Noname manuscript No.
(will be inserted by the editor)

An Empirical Study of Issue-Link Algorithms: Which Issue-Link Algorithms Should We Use?

Masanari Kondo · Yutaro Kashiwa · Yasutaka Kamei · Osamu Mizuno

the date of receipt and acceptance should be inserted later

Abstract The accuracy of the SZZ algorithm is pivotal for just-in-time defect prediction because most prior studies have used the SZZ algorithm to detect *defect-inducing commits* to construct and evaluate their defect prediction models. The SZZ algorithm has two phases to detect defect-inducing commits: (1) linking issue reports in an issue-tracking system to possible *defect-fixing commits* in a version control system by using an *issue-link algorithm (ILA)*; and (2) tracing the modifications of defect-fixing commits back to possible defect-inducing commits. Researchers and practitioners can address the second phase by using existing solutions such as a tool called `cregit`. In contrast, although various ILAs have been proposed for the first phase, no large-scale studies exist in which such ILAs are evaluated under the same experimental conditions. Hence, we still have no conclusions regarding the best-performing ILA for the first phase. In this paper, we compare 10 ILAs collected from our systematic literature study with regards to the accuracy of detecting defect-fixing commits. In addition, we compare the defect prediction performance of ILAs and their combinations that can detect defect-fixing commits accurately. We conducted experiments on five open-source software projects. We found that all ILAs and their combinations prevented the defect prediction model from being affected by missing defect-fixing commits. In particular, the combination of a natural language text similarity approach, Phantom heuristics, a random forest approach, and a support vector machine approach is the best way to statistically significantly reduced the absolute differences from the

Masanari Kondo, Yutaro Kashiwa, Yasutaka Kamei
Principles of Software Languages group (POSL)
Kyushu University, Japan
E-mail: {kondo, kamei, kashiwa}@ait.kyushu-u.ac.jp

Osamu Mizuno
Software Engineering Laboratory (SEL)
Kyoto Institute of Technology, Japan
E-mail: o-mizuno@kit.ac.jp

ground-truth defect prediction performance. We summarized the guidelines to use ILAs as our recommendations.

1 Introduction

Just-in-time defect prediction models help in identifying whether a commit (i.e., source code changes) is likely to be defective [39]. A just-in-time defect prediction model has several advantages compared with traditional file/package-level defect prediction models [39, 48, 50]; for example, it can provide faster feedback. Hence, numerous prior studies have investigated just-in-time defect prediction models [29, 37, 39, 41, 48, 50, 75, 86].

The SZZ algorithm [69] is a de facto standard algorithm to prepare a dataset to construct and evaluate a just-in-time defect prediction model. The key concept is to link issue reports corresponding to defects and commits that fixed such defects. Such linked commits (a.k.a. *defect-fixing commits*) are used to identify commits that induced changes that needed to be fixed (a.k.a. *defect-inducing commits*). Researchers and practitioners usually use such defect-inducing commits to construct a defect prediction model and evaluate it in terms of the accuracy of predicting defect-inducing commits.

Hence, the accuracy of the SZZ algorithm may affect the reliability of the evaluation of defect prediction performance. For example, Bird et al. [16] reported that the defect-fixing commits linked with issue reports are biased; such biased defect-fixing commits result in underperformance in defect prediction. Kim et al. [42] reported that defect prediction models are durable to the false-positive/negative defect-inducing commits up to a certain threshold (20%), though those over the threshold have a significant effect on prediction performance.

Nowadays, the SZZ algorithm can detect defect-inducing commits accurately with existing solutions [30, 36, 53, 56]; however, even if we use such solutions, false-positive/negative defect-*fixing* commits may induce false-positive/negative defect-inducing commits. Hence, the SZZ algorithm needs an implementation to detect defect-fixing commits accurately.

To improve the accuracy of detecting defect-fixing commits, many prior studies have proposed various *issue-link algorithms (ILAs)* that use several *criteria* to detect defect-fixing commits [10, 12, 17, 18, 26, 49, 55, 63, 69, 71–74, 82, 84, 85]. For example, Wu et al. [84] proposed an ILA called `ReLink`. This approach uses three criteria: (1) the difference between the resolved dates of issue reports and the dates of commits; (2) the consistency of developers; and (3) text similarity.

However, two challenges remain in prior studies: *dataset inconsistency* and *small comparisons*. More specifically, prior studies evaluated their ILAs on different datasets (data inconsistency). In addition, they overlooked the comparison across their proposed ILAs with the majority of prior ILAs. To present the effectiveness of their proposed ILAs (state-of-the-art approaches), they only compared their ILAs with a few conventional ones (small comparisons). We discuss the details in Section 3.3. Owing to these two challenges, it is difficult to conclude the best-performing ILA in terms of the accuracy of detecting defect-fixing commits and the impact to the defect prediction performance.

In this paper, we compared all criteria that were used by the previous ILAs as our ILAs and their combinations on the same dataset. As we divided the previous ILAs into some criteria, our comparison covers not only previous ILAs, but also other combinations. More specifically, we compared 10 criteria as our ILAs (i.e., time filtering, natural language text similarity, natural language text similarity with word association, message generation from source code, loners heuristics, phantom heuristics, modified text files, PU learning, random forest, and support vector machine) in terms of the accuracy of detecting defect-fixing commits. To collect these ILAs, we conducted a systematic literature study with the *snowballing approach* [83]. In addition, we investigated the impact of the ILAs and their combinations to defect prediction performance in terms of the absolute differences to the *ground-truth defect prediction performance*. The ground-truth defect prediction performance is measured in the dataset where almost all defect-fixing commits are already detected accurately. The details of such datasets and defect-fixing commits are discussed in Section 5.1. We conducted our experiments on five open-source software projects from the Apache Software Foundation: the Avro [3], Tez [7], ZooKeeper [8], Chukwa [4], and Knox [6] projects.

Our ultimate goal is to clarify which ILA or combination of ILAs detects the most defect-fixing commits and prevents the defect prediction model from being affected by missing defect-fixing commits compared with the baseline ILA (i.e., used by the SZZ algorithm) called the *keyword extraction*. To achieve this goal, we investigated the following two research questions.

