferdianthung,jiekeshi,hytan.2018,fiona.wee.2018,jack.phan.2018,davidlo}@smu.edu.sg

***Abstract*—Bug localization is the task of identifying part of the source code that need to be changed to resolve a bug report. As this task is difficult, automatic bug localization tools have been proposed. The development and evaluation of these tools rely on the availability of high-quality bug report datasets. In 2014, Kochhar et al. identified three biases in datasets used to evaluate bug localization techniques: (1) misclassified bug report, (2) already localized bug report, and (3) incorrect ground truth file in bug report. They reported that already localized bug report statistically significantly and substantially impact bug localization results, and thus should be removed. However, their evaluation is still limited, as they only investigated 3 projects written in Java. In this study, we replicate the study of Kochhar et al. on the effect of biases in bug report dataset for bug localization. Further investigation on this topic is necessary as new and larger bug report datasets have been proposed without being checked for these biases.**

We conduct our analysis on a collection of 2,913 bug reports taken from the recently released Bugzbook dataset that fix Python files. To investigate the prevalence of the biases, we check the bias distributions. For each bias, we select and label a set of bug reports that may contain the bias and compute the proportion of bug reports in the set that exhibit the bias. We find that 5%, 23%, and 30% of the bug reports that we investigated are affected by bias 1, 2, and 3 respectively. Then, we investigate the effect of the three biases on bug localization by measuring the performance of IncBL, a recent bug localization tool, and the classical Vector Space Model (VSM) based bug localization tool, which was used in Kochhar et al. study. Our experiment results highlight that bias 2 significantly impact the bug localization results, while bias 1 and 3 do not have a significant impact. We also find that the effect sizes of bias 2 to IncBL and VSM are different, where IncBL have a higher effect size than VSM. Our findings corroborate the result reported by Kochhar et al. and demonstrate that bias 2 not only affects the 3 Java projects investigated in their study, but also other in other programming language (i.e., Python). This highlights the need to eliminate bias 2 from the evaluation of future bug localization tools. As a by-product of our replication study, we have released a benchmark dataset, which we refer to as CAPTURED, that has been cleaned from the three biases. CAPTURED contains Python programs and therefore augments the cleaned dataset released by Kochhar et al., which only contains Java programs.

***Index Terms*—Bug Report, Bug Localization, Bias, Python**

I. Introduction

Bugs occur in a project due to errors or mistakes in writing the source code or faults in its design. The presence of bugs in a project may affect the functionalities of the software it releases, causing unexpected behaviors. Bugs may also cause vulnerabilities, rendering the software susceptible to security attacks [1], [2]. Therefore, resolving bugs in a project is a high priority task for developers. Yet, it is not a simple task.

One of the main difficulties in bug resolution is finding the precise location of the bugs in the source code. Bug localization is the task of finding a part of the source code that needs to be modified to resolve a reported bug [3]. This task is difficult, especially in a big project containing a large number of files. To help software developers, automatic approaches and tools for bug localization have been proposed [4]–[8]. Most bug localization tools rely on information retrieval (IR), where given a bug report, the tool finds and ranks the source code files that may contain bugs based on their relevancy [9]. The availability of high-quality bug reports is important in the development and evaluation of these tools.

In 2014, Kochhar et al. [10] highlighted the problems that exist in bug reports dataset that are used to evaluate bug localization techniques. They analyzed three biases in bug reports dataset: (1) wrongly classified issue reports, (2) already localized bug report, and (3) existence of non-buggy files in ground truth. They found that already localized bug report statistically significantly and substantially impact bug localization results. However, their investigation is limited to three Java projects. Moreover, newer datasets and bug localization techniques have been proposed [9], [11]–[13] and utilized [14]–[16] without being checked against these three biases. Thus, further investigation on the impact of biases in bug localization is necessary. In this study, we propose to check the effect of the three biases identified by Kochhar et al. on newer datasets. We replicate the study of Kochhar et al. to analyze the effect of biases in bug report dataset towards bug-localization techniques.

At the time of writing this paper, Python is one of the most popular programming languages based on recent surveys and rankings.[1,2] Contrary to its popularity, research and study on localizing bugs in Python files are fairly limited. Due to the unique features in Python programming language (e.g., duck-typing, usage of heterogeneous collection, etc.), it is likely that the patterns and characteristics of bugs in Python are different than Java. Meanwhile, among the available bug reports dataset, only a small number of bug reports are for Python projects. More recently, Akbar and Kak [9] proposed a bug report

[1]https://octoverse.github.com/
[2]https://insights.stackoverflow.com/survey/2020

dataset named Bugzbook, which contains a collection of Java, C/C++, and Python code from GitHub. However, the bug reports within Bugzbook have not been checked against the three biases.

To address the above problems and analyze the effect of the biases, we develop CAPTURED (CurAted PyThon bUg REport Dataset), a bug report dataset for Python. CAPTURED consists of 9 projects containing 1,809 bug reports that have been cleaned from the three biases. We collect bug reports that fix Python files from the Bugzbook dataset and clean them from the three biases highlighted by Kochhar et al. [10]. Specifically, we ensure that each bug report is (1) correctly classified as such, (2) not already localized, (3) has correct ground truth files. For each bug report in CAPTURED, we provide its identifier, URL, and 3 labels indicating whether the bug report exhibits one of the biases. Our dataset is publicly available.[3] Then, to check the effect of the biases, we compare the performance of two bug localization tools (the VSM tool used in the original study, and a recently released bug localization tool named IncBL [14]) on the CAPTURED dataset (which has been cleaned from biases) against the original bug reports from the same Python projects in Bugzbook (which are still affected by biases).

In order to assess the influence of the aforementioned biases, we ask the following research questions:

- **RQ-1: How prevalent are the three biases?** We seek to investigate whether newer bug report dataset is also affected with biases highlighted by the study of Kochhar et al. [10]. By understanding the prevalence and distribution of the biases within the dataset, we hope to bring more attention to the problem of biases in bug reports dataset for bug localization.
- **RQ-2 : How do the three biases affect the performance of bug localization?** We compare the performance of bug localization conducted on CAPTURED dataset which has been cleaned from biases against all bug reports from the corresponding projects in Bugzbook (which are not cleaned from biases). Through this comparison, we aim to get better understanding on how each bias affects bug localization performance.
- **RQ-3: How does the effect of the three biases in CAPTURED compared to the effect of the three biases in the previous study?** We then compare the result of our investigation with the result of Kochhar et al. study and check whether their claims still hold on newer bug reports dataset.

In summary, our contributions are as follows:

1) We replicate Kochhar et al.'s work by conducting an analysis on the effect of biases in a newer bug report dataset, focusing on projects written in a different (but yet still popular) programming language than the one investigated by Kochhar et al. (Java). Specifically, we analyze bug reports of projects containing Python buggy files in Bugzbook [9].

[3]https://github.com/soarsmu/Python-Bug-Report-Dataset

TABLE I
UNCLEAN FIXED FILES IN BUG REPORTS (ZOOKEEPER-2777)

Summary	There is a typo in zk.py which prevents from using/compiling it.
Description	While trying to create an RPM from zookeeper 3.4.10, I got an error when it tried to compile the file : zookeeper/contrib/zkpython/src/python/zk.py", line 55 """Pretty print(a zookeeper tree, starting at root""") SyntaxError: invalid syntax
Fixed Files	**Nikhil Bhide <nikhilbhide15@gmail.com>** src/contrib/zkpython/src/python/zk.py
Modified Fixed Files	src/contrib/zkpython/src/python/zk.py

2) We release CAPTURED, a bug report dataset for Python that has been cleaned from biases.