RQ1: **Which issue-link algorithm is the best to detect defect-fixing commits?**
Motivation: Many prior studies have proposed ILAs to detect defect-fixing commits accurately. However, no studies have conducted a large empirical comparison across ILAs. In this RQ, we compared 10 ILAs. Our goal is to identify ILAs that detect defect-fixing commits accurately.
Results: The time filtering approach and the natural language text similarity approach recovered the statistically significantly largest number of missing defect-fixing commits compared with the other ILAs in different projects. The random forest approach achieved the statistically significant highest precision in 22 out of 25 results.

RQ2: **Which issue-link algorithm is the best to prevent a defect prediction model from being affected by missing defect-fixing commits in defect prediction?**
Motivation: Researchers and practitioners should carefully select an ILA if ILAs prevent a defect prediction model from being affected by missing defect-fixing commits. In this RQ, we studied how ILAs and their combinations affect defect prediction performance.
Results: All ILAs including the combinations of ILAs that detect defect-fixing commits accurately result in a statistically significant reduction in the impact to defect prediction performance compared with the baseline ILA, the keyword extraction approach. These ILAs are robust to the datasets including missing defect-fixing commits. In particular, the combination of the natural language

text similarity, Phantom heuristics, random forest, and support vector machine approaches is the best method to prevent the defect prediction performance from being affected by missing defect-fixing commits.

Our results provide researchers and practitioners who study/use defect prediction or investigate defect-fixing commits with guidelines to choose the best ILA for their purpose. We recommend using the combination of the natural language text similarity, Phantom heuristics, random forest, and support vector machine approaches to remove the bias of missing defect-fixing commits on defect prediction performance. If researchers and practitioners want to investigate defect-fixing commits on a dataset in which no false-positive defect-fixing commits exist, we recommend using the random forest approach. If researchers and practitioners need more defect-fixing commits to investigate while allowing false-positive defect-fixing commits, we recommend using the time filtering or natural language text similarity approach. In addition, before using any ILAs, we recommend using the dates of the commit and the issue report to remove noise of defect-fixing commits for defect prediction.

The four main contributions of this paper are as follows:

– We have conducted the first large-scale empirical study to evaluate the ILAs on the same experimental setup.
– We have proposed guidelines for the use of ILAs according to the purpose of each study.
– We have implemented all the studied ILAs that were collected by our systematic literature study [45].
– We have conducted a systematic literature study of the ILAs.

We summarized our ILAs and the validation technique in defect prediction as Python packages [45, 46]. In addition, we made the replication package [44]. These packages can be used to replicate/update our experiment.

The organization of our paper is as follows. Section 2 presents a motivating example. Section 3 introduces related work and contextualize our research. Section 4 presents the experimental design. Section 5 presents our methodology. We also explain our studied ILAs in this section. Section 6 presents the results of our experiment. Section 7 discusses these results. Section 8 describes the threats to the validity of our findings. Section 9 presents the conclusion.

2 Motivating Example

2.1 Defect Prediction and ILAs

Firstly, an overview of just-in-time defect prediction and ILAs is depicted in Figure 1. For more details, prior studies such as the study by Kamei et al. [38] may be referred to. Defect prediction mainly consists of three phases: *the data preparation phase*, *the model construction phase*, and *the evaluation phase*.

Data Preparation: This phase prepares the data for defect prediction. The data are (1) software entities (e.g., commits) that are the target of the prediction, (2) the label that indicates whether entities include defects (i.e., defect-inducing entities), and

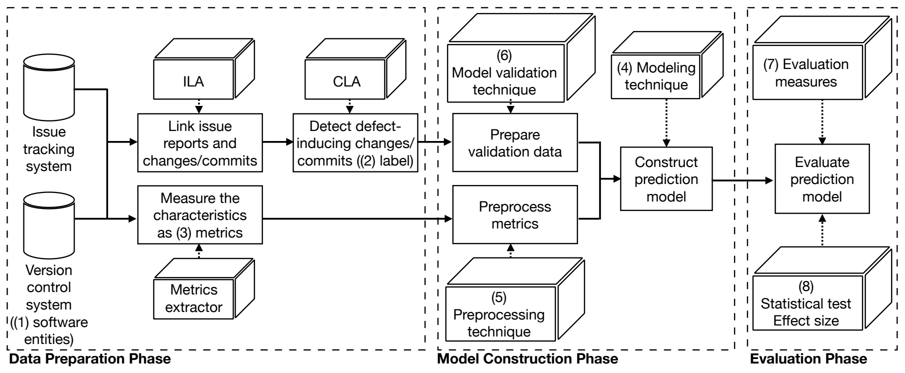
Issue tracking system
Version control system ((1) software entities)
ILA
Link issue reports and changes/commits
CLA
Detect defect-inducing changes/ commits ((2) label)
Measure the characteristics as (3) metrics
Metrics extractor
Data Preparation Phase
(6) Model validation technique
Prepare validation data
(4) Modeling technique
Construct prediction model
Preprocess metrics
(5) Preprocessing technique
Model Construction Phase
(7) Evaluation measures
Evaluate prediction model
(8) Statistical test Effect size
Evaluation Phase

Fig. 1: Overview of just-in-time defect prediction and ILAs.

(3) the metrics that measure the characteristics of entities such as the change metrics [39]. To prepare the label, researchers use two techniques: ILAs and commit-link algorithms (CLAs). ILAs link issue reports to software entities, whereas CLAs find entities that induce defects from entities that are linked to issue reports related to defects (e.g., the SZZ algorithm [69]). All data are collected from two data sources: the issue tracking system (e.g., JIRA) and the version control system (e.g., GitHub).

Model Construction: This phase constructs defect prediction models based on the data that are prepared in the previous phase. To construct the prediction model, researchers need to select (4) modeling techniques (e.g., logistic regression), (5) preprocessing techniques (e.g., z-score), and (6) model validation techniques (e.g., bootstrap-sampling).