II. BACKGROUND

A. IR-based Bug Localization

IR-based bug localization techniques use a bug report (i.e., textual summary and description of the bug report, as shown in Table I) as a query and the source code files as documents. From these inputs, it will produces a list of relevant source code files that correspond to the bug report using an information retrieval technique. The main idea of information retrieval technique is based on the textual similarity between bug reports and source code files, where it is expected that the file that shares many common words with the bug report is more likely to be buggy. Utilizing this idea, IR-based Bug Localization computes the similarity between the bug report and source code files. It will then sorts the relevant files from the most to the least similar to the corresponding bug report.

Generally, the workflow of IR-based bug localization techniques consists of these three steps:

*1) **Source Code Files Pre-Processing and Indexing**:* IR-based bug localization techniques need to extract the textual content of the source code files. Typically, there are three main preprocessing steps that are performed:

- *Normalization*: In this step, we extract comments and identifier names from source code files. Several studies [10], [17] on Java programming language utilize Eclipse JDT tool[4] for this preprocessing. The tool is used to extract the Abstract Syntax Trees (AST) that can be traversed to extract the textual content.
- *Stop word removal*: After we receive the textual content from the normalization step, we need to remove special symbols, punctuation marks, number literals, and common occurring English words such as "the", "he", "have", etc. On top of this, stop word removal for the source code files is also used to remove programming keywords that often appear such as *if, while, for, etc.*
- *Stemming*: Stemming is used to reduce words to their root or base form. For example, words such as "developed", "development" are stemmed to the word "develop". The Porter

[4]http://www.eclipse.org/jdt/

Stemming Algorithm[5] is one of the stemming approaches that have been widely used by previous works [10], [17].

*2) **Bug Report Pre-Processing**:* The preprocessing step for the bug reports is similar to the preprocessing step for the source code files. First, we need to extract the textual content from bug report, which includes the summary and description fields of the bug report. An example of these textual content is shown in Table I. We then break down the textual content into tokens (i.e., bag of words). Finally, we run a stop-word removal and stemming preprocessing steps on these tokens.

*3) **Retrieval and Ranking**:* Many techniques have been proposed to measure the similarity between a query and a set of documents, and then retrieve documents that are similar to the query. One of the techniques that is commonly used among many IR-based bug localization techniques is Vector Space Model (VSM). In VSM, each query and document are presented as a vector of weights. Each weight corresponds to a word in a query or a document. The value of each weight is often computed using *term frequency–inverse document frequency* (TF-IDF). The number of times a word appears in a document is referred to as term frequency (TF). Meanwhile, the number of documents in the set of documents that contain the corresponding word is referred to as inverse document frequency (IDF). Higher value of TF-IDF for a given word indicates that the word is more important. TF-IDF is then computed using the following formula:

$$\textit{TF-IDF}(w, d, D) = log(\textit{freq}(w, d)) \times log \frac{|D|}{|d_i \in D : w \in d_i|}$$

where w is a word, d is a document, D is a set of documents, $\textit{freq}(w, d)$ is the number of times word w appear in document d, and $d_i \in D : w \in d_i$ refers to documents that include the word w.

Using the vectors of weights from the query and document, VSM calculates the similarity between the two vectors. The similarity is calculated by using a standard metric such as cosine similarity. The following is the equation to compute the cosine similarity between query q and document d:

$$sim(q, d) = \frac{v_q \cdot v_d}{\|v_q\| \, \|v_d\|}$$

where v_q and v_d are vectors of term weights for query q (i.e., bug report) and document d (i.e., source code file), respectively.

B. Evaluation Metrics

The output of IR-based bug localization is a list of files ranked from the most to the least suspicious. There are several metrics that can be used to evaluate the effectiveness of the bug localization techniques based on the rank of the buggy files in the ranked list. In this study, we use the metrics that are commonly used in the evaluation of bug localization techniques in previous works [10], [17], [18]:

- **MAP (Mean Average Precision)**: This metric is computed by averaging the average precision (AP) of all bug reports. Higher value of MAP indicates better effectiveness of the bug localization technique. The following formula is used to compute the average precision of a single bug:

$$AP = \sum_{i=1}^{M} \frac{P(i) \cdot rel(i)}{\#\textit{buggy files}}$$

where M is the number of source code files and $rel(i)$ is a binary value that indicates whether the i^{th} file is the buggy file. $P(i)$ is the precision at Top-i ranked files, which can be calculated using the following formula:

$$P(i) = \frac{\#\textit{buggy files in top } i}{i}$$

- **HIT@N**: It is the percentage of bug reports whose buggy files can be found in the Top-N of the ranked list. Following the previous study [10], we use 1 as the value of N.
- **MRR (Mean Reciprocal Rank)**: This metric is calculated based on the reciprocal ranking (i.e., the inverse of the rank) of the first buggy file in the ranked list. The MRR is computed by averaging the reciprocal rank of all bug reports. The formula of MRR is defined as a follow:

$$MRR = \frac{1}{|B|} \sum_{i=1}^{B} \frac{1}{rank_i}$$

where B is a set of bug reports and $rank_i$ is the rank of the first buggy file from the ranked list.

III. The Three Biases and Their Prevalence

In this section, we investigate our first research question **RQ1**: **How prevalent are the three biases?** At the time of writing this paper, several new bug report datasets have been proposed. BuGL [11] is a dataset for bug localization consisting of projects written in C, C++, Java, and Python. The main focus of BuGL dataset is to investigate similarities and differences between bugs in different programming languages. Bugzbook [9] is a diverse bug report dataset for IR-based bug localization containing a collection of Java, C/C++, and Python code from GitHub. These newer datasets are not curated by removing the three biases that may skew the bug localization result [10]. Only limited manual checks were conducted on the bug reports: BuGL dataset was only checked by exploring the connections between GitHub issues and pull requests based on descriptions and keywords. Authors of Bugzbook only manually checked two random bug reports from each of the projects by comparing their collected bug reports with the online entry in GitHub or Jira. None of the currently available bug report datasets have been checked thoroughly to ensure that they are free from biases that have been pointed out by Kochhar et al [10]. Thus, in this RQ, we aim to check whether the bug reports in newer dataset contain these biases. We intend to draw greater attention to the bias problem in the bug reports dataset by examining the prevalence and distribution of the three biases within a newer dataset.

We focus our study on the Python programming language, as it is currently one of the most popular programming

[5]http://tartarus.org/martin/PorterStemmer/

language but only has limited number of bug report dataset. We choose to utilize the Bugzbook dataset that is built by Akbar and Kak [9], as it contains bug reports for Python code and has a bigger number of bug reports (i.e., 21,253 bug reports) compared to the BuGL dataset (i.e., 10,000 bug reports). Bugzbook links the bug reports to the corresponding source code files that are fixed or modified to address the bug reports. We filter the bug reports by only taking the bug reports that fix Python files. Through this filtering process, we obtain a total of 2,913 bug reports from 12 projects. However, we find that there are some bug reports where the fixed files entry contains a link to the developer's email, as shown in Table I. As this entry is not a buggy file, we conduct an additional step to manually check the pushed commit that fix the bug, and remove any unrelated entry in the fixed files, such as the email in the shown example.