First, researchers need to select modeling techniques for defect prediction. Based on the selected modeling techniques, the preprocessing techniques must be decided. Usually, the z-score approach [47] is utilized. However, according to the requirements of the selected modeling techniques, we might choose another preprocessing technique such as the min-max scaling approach [47].

The model validation technique divides the data into the training data and the test data to improve the validity of the evaluation of the prediction models. One technique needs to be selected from the various existing model validation techniques (e.g., bootstrap sampling).

Finally, we construct defect prediction models based on the selected modeling techniques, preprocessing techniques, and model validation techniques.

Evaluation: This final phase evaluates the constructed defect prediction model. Similar to validation techniques, various (7) evaluation measures also exist. Researchers usually evaluate the prediction performance such as precision, recall, and F1-score. In addition, cost-aware evaluation measures are utilized (e.g., Norm(P_{opt})). To evaluate the applicability of defect prediction models to practical scenarios, the execution time might also be evaluated. To evaluate the difference across prediction models, (8) the statistical test (e.g., the Scott-Knott ESD test [80]) and the effect size (e.g., the Cohen's d effect size [20]) are computed.

ILAs studied in this paper are utilized in the data preparation phase. In particular, ILAs link the issue reports extracted from the issue tracking system to the commits/changes extracted from the version control system to prepare the label. This indicates that ILAs are applied as the first step in defect prediction. Hence, ILAs are important because the accuracy of the links (i.e., the label) affects all the phases. Our study will support the improvement of the accuracy of links and improve the reliability of the defect prediction research.

2.2 Do ILAs Affect Defect Prediction?

In Section 2.1, we introduce the accuracy of ILAs is important for defect prediction. Our next questions are that are existing ILAs inaccurate and do such ILAs affect defect prediction? If so, that should be the motivation for our study. In this section, to answer these questions, we introduce false-positive and false-negative defect-fixing commits induced by the most popular ILA, and show a simple survey that clarifies do ILAs affect defect prediction.

False-positive defect-fixing commits indicate defect-fixing commits that are linked with unrelated issue reports while false-negative defect-fixing commits indicate defect-fixing commits that should link with issue reports but do not. Prior studies usually use an ILA called *keyword extraction approach*, which uses regular expressions to identify defect-fixing commits in which commit messages include issue ids. However, this approach induces false-positive/negative defect-fixing commits.

For example, the commit of `cf3318e1b` in the Tez project, which is a studied project in this paper, includes two issue ids, `TEZ-8` and `TEZ-1594`. `TEZ-8` corresponds to a defect-fixing process while `TEZ-1594` does not. The keyword extraction approach links this commit and these two issue reports and refers to this commit as a defect-fixing commit. However, `TEZ-8` is not directly related to this commit. Hence, the commit of `cf3318e1b` is a possible false-positive defect-fixing commit.

Also, the commit message of the commit of `0b74bd5e` in the Avro project, which is also a studied project, does not include any issue ids. Hence, the keyword extraction approach does not refer to this commit as a defect-fixing commit. However, the changed file by this commit includes an issue id (`AVRO-2033`) that corresponds to a defect-fixing process. Hence, this commit is a possible false-negative defect-fixing commit. Therefore, even the most popular existing ILA may induce false-positive/negative defect-fixing commits.

Next, let us show a simple survey that clarifies do ILAs affect defect prediction. We conducted the preliminary survey that has been used by prior defect prediction studies [28, 87]. The procedure of our survey is as follows:

1. We search for studies that use the keyword "defect prediction" and are published in the top venues[1] using Google Scholar.
2. We read the title and the abstract and exclude non-defect prediction studies (e.g., issue report studies) and studies that do not have any PDF links. The remaining

[1] The top venues were defined by Google Scholar Metrics of "Software Systems." The number of top venues is 20 (final access: 2021/11/1). We can find all top venues: `https://scholar.google.com/citations?view_op=top_venues&hl=en&vq=eng_softwaresystems`

studies are considered *defect prediction studies*. We call this set of studies Group A.
3. We read the papers and collect the ILAs that are explicitly written in the papers. Also, we exclude studies that are not change/commit-level defect prediction (a.k.a. *just-in-time defect prediction*) studies The remaining studies are considered *just-in-time defect prediction studies*. We call this set of studies Group B.

The studies collected into Group A and Group B can be found in our Google sheet[2].

Only 16.1% of the prior studies use datasets that were generated using ILAs except for the keyword extraction approach. However, 83.3% of them reuse the publicly available dataset. Our survey collects 112 studies in Group A. The proportion of the studies that explicitly use any datasets that were generated using ILAs except for the keyword extraction approach is only 16.1% (18/112). In addition, 83.3% of them (15/18) reuse the ReLink dataset that is generated by an ILA, ReLink [84]. Since the ReLink dataset was publicly available[3], several prior defect prediction studies used this dataset, which implies that almost all prior defect prediction studies do not consider ILAs but simply use the publicly available dataset or the most popular ILA, the keyword extraction approach.

However, we have another question: how many commits can the keyword extraction approach link with issue reports? If the number of linked commits is high and such links are accurate, we do not need to use any ILAs. Since we have already discussed that the keyword extraction approach induces false-positive/negative defect-fixing commits, we counted the number of linked commits. To answer this question, we investigated all the projects that were used as the target projects in Group B except for the unclear or unreachable projects (e.g., "Mozilla" is used as a target project by prior studies while Mozilla is an organization having several projects, not a project). There are 24 studies in Group B. We used Group B because such linked commits are used in just-in-time defect prediction studies. We applied the following regular expressions that were modified regular expressions of the original SZZ [69] to all commit messages and computed the proportion of commits where commit messages include at least one issue id candidate (i.e., the proportion of linked commits):

– bug[#\s\t] ∗ [0 − 9]+
– fix[#\s\t] ∗ [0 − 9]+
– pr[#\s\t] ∗ [0 − 9]+
– show_bug\.cgi\?id = [0 − 9]+
– \[[0 − 9] + \]

In addition, for the Apache projects, we considered the issue ids that are used in the Apache projects such as CAMEL-{{issue id}} in the Camel project. If the proportion is high, the number of linked commits by the keyword extraction approach is high.

Table 1 lists the proportion of commits in which we can find issue id candidates with the regular expressions in the studied projects in Group B. The number of studied projects without duplication is 58. The gold cell indicates a proportion of over

[2] https://docs.google.com/spreadsheets/d/15C1cFH8KZa594DEJzOM5bx1zurNT7m_GLDcWZsOeYsk/edit?usp=sharing

[3] The dataset was able to be downloaded from the site of https://cse.hkust.edu.hk/~scc/ReLink.htm. However, currently, the download link is expired (2021/10/9).

Table 1: The proportion of commits in which we can find issue id candidates in the studied projects in Group B (58 projects from 24 studies). The numerator is the number of commits in which we can find issue id candidates; the denominator is the number of all commits without merge commits. We cloned all the projects on Oct. 7, 2021.

Project	Proportion (%)
ABINIT	1.1 (142/13,316)
Accumulo	81.6 (6,402/7,844)
ActiveMQ	5.5 (709/12,807)
Amber	5.2 (32/618)
AngularJS	0.8 (97/12,755)
Ant	6.4 (1,068/16,776)
ArgoUML	0.6 (177/28,722)
Eclipse JDT	27.2 (6,799/24,983)
Bitcoin	0.9 (204/22,931)
Buck	0.3 (90/26,497)
Bugzilla	83.4 (12,036/14,424)
Camel	53.9 (36,245/67,206)
Columba	0.0 (0/369)
LibreOffice	2.3 (12,585/538,108)
Derby	84.6 (6,994/8,269)
Flink	65.2 (24,219/37,143)
Geronimo	37.9 (6,283/16,583)
Gerrit	3.9 (1,338/33,896)
Gimp	12.4 (7,163/57,739)
GWT	0.4 (45/10,598)
Hadoop	28.4 (20,374/71,704)
Hadoop Common	0.3 (99/33,473)
HBase	90.0 (47,655/52,948)
HOOMD-blue	0.1 (16/16,507)
ITK	1.1 (492/44,655)
iText	0.7 (28/4,279)
JDeodorant	0.2 (3/1,696)
Jackrabbit	1.9 (270/14,071)
Jaxen	7.4 (110/1,485)
Jetty	2.5 (494/19,576)
JRuby	11.0 (5,275/48,059)
LAMMPS	0.2 (50/25,274)
libMesh	0.4 (109/25,333)
Liferay Portal	0.0 (232/823,321)
Linux	2.5 (23,811/966,548)
Lucene-Solr	0.3 (174/64,639)
Mahout	51.2 (2,346/4,578)
Maven	19.9 (2,660/13,375)
MDAnalysis	2.7 (172/6,443)
OsmAnd	2.0 (1,262/63,102)
OpenJPA	68.6 (5,071/7,397)
OpenStack	6.3 (15,379/242,338)
PCMSolver	0.3 (5/1,756)
Perl 5	1.4 (1,638/114,320)
Pig	88.2 (5,492/6,227)
PostgreSQL	4.8 (3,797/79,044)
Qt Base	0.7 (379/56,273)
RMG-Py	0.4 (64/14,351)
Rails	1.2 (884/76,060)
Rhino	16.3 (711/4,365)
Spring Framework	0.3 (79/26,776)
Synapse	0.3 (64/19,942)
Tomcat	16.5 (11,265/68,226)
VTK	1.5 (998/64,943)
Voldemort	0.2 (10/4,413)
Xenon	0.1 (2/2,047)
X server	7.7 (1,849/23,940)
XStream	0.2 (8/3,456)

80%. This is because, in this paper, we used projects in which over 80% of commits include at least one issue id candidate as our studied projects. The blue cells indicate a proportion of under 50%. We observe only 5 of 58 projects are over 80%. On the contrary, almost all projects (49/58 projects) are less than 50%.

In summary, these result show that we need ILAs to improve the accuracy and number of links between commits and issue reports. Otherwise, many commits exist that do not correspond to any issues reports. Such commits could potentially be defect-fixing commits that are not detected (false-negative defect-fixing commits). In addition, false-positive defect-fixing commits also exist. Such commits affect defect prediction. Hence, in this study, we investigated the impact of ILAs to defect prediction.

3 Related Work

Locating defect-fixing and defect-inducing commits by using the issue ids in commit/log messages is a common practice in software engineering [10, 12, 17, 18, 21, 26, 27, 49, 55, 63, 69, 71–74, 82, 84, 85]. For example, Fischer et al. [26] applied regular expressions to log messages to retrieve issue ids.

In defect prediction, the SZZ algorithm [69] is the de facto standard approach to detect both defect-fixing and defect-inducing commits by using the issue ids in commit/log messages. This algorithm uses two data sources (i.e., a version control system and an issue-tracking system) and links these data sources to detect defect-fixing commits. Defect-inducing commits are tracked based on the modifications in such defect-fixing commits.

3.1 Challenges of the SZZ Algorithm

The SZZ algorithm has challenges to detect defect-fixing and defect-inducing commits [9, 11, 12, 16, 22, 25, 33, 35, 40, 43].

Multiple-purposes commits: A defect-fixing commit could include modifications that accomplish other purposes apart from fixing defects. Herzig et al. [35] called commits that have multiple purposes *tangled changes*. Such changes affect the SZZ algorithm when detecting defect-inducing commits from defect-fixing commits. Kawrykow et al. [40] found that up to 15.5% of method updates occur by non-essential modifications only. Kim et al. [43] modified the SZZ algorithm to handle not only defect-fixing hunks but also other purpose hunks in a defect-fixing commit. The modified SZZ algorithm improved the accuracy of detecting defect-inducing commits compared with the original SZZ algorithm. Herbold et al. [33] found that half of defect-fixing commits that were detected by the SZZ algorithm are not actual defect-fixing commits.