Following Kochhar et al.'s work on bias in bug localization [10], we proceed to clean the filtered bug report dataset from three biases: (1) misclassified report; (2) already localized bug reports; (3) incorrect ground truth.

A. Bias 1 - Misclassified Reports

An issue report may describe a bug, improvement request, documentation change, etc. Herzig and Zeller [19] highlighted that there are many issue reports that are misclassified as a bug. In their study, they showed that the misclassification greatly impacts the experiments that determine whether a file is possibly buggy or not based on the history of past bugs. The misclassification may also affect the result of bug localization as there may be different characteristics between different type of issue reports (e.g., bug report, documentation, enhancement request, etc.). In this step, we label whether an issue report is indeed a bug report. We follow the work of Herzig et al. [20] which considers an issue report as a bug report if it complies with the following points:

1) It reports a Null Pointer Exception (NPE).
2) It fixes runtime or memory issues caused by defects (e.g endless loops).
3) The discussion concludes that the code had to be changed semantically to perform a corrective maintenance task.

We manually analyze the bug reports obtained from Akbar and Kak's work [9], which comprise of 2,913 bug reports from 12 projects. We read the summary and description of each bug report to determine whether the report is indeed a bug report. An example of a misclassified bug report is shown in Table II. In this example, the bug report should be classified as an RFE (Request For Enhancement). This bug report contains a request to give an additional option to separate the *DBA* mode. It is clear that this is a feature request and not a bug report.

To prevent bias and ensure the labeling quality, at least two authors independently label each bug report. We determine a bug report as misclassified if and only if all the authors who label the report reach a consensus that the report is not actually a bug report. For bug report which both authors label as not misclassified, we include them as the not misclassified bug report, amounting to 2,367 bug reports. Meanwhile, for bug reports which both authors label as misclassified, we include them as the misclassified bug reports, amounting to 121 bug reports. We do not include bug reports where both authors do not reach the same label, as we want to have a high confidence and accuracy in our dataset. From the first bias labeling, 2,367 bug reports are not misclassified and 121 bug reports are misclassified.

TABLE II
MISCLASSIFIED ISSUE REPORTS (AMBARI-11237)

Summary	Provide option on Ranger UI for separate DBA mode
Description	On Ranger Admin UI, provide a checkbox "Do DB Setup?". If this option is set, BE will setup the DB User and the DB. If not set, it will be assumed that DBA has already done it and those properties will be used to connect to it.
Fixed Files	params.py

B. Bias 2 - Already Localized Bug Reports

An already localized bug report is a bug report where the locations of the files that need to be fixed are already mentioned in the bug report itself. As the bug report already contains the location of the bug fix, it means that the bug is already localized and does not require bug localization. Previous research on the evaluation of IR-based bug localization does not filter out these already localized bug reports. This occurrence may skew the result of bug localization technique. In this subsection, we describe how we filter out the already localized bug reports and show the distribution of the bug reports that are affected by this bias.

In determining the already localized bug reports, we follow the categorization proposed by Kochhar et al. [10]:

- **Fully Localized**: all buggy files are mentioned in the summary and description of the bug report.
- **Partially Localized**: some of the buggy files are mentioned in the summary and description of the bug report.
- **Not Localized**: no buggy files are explicitly mentioned in the summary and description of the bug report.

For the purpose of determining bug reports that are already localized from the dataset, we conduct two steps:

1) Manually Label Bug Reports: From 2,367 bug reports that are not misclassified, we randomly selected 400 bug reports to label. This number is larger than 331, which is the number of statistically representative samples with 5% margin of error and 95% confidence level. We look at the summaries and descriptions of the 400 bug reports, and determine whether the files to fix are already mentioned. To prevent bias and ensure the labeling quality, two authors independently label these bug reports. We then use only bug reports that receive the same labels from the two authors, amounting to 334 labeled bug reports. The purpose of this manual labeling process is to evaluate the performance of the automatic labeling approach that we will explain in the second step.

Examples of bug reports that have been labeled based on their localized properties are shown in Table III, which include not localized, fully localized, and partially localized bug reports, respectively. In the not localized bug report example, it is shown that the buggy file, *params.py*, is not

mentioned in the summary and description of the bug report. Therefore, we determine that this bug report is not localized. In the fully localized example, the names of the buggy files are found in the summary of the bug report. As all the buggy files are mentioned in the bug report, we label the bug report as fully localized. In the partially localized example, we see that one of the buggy files, *TestHeartbeat.py*, is written in the description of the bug report. However, the other buggy file, *TestHostCleanup.py*, does not appear in the summary and description of the bug report. Since not all of the buggy files appear in the bug report, we label the corresponding bug report as partially localized.

2) Automatically Label Bug Reports: We also utilize an automated approach to label the localized bug report by building a Python program to automate the process. The program processes the summary and description from the bug report by tokenizing the text into word tokens. We use space, tab, newline, backslash, and several commonly used symbols (e.g., ":", ";", "/", etc.) as the tokenization separators. Then, we check whether the fixed file names (without their location) exist in the word tokens. We keep the file extension since some file names (without its extension) are often named using words that frequently appear in bug reports, such as "*util*" and "*test*". If all of the file names exist in the words tokens, we label the bug report as fully localized. If only some of the file names exist in the word tokens, we label the bug report as partially localized. Finally, if the file names do not exist at all in the word token, the bug report is categorized as not localized.

To check the quality of the automatically assigned labels, we run the automatic process on the 334 bug reports with has been manually labeled, as described in Section III-B1. By treating the manual labels as the ground truth, we find that the accuracy of the automatic labeling is 99.40%. The 0.6% differences between the manual and automatic labeling processes are due to the bug reports that have the fixed file name written in the summary or description but without the file extension (e.g., utils instead of utils.py). The output from the manual label is partially localized whereas the output from our automatic label is not localized. Since the difference is very small, we can apply this automatic labelling program to the 2,367 bug reports that are not misclassified. From the automatic labeling, 521 bug reports are labeled as fully localized, 15 as partially localized, and 1,831 as not localized. Throughout the dataset, around 22% of the bug reports are already localized. This demonstrates that a large number of bug reports are already localized and would not benefit from bug localization.

C. Bias 3 - Incorrect Ground Truth

Within the Bugzbook dataset [9], all of the files that are changed in the commit as a response to a bug report (i.e., changed files) are considered as the ground truth to evaluate the accuracy of bug localization techniques. However, these changed files might not contain the reported bug. We call these files which do not contain bug-fix as non-buggy files. These non-buggy files need to be excluded from the study as they may skew the bug localization result.

```
diff --git a/ambari-server/src/main/.../hdfs_namenode.py
b/ambari-server/src/main/.../hdfs_namenode.py
--- a/ambari-server/src/main/.../hdfs_namenode.py
+++ b/ambari-server/src/main/.../hdfs_namenode.py
@@ -385,5 +385,5 @@ def is_namenode_formatted(params):
       marked = True
-
+
   if marked:
     return True
```

Fig. 1. Example of file changes. In this changes the newline with trail space is replace with a newline without trailing space.

In this step, we determine whether there are wrongly identified buggy files. If a file change only involves non-essential changes such as refactoring and cosmetic changes, the changed file is regarded as a non-buggy file. We check bug reports that are not already localized and make modifications in more than one file, amounting to 68 bug reports with 137 files. We only check bug reports which contain more than one changed file since if a bug report only involves changes to one file, then the changed file must be the correct ground truth (i.e., the buggy file). Meanwhile, if a bug report involves more than one changed file, it is possible that not all of the changed files are relevant to the bug fix. The total of bug reports that only have one changed file is 1,745 bug reports.