A small number of detected commits: The SZZ algorithm uses an issue-tracking system to detect defect-fixing commits; however, this approach can only detect a fraction of the defect-fixing commits [9, 11, 12, 16]. For example, Bachmann et al. [11] reported the rate of fixed issue reports that are linked with commits. They found that the rate for the Apache HTTPD project is 43.43%, the Eclipse project is 33.05%, the GNOME project is 38.99%, the NetBeans project is 54.60%, the OpenOffice project is 7.43%, and the BSZKB project is 37.31%. Ayari et al. [9] reported that the heuristic is not sufficient to find links between issue reports and changes. Indeed, our motivating example (Section 2) also reported that only a fraction of commits include issue id candidates. Hence, the SZZ algorithm needs to detect defect-fixing commits based on incomplete information.

In summary, the SZZ algorithm has two challenges: (1) detecting defect-inducing commits based on multiple-purpose defect-fixing commits; (2) detecting defect-fixing commits based on incomplete information.

3.2 Detecting Defect-Inducing Commits Based on Multiple-Purpose Defect-Fixing Commits

To address the first challenge that a defect-fixing commit intends to accomplish multiple purposes or is not related to defects [35, 40, 41, 51, 52, 56, 59], prior studies have proposed several solutions [36, 52–54, 56, 62]. Jung et al. [36] excluded non-fixing hunks from a defect-fixing commit. They identified 11 non-fixing hunk patterns, which can be divided into two categories: *syntactically detectable patterns* and *semantically detectable patterns*. For example, renaming is a non-fixing hunk pattern in the syntactically detectable patterns category. Pan et al. [59] also summarized code patterns in defect-fixing hunks. They found 27 code patterns; the defect-fixing hunks that include one of them account for around 50%. Nguyen et al. [56] called such commits mixed-purpose fixing commits (MFCs). They proposed a tool, Cardo, which achieved an average of 93% precision and 61% recall to detect MFCs. Neto et al. [53] tried to remove refactoring changes by modifying the SZZ algorithm and called this approach *refactoring aware SZZ implementation (RA-SZZ)*. They reported that RA-SZZ removed 20.8% lines that were identified as defective lines by another state-of-the-art SZZ implementation.

`cregit` [30] has been utilized to detect defect-inducing commits from defect-fixing commits including non-source code changes (e.g., style changes). This tool converts a Git repository into a *view repository* in which specified types of files (e.g., Java files) are converted into token per line files. Each token also has an AST type. Hence, we can track the modification at the token level and easily ignore all non-source code modifications (e.g., comments, blanks, and format changes).

However, even if researchers use these previous solutions to detect defect-inducing commits, they need to detect defect-*fixing* commits first. If such detected defect-fixing commits are inaccurate, any of the previous solutions induce false defect-inducing commits. Hence, detecting defect-fixing commits accurately is important in detecting defect-inducing commits to take full advantage of the previous solutions. Hence, in this paper, we focus on the second challenge, detecting defect-*fixing* commits based on incomplete information.

3.3 *Issue-Link Algorithm*: Detecting Defect-Fixing Commits Based on Incomplete Information

To address the second challenge, many studies have attempted to improve the accuracy of detecting defect-fixing commits [10, 12, 17, 18, 26, 49, 55, 63, 69, 71–74, 82, 84, 85]. More specifically, ILAs (issue-link algorithms) have been proposed to link issue reports to commits. For example, Fischer et al. [26] proposed an ILA that extracts issue ids from log messages to link between issue reports and commits.

As we described in Section 1, prior studies that proposed ILAs have two challenges: the data inconsistency and the small comparisons. Table 2 provides an overview of the studied projects (in the "Studied Projects/Organizations" column) and ILAs (in the "Baseline ILAs name" column) that were compared with the proposed ILA (in the "Proposed ILAs name" column) in prior studies. "NaN" indicates that no infor-

Table 2: Data inconsistency and small comparisons in prior studies for ILAs. The numbers in parentheses in the "Studied Projects/Organizations" column indicate the numbers of retrieved projects from the Apache Software Foundation or studied projects.

Reference	Year	Studied Projects/Organizations	Baseline ILAs Name	Proposed ILA Name
Fischer et al. [26]	2003	Mozilla	NaN	NaN
Śliwerski et al. [69]	2005	Mozilla, Eclipse	NaN	SZZ
Bachmann et al. [10]	2009	Apach HTTP Server, Eclipse etc. (6)	NaN	NaN
Bachmann et al. [12]	2010	Apache HTTP Server	NaN	LINKSTER
Bird et al. [17]	2010	NaN	NaN	LINKSTER
Sureka et al. [74]	2011	Apache HTTP Server, WikiMedia	NaN	NaN
Wu et al. [84]	2011	ZXing, OpenIntents, Apache HTTP Server	Traditional	ReLink
Nguyen et al. [55]	2012	ZXing, OpenIntents, Apache HTTP Server	ReLink, BugScout	MLink
Bissyandé et al. [18]	2013	Apache Software Foundation (10)	ReLink, IR techs.	NaN
Le et al. [49]	2015	Apache Software Foundation (6)	MLink	RCLinker
Schermann et al. [63]	2015	Apache Software Foundation (15)	NaN	PaLiMod
Sun et al. [72]	2016	Apache Software Foundation (18)	FRLink, RCLinker	NaN
Sun et al. [73]	2017	CLI, Collections etc. (6)	RCLinker	FRLink
Sun et al. [71]	2017	Apache Software Foundation (12)	FRLink	PULink
Xie et al. [85]	2019	Apache Software Foundation (6)	PULink	DeepLink
Tu et al. [82]	2020	LAMMPS, RMG-PY etc. (9)	Keyword labeling	EMBLEM

mation was available or that the authors did not provide any names with their ILAs (e.g., heuristics). To collect these ILAs, we conducted a systematic literature study with the *snowballing approach* [83]. This is because we want to collect prior studies that proposed ILAs regardless of their venues and years. We observe that prior studies used different studied projects (data inconsistency), and compared their proposed ILAs with only few ILAs (small comparison). As a result, it is difficult to compare their results and conclude on the best-performing ILA in terms of the accuracy of detecting defect-fixing commits and improving defect prediction performance.