We analyze the changes done in the changed files and also look at the summary and description of the bug report to help in identifying whether the files are indeed relevant to the bug fix. For this purpose, we conduct manual analysis as the automatic identification of real ground truth files that is proposed by a previous study [21] only has a relatively low accuracy (i.e., 71.8% recall and 76.4% precision).

An example of a non-buggy file is shown in Figure 1, which highlights a file that is changed to resolve bug report AMBARI-19402. From the code change diff, we can see that there are only cosmetic changes that did not include bug fixes, as the changes only replace the existing newline with a trailing space with a newline without trailing space. These changes have no effect on the program output or execution. From the labeling process, we find 95 files that contain bug fixes from 64 bug reports and 22 files that do not contain bug-fixes from 20 bug reports. There are 16% of the files that are wrongly classified as a ground truth as these files do not include changes that are relevant to the bug fixes.

D. Overview Distribution

We investigate the prevalence of biases highlighted by Kochhar et al. [10] from the 2,913 bug reports that fix Python files obtained from the Bugzbook [9] dataset. The three biases are (1) misclassified report, (2) already localized bug reports, and (3) incorrect ground truth. The summarized results of our investigations can be seen in Table IV, which shows the number and distribution of each bias.

Initially, we have 2,913 bug reports, from which we have 121 misclassified bug reports and 425 bug reports that receive different labels from the manual labeling process in our investigation of the first bias (i.e., misclassified bug report). Next, from the remaining 2,367 bug reports that are not misclassified, we identify the second bias (i.e., already localized

TABLE III
NOT LOCALIZED BUG REPORTS (AMBARI-10001), FULLY LOCALIZED BUG REPORTS (AMBARI-10258), AND PARTIALLY LOCALIZED BUG REPORTS (AMBARI-11594)

	AMBARI-10001	**AMBARI-10258**	**AMBARI-10548**
Summary	Missing property "Common Name for Certificate" in Advanced ranger-hbase-plugin-properties	after-INSTALL hook **shared_ initialization.py** should call compare_versions not params.stack_version_unformatted	Hadoop Home Directory Should Be Chosen Correctly During An Upgrade
Description	Under HBase config, in Advanced ranger-hbase-plugin-properties section, property "Common Name for Certificate" is missing and needs to be added. This is an optional parameter, but needs to be added for SSL.	Install error compare_versions return invalid type error int(). after-INSTALL hook shared_initialization.py should call compare_versions(params.hdp_stack_version, '2.2') not params.stack_version_unformatted	We currently have a pattern that we use for lib/libexec/bin/sbin ... we should extend this to "home" as well to make our **params.py** consistent.
Fixed Files	params.py	shared_initialization.py	conf_select.py, params.py

TABLE IV
BIASES DISTRIBUTION OVERVIEW FROM 12 PROJECTS IN BUGZBOOK

	Bias 1: Misclassified Issue Reports		
	Not Misclassified		**Misclassified**
#Bug Reports	2,367		121
	Bias 2: Already Localized Bug Report		
	Not Localized	**Partially Localized**	**Fully Localized**
#Bug Reports	1831	15	521
	Bias 3: Incorrect Ground Truth		
	Correct Ground Truth		**Incorrect Ground Truth**
#Bug Reports	1809		20

TABLE V
CAPTURED CLEAN DATASET FROM THREE BIASES

Project	**#Issue Reports**
Ambari	1571
Spark	204
Hive	12
Cassandra	10
Bigtop	4
HBase	3
Solr	3
Sqoop	1
Tez	1

bug reports), where we find 15 bug reports that are partially localized, and 521 bug reports are fully localized. To check the third bias, we remove these partially and fully localized bug reports, and proceed with the remaining 1,831 bug reports. For the third bias (i.e., incorrect ground truth), we manually check 68 bug reports with 137 files. We do not need to check the remaining 1,745 bug reports as these bug reports only have one fixed file, which make them unaffected by the third bias. From the manual analysis, we find that 20 bug reports are affected by incorrect ground truth bias, where we find a total of 22 incorrect ground truth files. There are also 20 files from 8 bug reports where we do not have a consensus in the manual labeling process. The remaining 95 files from 64 bug reports have a correct ground truth or contain bug fixes in them.

After these cleaning processes, we retain 1,809 bug reports that do not exhibit any of the 3 biases. These 1,809 clean bug reports are collected from 9 different projects. The details of this clean dataset, which we refer to as CAPTURED, is shown in Table V. We can see that the biggest percentage of bias is found in the second bias, which is bug reports that are already localized. As the number of bug reports that are affected by the three biases are not small, we encourage researchers to pay more attention to these biases when using or collecting bug reports data for bug localization. In particular, the second bias is the most common but can be filtered automatically.

RQ1 Findings: We find that 121 out of 2,488 (5%) bug reports are affected by bias 1, 536 out of 2,367 (23%) bug reports are affected by bias 2, and 20 out of 68 (30%) bug reports are affected by bias 3. This distribution shows that newer datasets is also affected by the three biases.

IV. EFFECTS OF BIASES ON BUG LOCALIZATION

Following the identification of the biases described in Section III, in this section, we investigate our second research question (**RQ2**): **How do the three biases affect the performance of bug localization?** To better understand the effect of each bias, we conduct several investigations: (1) the effect of misclassified bug reports; (2) the effect of already localized bug reports; and (3) the effect of incorrect ground truth in bug reports. For each bias investigation, we first prepare the evaluation datasets, which consist of the *cleaned dataset* that has been cleaned from the bias and the *dirty dataset* that still contains the bias. Using these datasets, we run the VSM-based (Vector Space Model) bug localization technique that was used in evaluating bias in the previous study [10], and IncBL [14], a recent tool for incremental bug localization.

For evaluating the bug localization results, we compute MAP, HIT@N and MRR, as described in Section II. Following the previous work [10], we use Mann-Whitney-Wilcoxon [22] test (at 5% significance level) and compute Cohen's d effect size [23] to compare the bug localization results when evaluated using the dataset with and without bias. Mann-Whitney-wilcoxon test is used to check how significant the differences between two groups are. Meanwhile, effect size is a metric that shows the magnitude of the experimental effect. Effect size shows how substantial the difference between the bug localization tool results with and without bias are. For the interpretation of the effect size, we follow the suggestion from Cohen [23] where $d < 0.2$ is a 'negligible' effect size, $0.2 \leq d < 0.5$ is a 'small' effect size, $0.5 \leq d < 0.8$ is a 'medium' effect size and $0.8 \leq d < 1.3$ represents a 'large' effect size.

TABLE VI
MAP SCORES OF REPORTED AND ACTUAL DATASETS FOR BIAS 1. REPORTED CONTAINS THE BIAS, WHILE ACTUAL DOES NOT.

Technique	Reported	Actual	Difference	p-value	d
VSM-based	0.195	0.194	-0.31%	0.461	0.002
IncBL	0.265	0.266	0.55%	0.432	0.004

TABLE VII
MANN-WHITNEY-WILCOXON TEST AND COHEN'D VALUES FOR MRR BETWEEN CLEAN AND DIRTY DATASET.