3.4 Defect Data Quality in Defect Prediction Research

If an ILA induces false-positive/negative defect-fixing commits, the ground-truth data that is used to train and evaluate defect prediction models would be biased. Indeed, prior studies have investigated the importance of data quality [16, 57]. Nguyen et al. [57] investigated the impact of missing links for a commercial project. They found that even a commercial project, which adheres to strict rules, also provides a biased dataset. Bird et al. [16] reported that the defect-fixing commits that were detected by using an issue-tracking system are not accurate and affect defect prediction performance.

In addition, prior studies have investigated the impact of noisy data on defect prediction performance [34, 42, 60, 61, 78]. Kim et al. [42] reported the impact of the false-positive/negative rate of detected defects by an ILA on defect prediction performance. They found that the proportion of false-positive/negative rates over a certain threshold (e.g., 20%) had a significant effect on defect prediction performance. Ramler et al. [61] studied the noise in a defect dataset. They reported that the prediction performance is not significantly affected by 20% noise. Rahman et al. [60] compared the impact of the bias and sample size on defect prediction performance, reporting

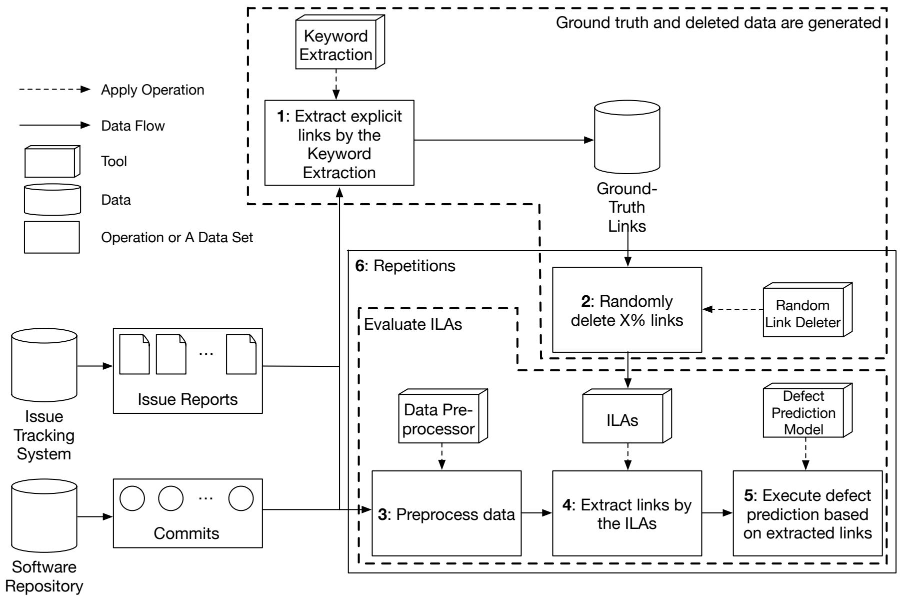

Fig. 2: Overview of our experimental design.

that the sample size is more important than the bias. They found that researchers need to focus more on collecting samples rather than the bias. Herzig et al. [34] reported that bug reports are frequently misclassified (33.8% of bug reports). In addition, they found that 39% of files that are labeled as defective are not defective on average. They also showed that such misclassified data potentially decrease the defect prediction performance. Tantithamthavorn et al. [78] evaluated the impact of mislabeled data on defect prediction. They found that such mislabeled data rarely affect precision values; in contrast, they do affect recall values.

To remedy such biased ground-truth data, we need to improve ILAs. To the best of the authors' knowledge, no studies have conducted large-scale empirical comparisons across ILAs, though many prior studies have proposed various ILAs (as described in Table 2). Hence, we conducted a large-scale empirical comparison across ILAs and evaluated the impact to defect prediction performance. Note that detecting defect-inducing commits based on defect-fixing commits is beyond the scope of this paper. We use a basic approach to detect defect-inducing commits to evaluate the impact to defect prediction performance.

4 Experimental Design

In this section, we give an overview of our experimental design. Figure 2 shows the steps of our experiments. In the following, we describe these steps in detail.

1. Extract explicit links by the Keyword Extraction. The keyword extraction approach uses issue ids in the commit messages to make links between issue reports and com-

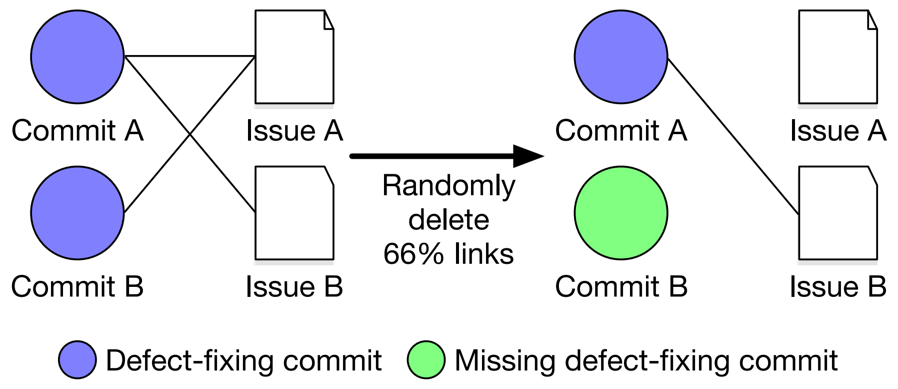

Fig. 3: An example to delete 66% of the explicit links from three of them.

mits. We regard commits that are linked with studied issue reports labeled `Bug` as defect-fixing commits and use them as the ground-truth defect-fixing commits. We call the links of such ground-truth defect-fixing commits *explicit links*.