Bias Type	Technique	p-value	d
Bias 1	VSM-based	0.454	0.002
	IncBL	0.442	0.003
Bias 2	VSM-based	$9.56e^{-18}$	0.326
	IncBL	$5.39e^{-15}$	0.292
Bias 3	VSM-based	0.419	0.017
	IncBL	0.453	0.024

A. Effect of Misclassified Bug Reports

From our investigation of RQ-1 for the first bias, we construct two categories of bug reports, which are the bug reports that have been manually checked and labeled as actual bugs (i.e., actual dataset) and bug reports that are classified as bugs in the tracking system but is actually misclassified (i.e., reported dataset). To investigate the effect of misclassified bug reports, we run the IncBL and VSM-based bug localization tools on both categories of bug reports. The MAP scores for both tools on these two categories are shown on Table VI.

We compare the performance of bug localization techniques on the reported and actual datasets. The results show that the differences between MAP scores on the reported and actual datasets are small, with -0.52% and 0.77% differences for the VSM-based and IncBL bug localization techniques, respectively. VSM-based technique achieves a higher score on the reported dataset, while IncBL achieves a higher score on the actual dataset. Furthermore, the result of Wilcoxon statistical test shows that the difference is not statistically significant. The Cohen's d effect size also shows a negligible effect size, meaning that the difference between the reported and actual datasets is not substantial.

Beside MAP scores, we also analyze MRR and HIT@1 scores. Figure 2 shows the effect of bias 1 towards the MRR and HIT@1 scores. We observe that the score differences between the reported and actual datasets for both tools are minimal. Similar to MAP, the MRR score for VSM-based technique is higher on the reported dataset, while the MRR score for the IncBL is higher on the actual dataset. Meanwhile, for the HIT@1 scores, the results for both VSM-based technique and IncBL on the actual dataset achieve have lower scores compared to the scores on the reported dataset. Following the previous study [10], we also run a statistical test and the effect size test for MRR. The comparison of the statistical test result and effect size is shown in Table VII. We observe that the impact of this bias is not statistically significant.

B. Effect of Already Localized Bug Reports

In RQ-1, we have categorized bug reports to already localized, partially localized, and not localized. We run both IR-based bug localization tools (i.e., VSM-based and IncBL) on these bug reports. Then, we evaluate the effectiveness of the bug localization tools for each bug report category.

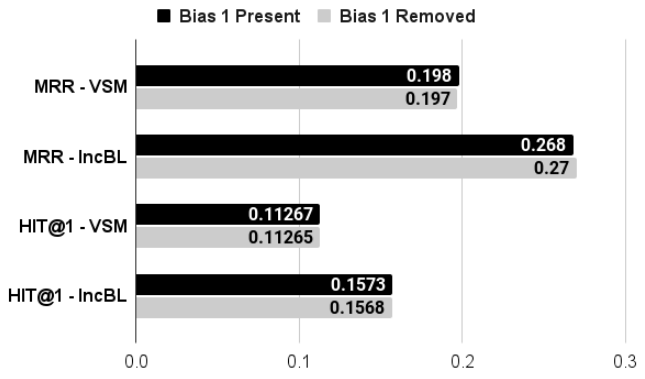

Fig. 2. MRR and HIT@1 scores before and after removing bias 1 for VSM-based and IncBL localization tools

TABLE VIII
MAP SCORES FOR BIAS 2 BETWEEN FULLY LOCALIZED VS PARTIALLY LOCALIZED VS NOT LOCALIZED

Technique	Fully-localized	Partially-localized	Not-localized
VSM-based	0.277	0.122	0.171
IncBL	0.350	0.142	0.243

Table VIII shows the MAP scores of both tools for each set of bug reports, which are fully localized, partially localized, and not localized. The score differences between fully localized and not localized bug reports are 35.82% and 47.29% for the IncBL and VSM-based bug localization tools, respectively. Surprisingly, the MAP results for partially localized are worse than the results for the not localized bug reports, where the score differences between the partially and not localized are 52.79% and 35.69% for IncBL and VSM-based bug localization tools, respectively. The differences between fully localized and partially localized bug reports are the highest, where the differences are 84.61% and 77.88% for IncBL and VSM-based bug localization tools, respectively.

The differences between (1) fully localized vs. partially localized bug reports, (2) partially localized vs. not localized bug reports, and (3) fully localized vs. not localized bug reports are shown in Table IX. Based on these results, we observe that the differences are statistically significant at 5% significance level, except for the partially localized vs. not localized bug reports. Meanwhile, the effect size results show that it spans from negligible (e.g., partially vs not localized with VSM-based) to medium (e.g., fully vs partially localized with IncBL). We also compare the results between fully and partially localized vs. not localized. The results show that the difference is significant with p-value $9.54e^{-18}$ and $7.21e^{-15}$ for VSM-based and IncBL, respectively. This finding shows that, for both VSM-based and IncBL approaches, there are statistically significant differences between already localized and not localized bug reports, as indicated by the p-value that is less than 0.05. For the effect sizes, VSM-based approach have negligible to small effect size, while IncBL have small

to medium effect size. This suggests that bias 2 affects IncBL approach more than it affects VSM-based approach.

Results for HIT@1 and MRR metrics are shown in Figure 3. From the results, we see that the effect of bias 2 is substantial. The HIT@1 and MRR scores for both IncBL and VSM are lower for the dataset with bias 2 removed compared to the one containing bias 2. Contrary to the MAP results, the MRR and HIT@1 results show that partially localized has higher score than not localized. This is due to MAP accounts for the ranking of all buggy files, while MRR and HIT@1 only account for the best localized buggy file. Our finding highlights that the presence of bias 2 in the dataset increases the effectiveness of bug localization approaches. Results of the statistical test and effect size for the MRR are shown at Table VII. The results show that bias 2 affects the bug localization results statistically significantly with a small effect size.

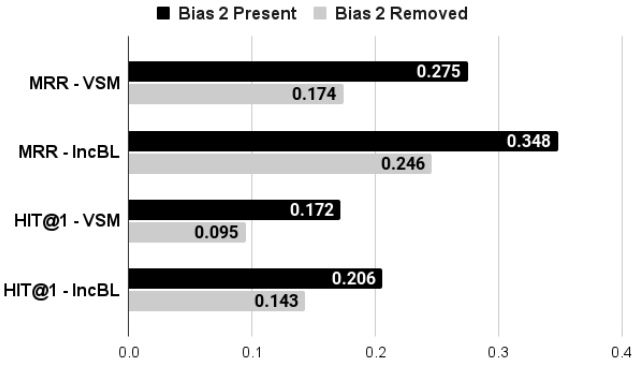

Fig. 3. MRR and HIT@1 scores before and after removing bias 2 for VSM-based and IncBL localization tools

C. Effect of Incorrect Ground Truth in Bug Reports

In the manually identified ground truth files for bias 3, we find that there are 68 bug reports that are not misclassified, not already localized, and have more than one ground truth files. We manually identify whether the files are changed to fix the bug. Based on these identified incorrect ground truth files, we build the dirty and clean ground truth datasets, where the dirty ground truth contains all the changed files (i.e., 117 files), while the clean ground truth only contains the files that are related to bug fixes (i.e., 95 files). We then run VSM-based bug localization and IncBL on these two datasets.