2. Randomly delete X% links. We randomly deleted *X*% explicit links on our studied datasets (Section 5.1). By randomly deleting explicit links and regarding defect-fixing commits that are only linked with such deleted links as *missing defect-fixing commits*, we can simulate and evaluate a scenario in which datasets have low link proportions. Figure 3 shows an example. Let us assume we have three explicit links (Commit A and Issue A, Commit A and Issue B, and Commit B and Issue A) and delete 66% of the links. We might delete two links: Commit A and Issue A, and Commit B and Issue A. We regard Commit B as a missing defect-fixing commit; Commit A is still a defect-fixing commit because Commit A is still linked with Issue B. We describe the studied *delete rates* (i.e., *X*%) in our results section (Section 6.1 and Section 6.2).

3. Preprocess data. We executed the preprocessing for the ILAs. The missing defect-fixing commits should not have any issue ids on commit messages because we assume that the keyword extraction approach overlooks such commits. Hence, we removed issue ids from the commit messages to conduct a fair comparison when using the commit messages on the missing defect-fixing commits. In addition, we applied a *basic restriction*. We describe the details of this restriction in Section 5.2. The details of the preprocessing for each ILA are given in Appendix A.

4. Extract links by the ILAs. We applied the ILAs to the preprocessed commits and issue reports for each delete rate. When using a delete rate greater than 0%, ILAs are trained on the explicit links without the deleted links if such ILAs need to be trained.

5. Execute defect prediction based on extracted links. We executed the defect prediction on the extracted links. We first used the extracted links to identify defect-fixing commits. We used such defect-fixing commits to identify defect-inducing commits by using the *commit-link algorithm*. We describe the details of our commit-link algorithm implementation in Section 5.3. Based on the defect-inducing commits, we trained the defect prediction model and evaluated the performance across different ILAs.

6. Repetitions. To relieve data selection bias on the deleted links, we repeated steps 2–5. We repeated steps 1–4 100 times while we repeated step 5 20 times. We used the 100 results of step 4 as the RQ1 results and the 20 results of step 5 as the RQ2 results. We employed different times because the execution time of step 5 would be too long to conduct 100 repetitions. We discussed the details of the execution time in Section 6.2.

A running example. Let us describe these steps with an example: the Avro project. In particular, we utilize the commit `a439bf9`. In step 1, the keyword extraction approach forms the links. The commit message of the commit `a439bf9` includes a studied issue id of `AVRO-2741` that is labeled `Bug`. Consequently, the keyword extraction approach links this commit to the issue report of `AVRO-2741` and refers to this commit as a defect-fixing commit. Also, this link is an explicit link. For all the commits in the Avro project (2,728), the keyword extraction approach links 778 commits to issue reports labeled `Bug`. These commits are also defect-fixing commits, and all the links are explicit links.

In step 2, we delete $X\%$ links. If X is zero, no links are deleted. However, if X is not zero, $X\%$ links are deleted. For example, if X is 50, half of the links in the Avro project are randomly deleted. We refer to all commits whose all links are deleted as missing defect-fixing commits. For example, if the link for the commit `a439bf9` is deleted, the commit `a439bf9` is a missing defect-fixing commit.

In step 3, if the commit `a439bf9` is a missing defect-fixing commit, the issue id (i.e., `AVRO-2741`) is excluded from the commit message. We apply this exclusion process to all missing defect-fixing commits. In addition, we also apply the basic restriction to all remaining links to exclude the false-positive links (Section 5.2). The link of the commit `a439bf9` is not the false-positive link; and therefore, it is not the target of the basic restriction.

In step 4, ILAs are applied to all commits and issue reports. If the commit `a439bf9` is a missing defect-fixing commit, ILAs may recover the link between the commit and the issue report of `AVRO-2741`. However, ILAs may form links between the commit and other issue reports as false-positive links. Similarly, ILAs may recover links between any commits and any issue reports.

In step 5, we conduct defect prediction. First, we use all the commits that are defect-fixing and not missing defect-fixing commits, and all the commits that are not defect-fixing but linked to issue reports labeled `Bug` by ILAs to find the corresponding defect-inducing commits. For the commit `a439bf9`, if either the link is not deleted in step 2 or the link is recovered in step 4, this commit is referred to as a defect-fixing commit and it is utilized to find the corresponding defect-inducing commits. Otherwise, this commit is not referred to as a defect-fixing commit even if it is an actual defect-fixing commit. We build a defect prediction model based on the defect-inducing commits.

In step 6, steps 2-5 are repeated. This repetition allows us to study the impact of various deleted links and false-positive links (i.e., the combination of defect-fixing commits, false-positive defect-fixing commits, and missing defect-fixing commits).

Table 3: Overview of studied Apache projects

Project	# Commits	# Merge Commits	# Issues (Type: Bug)	% Linked Commits	% Defect-Fixing Commits	% Defect-Inducing Commits	Latest Abbr. Commit Hash
Avro	2,788	60	908	81.1 (2,213/2,728)	28.5 (778/2,728)	9.9 (269/2,728)	`791ec60`
Tez	3,866	16	1,786	96.3 (3,706/3,850)	53.0 (2,040/3,850)	26.5 (1,019/3,850)	`ba441c1`
ZooKeeper	3,784	6	1,339	84.6 (3,197/3,778)	44.7 (1,689/3,778)	15.5 (586/3,778)	`b5feadc`
Chukwa	1,104	2	340	82.4 (908/1,102)	36.8 (405/1,102)	19.6 (216/1,102)	`38742d2`
Knox	2,856	32	1,047	82.7 (2,336/2,824)	35.1 (992/2,824)	22.3 (629/2,824)	`a821cf3`

For the commit `a439bf9`, we study both cases where the commit `a439bf9` is a missing defect-fixing commit and a defect-fixing commit.

5 Methodology

In this section, we describe our methodology. In particular, we discuss our studied datasets, ILAs, a commit-link algorithm, defect prediction models, evaluation measures, preprocessing steps, a resampling approach, and validation schemes. The tools, data, and operations in Figure 2 correspond to each method.