The MAP scores of the VSM-based and IncBL tools for both the dirty and clean datasets are shown in Table X. Based on these results, we observe that the differences in the MAP score between the dirty and cleaned dataset for the VSM-based bug localization tool are higher compared to IncBL, where the differences are 6.35% for VSM and 2.67% for IncBL. However, the statistical test indicates that there are no significant differences between the results on dirty and cleaned dataset for both tools, as indicated by the p-values that are higher than 0.05. Regarding the effect size between the two datasets, both tools have a negligible effect size ($d < 0.2$).

We further investigate the detailed results of each project to better understand the effect of the incorrect ground truth. The 68 bug-reports that are checked for this bias are taken from Ambari and Spark projects. The detailed results for each project are shown on Table XI. Based on the IncBL results, we observe that there are no significant differences between the results on the dirty and clean ground truth dataset, as the differences are only 2.91% to 2.94% with a negligible effect size (0.019 and 0.088) for Spark and Ambari, respectively. Different from the IncBL results, the VSM-based results show that the Spark project have a higher differences (39.31%) with a medium effect size. We conclude that the project-level results are consistent with the overall results, where the VSM-based approach results have higher differences between the dirty and clean ground truth dataset compared to the IncBL tool.

MRR and HIT@1 scores for the dirty and clean datasets are shown in Figure 4. Compared to the MRR and HIT@1 results for bias 1, it is apparent that bias 3 has more effect. However, the score differences for bias 3 are not as substantial as bias 2. VSM-based results for both MRR and HIT@1 show a higher score when the bias 3 removed compared to when there are instances of bias 3 in the dataset. However, for IncBL, the results when bias 3 is present are higher. This is inline with the overall MAP results. The statistical test and effect size results on MRR between the dataset with and without bias 3 is shown in Table VII. It shows that the effect is not significant.

RQ2 Findings: Bias 1 and Bias 3 do not significantly and substantially affect bug localization results. Meanwhile, Bias 2 statistically significantly affect bug localization results, with negligible to small effect size for VSM, and small to medium effect size for IncBL.

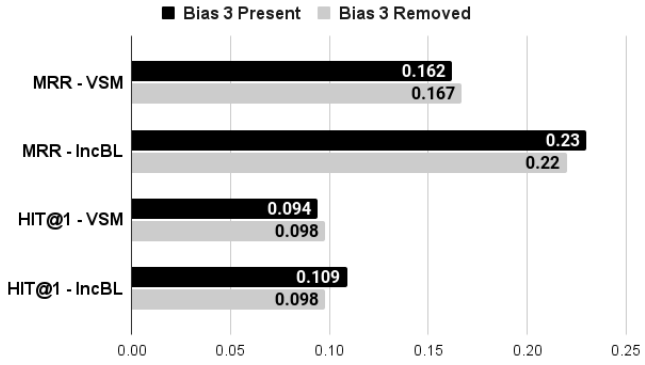

Fig. 4. MRR and HIT@1 scores before and after removing bias 3 for VSM-based and IncBL localization tools

V. Comparison with previous study

In this section, we investigate our third research question (**RQ3**): **How does the effect of the three biases in CAPTURED compared to the effect of the three biases in the previous study?** We want to compare our evaluation results with the evaluation results of bug localization tool that have been done by Kochhar et al. study [10]. To be fair, we only compare results of the VSM-based bug localization

TABLE IX
MANN-WHITNEY-WILCOXON STATISTICAL TEST AND COHEN'S D EFFECT SIZE FROM THE MAP SCORES FOR FULLY VS. PARTIALLY VS. NOT LOCALIZED

Technique	Fully-Partially			Partially-Not			Fully-Not		
	p-value	d	Effect Size	p-value	d	Effect Size	p-value	d	Effect Size
VSM-based	0.030	0.437	small	0.428	0.167	negligible	$2.0e^{-18}$	0.344	small
IncBL	0.014	0.568	medium	0.282	0.303	small	$1.211e^{-15}$	0.309	small

TABLE X
OVERALL RESULTS OF BIAS 3 (MAP)

Technique	Dirty	Clean	Difference	p-value	d
VSM-based	0.122	0.129	-6.35%	0.499	0.032
IncBL	0.183	0.179	2.67%	0.493	0.001

TABLE XI
MAP RESULTS FOR BIAS 3 BETWEEN EACH PROJECT IN DIRTY VERSUS CLEAN GROUND TRUTH

IncBL					
Project	**Dirty**	**Clean**	**Difference**	**p-value**	**d**
Ambari	0.168	0.173	2.91%	0.5	0.019
Spark	0.515	0.5	-2.94%	0.341	0.088
VSM-based					
Project	**Dirty**	**Clean**	**Difference**	**p-value**	**d**
Ambari	0.121	0.127	4.96%	0.486	0.024
Spark	0.145	0.202	39.60%	0.349	0.596

technique that was also used in Kochhar et al. study [10]. As we utilize a different dataset, we cannot directly compare the MAP results between the two studies. Thus, we conduct a qualitative analysis on the evaluation results between our study and Kochhar et al. study, by checking whether the claim in Kochhar et al.'s study regarding effect the of the biases on bug localization are inline with the findings in our study.

- For bias 1, our evaluation shows that the VSM-based bug localization tool is not affected by misclassified bug reports. Compared to the claim from the previous study, which states that bias 1 does not significantly impact the bug localization results (i.e., except for one project), our finding is generally inline with the claim. Our evaluation also highlights that the effect size of bias 1 is negligible, which is also inline with the finding in the previous study.
- For bias 2, previous study claims that already localized bug reports statistically significantly (p-value < 0.05) and substantially (effect size > 0.8) impact bug localization results. Indeed, in our study, bias 2 has a statistically significant impact. However, our evaluation shows that the effect size of bias 2 is not as high as the effect size in the previous study. In our study, the effect size for VSM-based technique spans from negligible to small, whereas in the previous study, the effect sizes span from small to large, with most categories having a large or medium effect size. This finding suggests that bias 2 does affect the result of bug localization. However, the effect size may vary.
- For bias 3, the previous study claims that the incorrect ground truth (i.e., ground truth that contains non-buggy files) does not statistically significantly or substantially impact the bug localization results. Our finding corroborates this claim, as our evaluation shows that the p-value when testing the score differences between dirty and clean ground truth is higher than 0.05 with the effect size less than 0.2. However, when evaluating a project separately (i.e., Spark), we find that the effect size is medium, which differs from the result of the previous study that highlights that all projects have a negligible effect size. Thus, while the overall effect of bias 3 is not significant, researchers should still consider the presence of this bias in their dataset as it may have more impact on some projects.

RQ3 Findings: Based on our evaluation results on the effect of the three biases in bug localization task, we generally agree with the findings of the previous study, where only bias 2 has significant effect. However, the effect size of bias 2 in our dataset is smaller than the previous study.

VI. RELATED WORK

IR-based Bug Localization. Many studies on IR-based bug localization approaches have been proposed [14], [24]–[28]. Akbar and Kak [24] conduct a large-scale comparative evaluation of IR-based tools for bug localization. They highlight that tools that utilize the proximity, order, and semantic relationships are superior than other existing tools. Khatiwada et al. [25] investigate the impact of combining various IR-based method for bug localization. They find that optimized IR-hybrids can outperform individual methods. Scaffle [26] is a scalable IR-based bug localization technique that can handle millions of possible bug locations. Lam et al. [27] propose a combination of deep neural network and rVSM (revised Vector Space Model), an IR-based bug localization technique. Their evaluation shows that the combination of the two techniques can achieve a higher bug localization accuracy. IncBL [14] is a recent tool for incremental bug localization that can provide bug localization for Java, Python, C, and C++ code. IncBL provides up to 4.5 times faster bug localization results while maintaining the same degree of accuracy as similar IR-based bug localization techniques. Pathidea [28] is an IR-based bug localization approach which leverages logs in bug reports to re-construct the execution paths.

In our study, we utilize IR-based bug localization technique to check the effect of biases. We use VSM-based bug localization technique following the work of Kochhar et al. [10]. We also utilize IncBL tool, which is one of the recent IR-based bug localization tool that can work with Python code.

Bug Report Datasets. Several bug report datasets have been released [9], [11]–[13], [29]–[31]. Most of these datasets consist of bug reports for Java and C programming languages. Buglinks [12] is a dataset containing 4,650 Java and 396 C/C++ bug reports from Bugzilla that have been marked as "FIXED". Bench4BL [13] is a collection of 10,017 bug reports

from 46 open-source Java projects. Bugs.jar [30] is a dataset of 1,158 reproducible bugs and patches for Java programs with accompanying bug reports. moreBugs [31] is a dataset of more than 500 bug reports derived from 10 years of commits and development of AspectJ and JodaTime repositories.

More recently, several bug report datasets containing Python code have also been released. BuGL [11] is a dataset of 10,187 bug reports from C, C++, Java, and Python code, which is aimed to find similarities between bugs in different programming languages. Bugzbook [9] is a bug report dataset from a collection of C, C++, Java, and Python code, containing more than 20,000 bug reports for IR-based bug localization. Compared to the above bug report datasets, our work is focused on bug reports for Python files. We also focus our study in the investigation of biases that has been highlighted by Kocchar et al. [10], which have not been thoroughly examined in the bug report datasets from previous studies.

Bias in Software Engineering. Availability of quality dataset is important for software engineering studies. However, the dataset may be affected by bias, which affect the result of studies utilizing such dataset. Several research investigate biases in software engineering dataset [10], [32]–[36]. Herzig et al. [32] examine more than 7,000 issue reports from bug databases and find that 33.8% of the bug reports are misclassified and not actually contain a bug. Bird et al. [33] investigate historical bug dataset and find evidence of systematic bias on these datasets that threatens the effectiveness of processes that rely on these biased datasets. Herzig and Zeller [35] investigate the effect of tangled code change, which is a code change that contains unrelated or loosely related changes. They find that up to 15% of all bug fixes that they investigate consist of multiple tangled changes. Kawrykow and Robillard [36] observe that important changes to software are sometime accompanied by non-essential modifications. They find that up to 15.5% of system's method updates are related to non-essential modifications. Tantithamthavorn et al. [34] study the effect of data mislabelling in the performance of defect prediction models. They find that the mislabelling is not random and the models that are trained using these noisy data have a lower recall. Kochhar et al. [10] investigate the effect of three biases towards bug localization technique. The three biases that they consider are misclassified bug report, already localized bug report, and incorrect ground truth for bug report.

In this study, we replicate the work of Kochhar et al. on the biases for bug localization. We check the effect of biases on bug localization techniques using a newer dataset to reevaluate the previous findings of Kochhar et al.

VII. Threats to Validity

Internal Validity. Threats to internal validity relate to the tools and approaches that we use for bug localization. As the previous study by Kochhar et al. [10] does not publicly share the IR-based bug localization tools that they use, we replicate their tools for Python language based on the descriptions that are provided in their paper. To mitigate this risk, we publicly share our code and implementation of the tool that we replicate so that other researchers can validate our implementation. The code and the dataset that we use are available at: https://anonymous.4open.science/r/Bias-in-BL-0063/

Another concern related to internal validity is in the analysis of the biases. We utilize manual labeling to identify whether the bug reports contain biases. It is possible that, due to human mistake, there are errors in the labelling process. To reduce the likelihood of incorrect labeling, we require two authors to independently label each of the bug reports. Then, we only include bug report where both author give the same label, while we exclude bug report that do not receive the same label. Using this approach, we can have a higher confidence in the quality of the bias labeling and the produced cleaned dataset. We also made our dataset public for others to validate our findings. Thus, we believe that this threat is minimal.

Construct Validity. Threats to construct validity relate to the suitability of evaluation metrics that we use in our evaluation. To evaluate the bug localization result, we utilize MAP, HIT@N and MRR. For the comparison of the metric scores between the datasets, we use Mann-Whitney-Wilcoxon and Cohen's d effect size statistical tests. As these metrics and statistical tests are the same as the one used in the prior study of Kochhar et al. [10], we believe that this threat is minimal.

VIII. Conclusion and Future Work

In this work, we replicate the study of Kochhar et al. [10] on the effect of three biases in the evaluation of bug localization techniques. We utilize a recent Python bug report dataset for our evaluation. We analyze the bug reports in this dataset to check the prevalence of the three biases and find that 121 bug reports are affected by bias 1, 536 bug reports are affected by bias 2, and 20 bug reports are affected by bias 3. We clean the dataset from these biases and construct CAPTURED (CurAted PyThon bUg REport Dataset), a Python bug report dataset that is cleaned from the three biases. To check the effect of the biases in our dataset, we utilize IncBL, a recent bug localization tool, and the VSM-based technique which was used in Kochhar et al. study. For each bias category, we separate the bug reports into cleaned (i.e., already cleaned from the bias), and dirty (i.e., still contains the bias) and measure the performance of both tools on both datasets using MAP.

Our investigation highlights that bias 1 and bias 3 do not have a significant impact, while bias 2 significantly impact the results of bug localization. We also find that our findings are inline with Kochhar et al. study, where they also find that bias 2 significantly affect the results while bias 1 and 3 do not. However, we observe a lower effect size on our dataset than the dataset analyzed by Kochhar et al.

For future work, we plan to expand our investigation for more bug reports and more fault localization tools. Our study is currently limited to bug reports of buggy files written in Python, and two bug localization techniques (VSM and IncBL). We plan to investigate other popular programming languages (e.g., Javascript) and bug localization techniques (e.g., deep learning based techniques) in the future.

REFERENCES

[1] O. Aslan and R. Samet, "Mitigating cyber security attacks by being aware of vulnerabilities and bugs," 09 2017, pp. 222–225.

[2] F. Piessens, "A taxonomy of causes of software vulnerabilities in internet software," 2002.

[3] P. Loyola, K. Gajananan, and F. Satoh, "Bug localization by learning to rank and represent bug inducing changes," in *Proceedings of the 27th ACM International Conference on Information and Knowledge Management*, ser. CIKM '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 657–665. [Online]. Available: https://doi.org/10.1145/3269206.3271811

[4] 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.

[5] S. K. Lukins, N. A. Kraft, and L. H. Etzkorn, "Bug localization using latent dirichlet allocation," *Inf. Softw. Technol.*, vol. 52, no. 9, p. 972–990, Sep. 2010.

[6] M. M. Rahman and C. K. Roy, "Improving ir-based bug localization with context-aware query reformulation," in *Proceedings of the 2018 26th ACM Joint Meeting on European Software Engineering Conference and Symposium on the Foundations of Software Engineering*, ser. ESEC/FSE 2018. New York, NY, USA: Association for Computing Machinery, 2018, p. 621–632. [Online]. Available: https://doi.org/10.1145/3236024.3236065

[7] J. Zhou, H. Zhang, and D. Lo, "Where should the bugs be fixed? more accurate information retrieval-based bug localization based on bug reports," in *2012 34th International Conference on Software Engineering (ICSE)*, 2012, pp. 14–24.

[8] S. Wang and D. Lo, "Version history, similar report, and structure: Putting them together for improved bug localization," in *Proceedings of the 22nd International Conference on Program Comprehension*, ser. ICPC 2014. New York, NY, USA: Association for Computing Machinery, 2014, p. 53–63. [Online]. Available: https://doi.org/10.1145/2597008.2597148

[9] S. A. Akbar and A. C. Kak, "A large-scale comparative evaluation of ir-based tools for bug localization," in *Proceedings of the 17th International Conference on Mining Software Repositories*, ser. MSR '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 21–31. [Online]. Available: https://doi.org/10.1145/3379597.3387474

[10] P. S. Kochhar, Y. Tian, and D. Lo, "Potential biases in bug localization: Do they matter?" in *Proceedings of the 29th ACM/IEEE International Conference on Automated Software Engineering*, ser. ASE '14. New York, NY, USA: Association for Computing Machinery, 2014, p. 803–814. [Online]. Available: https://doi.org/10.1145/2642937.2642997

[11] S. Muvva, A. E. Rao, and S. Chimalakonda, "Bugl – a cross-language dataset for bug localization," 2020.

[12] B. Sisman and A. C. Kak, "Assisting code search with automatic query reformulation for bug localization," in *Proceedings of the 10th Working Conference on Mining Software Repositories*, ser. MSR '13. IEEE Press, 2013, p. 309–318.

[13] J. Lee, D. Kim, T. F. Bissyandé, W. Jung, and Y. Le Traon, "Bench4bl: Reproducibility study on the performance of ir-based bug localization," in *Proceedings of the 27th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2018. New York, NY, USA: Association for Computing Machinery, 2018, p. 61–72.

[14] Z. Yang, J. Shi, S. Wang, and D. Lo, "Incbl: Incremental bug localization," *arXiv preprint arXiv:2106.07413*, 2021.

[15] A. A. Seyam, A. Hamdy, and M. S. Farhan, "Code complexity and version history for enhancing hybrid bug localization," *IEEE Access*, vol. 9, pp. 61 101–61 113, 2021.

[16] D. Barros, F. Horita, I. Wiese, and K. Silva, "A mining software repository extended cookbook: Lessons learned from a literature review," in *Brazilian Symposium on Software Engineering*, 2021, pp. 1–10.

[17] T.-D. B. Le, R. J. Oentaryo, and D. Lo, "Information retrieval and spectrum based bug localization: Better together," in *Proceedings of the 2015 10th Joint Meeting on Foundations of Software Engineering*, 2015, pp. 579–590.

[18] V. Murali, L. Gross, R. Qian, and S. Chandra, "Industry-scale ir-based bug localization: A perspective from facebook," in *2021 IEEE/ACM 43rd International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP)*. IEEE, 2021, pp. 188–197.

[19] K. Herzig and A. Zeller, "The impact of tangled code changes," in *2013 10th Working Conference on Mining Software Repositories (MSR)*, 2013, pp. 121–130.

[20] K. Herzig, S. Just, and A. Zeller, "It's not a bug, it's a feature: How misclassification impacts bug prediction," in *Proceedings of the 2013 International Conference on Software Engineering*, ser. ICSE '13. IEEE Press, 2013, p. 392–401.

[21] F. Thung, D. Lo, and L. Jiang, "Automatic recovery of root causes from bug-fixing changes," in *2013 20th Working Conference on Reverse Engineering (WCRE)*. IEEE, 2013, pp. 92–101.

[22] H. B. Mann and D. R. Whitney, "On a test of whether one of two random variables is stochastically larger than the other," *The annals of mathematical statistics*, pp. 50–60, 1947.

[23] J. Cohen, "Statistical power analysis for the behavioral sciences. lawrence erlbaum associates," *Hillsdale, NJ*, pp. 20–26, 1988.

[24] S. A. Akbar and A. C. Kak, "A large-scale comparative evaluation of ir-based tools for bug localization," in *Proceedings of the 17th International Conference on Mining Software Repositories*, ser. MSR '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 21–31.

[25] S. Khatiwada, M. Tushev, and A. Mahmoud, "On combining ir methods to improve bug localization," in *Proceedings of the 28th International Conference on Program Comprehension*, ser. ICPC '20. New York, NY, USA: Association for Computing Machinery, 2020, p. 252–262.

[26] M. Pradel, V. Murali, R. Qian, M. Machalica, E. Meijer, and S. Chandra, "Scaffle: Bug localization on millions of files," in *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*, ser. ISSTA 2020. New York, NY, USA: Association for Computing Machinery, 2020, p. 225–236.

[27] A. N. Lam, A. T. Nguyen, H. A. Nguyen, and T. N. Nguyen, "Bug localization with combination of deep learning and information retrieval," in *2017 IEEE/ACM 25th International Conference on Program Comprehension (ICPC)*, 2017, pp. 218–229.

[28] A. R. Chen, T.-H. P. Chen, and S. Wang, "Pathidea: Improving information retrieval-based bug localization by re-constructing execution paths using logs," *IEEE Transactions on Software Engineering*, pp. 1–1, 2021.

[29] V. Dallmeier and T. Zimmermann, "Extraction of bug localization benchmarks from history," in *Proceedings of the Twenty-Second IEEE/ACM International Conference on Automated Software Engineering*, ser. ASE '07. New York, NY, USA: Association for Computing Machinery, 2007, p. 433–436.

[30] R. K. Saha, Y. Lyu, W. Lam, H. Yoshida, and M. R. Prasad, "Bugs.jar: A large-scale, diverse dataset of real-world java bugs," in *Proceedings of the 15th International Conference on Mining Software Repositories*, ser. MSR '18. New York, NY, USA: Association for Computing Machinery, 2018, p. 10–13.

[31] S. G. Rao and A. Kak, "morebugs: A new dataset for benchmarking algorithms for information retrieval from software repositories," 2013.

[32] 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.

[33] C. Bird, A. Bachmann, E. Aune, J. Duffy, A. Bernstein, V. Filkov, and P. Devanbu, "Fair and balanced? bias in bug-fix datasets," in *Proceedings of the 7th Joint Meeting of the European Software Engineering Conference and the ACM SIGSOFT Symposium on The Foundations of Software Engineering*, ser. ESEC/FSE '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 121–130. [Online]. Available: https://doi.org/10.1145/1595696.1595716

[34] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, A. Ihara, and K. Matsumoto, "The impact of mislabelling on the performance and interpretation of defect prediction models," in *2015 IEEE/ACM 37th IEEE International Conference on Software Engineering*, vol. 1, 2015, pp. 812–823.

[35] K. Herzig and A. Zeller, "The impact of tangled code changes," in *Proceedings of the 10th Working Conference on Mining Software Repositories*, ser. MSR '13, 2013, p. 121–130.

[36] D. Kawrykow and M. P. Robillard, "Non-essential changes in version histories," in *Proceedings of the 33rd International Conference on Software Engineering*, ser. ICSE '11. New York, NY, USA: Association for Computing Machinery, 2011, p. 351–360. [Online]. Available: https://doi.org/10.1145/1985793.1985842