5.1 Studied Datasets

We used five open-source software projects (the Avro [3], Tez [7], ZooKeeper [8], Chukwa [4], and Knox [6]) from the Apache Software Foundation as our studied datasets. Table 3 describes the basic information of the projects. Avro is a data serialization system. Developers can use Avro to transform raw data into rich binary data. Tez is a framework on Hadoop that allows developers to process data. ZooKeeper is a centralized service for managing distributed systems. Chukwa is a monitoring system for distributed systems. Knox provides developers with an application gateway on Hadoop. As a result, we used two domains (data serialization and distributed system) in this study. We extracted Git repositories on GitHub and issue reports on JIRA for these five projects. The studied data include over 10k linked commits and 5k issue reports.

We chose these five projects because almost all commit messages of the repositories include issue ids on JIRA. The proportion of linked commits (i.e., including issue ids on commit messages) for the Avro project is 81.1%, for the Tez project is 96.3%, for the ZooKeeper project is 84.6%, for the Chukwa project is 82.4%, and for the Knox project is 82.7%; the proportion of defect-fixing commits that are detected by our keyword extraction approach (we describe the details in Section 5.2) for the Avro project is 28.5%, for the Tez project is 53.0%, for the ZooKeeper project is 44.7%, for the Chukwa project is 36.8%, and for the Knox project is 35.1%.

We considered these defect-fixing commits as the ground-truth defect-fixing commits. This is commonly used when evaluating ILAs [11, 71, 72, 74, 85] because prior studies have validated this practice [11, 71, 74]. Prior studies [18, 71, 74] executed a manual inspection to validate the accuracy of their data. Similar to such prior studies, to validate the accuracy of our ground-truth data, we also executed a

Table 4: Overview of studied ILAs

ILA	Abbr.	References
Keyword Extraction	KE	[10, 18, 26, 55, 63, 69, 72, 74, 84, 85]
Time Filtering	TF	[10, 18, 49, 55, 63, 69, 71–74, 84, 85]
Natural Language Text Similarity	TS	[18, 49, 55, 71–74, 84, 85]
Natural Language Text Similarity with Word Association	WA	[55]
Message Generation from Source Code	GS	[49]
Loner Heuristics	LO	[63]
Phantom Heuristics	PH	[63]
Modified Text Files	MT	[72, 73]
PU Learning	PU	[71]
Machine Learning	ML	[49, 85]

manual inspection for both false-positive and -negative defect-fixing commits by two of the authors. We first randomly extracted 361 defect-fixing commits to validate the number of false-positive defect-fixing commits and 367 non-defect-fixing commits to validate the number of false-negative defect-fixing commits from all projects. These numbers are determined by the condition where the confidence level is 95% and the confidence interval is 5. Two of the authors labeled these commits as false-positive/negative defect-fixing commits. The kappa coefficients [67] of this labeling process are 1.000 and 0.971, respectively. For the conflicts between two of the authors, the two of the authors carefully discussed and decided the final label. Given this manual inspection, we found that the accuracy of the defect-fixing commits and non-defect-fixing commits are 99.7% (360/361) and 89.1% (327/367), respectively, which are high accuracy values. Consequently, the ground truth data is reliable. We discuss the false-positive and -negative defect-fixing commits in Section 7.5 and the threats of this manual inspection in Section 8.3.

We studied the Java source code in the studied projects, though the Avro project also provides developers with implementations on multiple languages. Note that we removed merge commits from the studied commits. This is because merge commits only merge existing diff codes and do not add/modify any codes. In addition, we studied issue reports that are labeled `Bug` and the status is either `Resolved` or `Closed`. Note that there exists an issue report whose resolution date is missing. Hence, in our experiment, we used the closed date as a proxy of the resolution date if the resolution date is missing.

5.2 Studied ILAs

We first collected prior studies that propose ILAs. To prevent overlooking such prior studies, we used the snowballing approach [83] that we described in Section 3.3. In particular, when we find a paper that proposes ILAs, we also collect all studies that refer to this study and are referred by this study. This process allows us to collect studies regardless of their venues and years. Also, we used the result of our literature survey that we described in Section 2.2. Finally, we found 16 prior studies (Table 2).

Prior studies combined several criteria (e.g., text similarity) on their ILAs [10, 12, 17, 18, 26, 49, 55, 63, 69, 71–74, 82, 84, 85]. In this paper, we retrieved each of the criteria from the previous ILAs and call them and their combinations ILAs. This is because such criteria are the finest-grained algorithms when linking issue reports to commits, and we can cover not only previous ILAs, but also other combinations of criteria. Note that we exclude the studies that used manual analysis [12, 17, 82]. Table 4 lists all the studied ILAs. We studied 10 ILAs including the baseline ILA (the keyword extraction approach). Note that because we used two models for the machine learning approach, the actual number of studied ILAs is 11. Before applying these ILAs, we applied an essential restriction:

– a linked issue report is created before the date of its linked commits are committed; and
– such a linked issue report is resolved after the date of its linked commits are committed.

All ILAs include this restriction. We call this restriction the *basic restriction*. This restriction reduces the number of false-positive defect-fixing commits. We discussed the details in Section 7.5. All the implementations of ILAs used in this paper can be seen as a Python package [45].

In the following, we give a brief overview of the ideas behinds ILAs. We describe the details of them in Appendix A.

– *Keyword Extraction (KE):* This is a de facto standard approach to identify defect-fixing commits extracting issue ids from commit messages with regular expressions. As described previously, we used the output of this ILA as the ground-truth defect-fixing commits. However, even if we use the projects in which almost all commits include issue ids, linking commits and issue reports is a difficult process. We describe this threat in Section 8.1.
– *Time Filtering (TF):* This approach makes links between commits and issue reports where the date information is within a certain time interval. The main idea is that an issue report might be resolved right after the commit date of the corresponding modification. Interestingly, prior studies used different time intervals. For example, Bachmann et al. [10] used seven days or less; Schermann et al. [63] used five minutes or less. Therefore, it is difficult to determine which time interval should be used. From our preliminary analysis, we decided to use 10 minutes as our time interval. We discuss the details of the preliminary study in Section 7.3.
– *Natural Language Text Similarity (TS):* This approach computes a text similarity value between issue report descriptions and commit messages. If such a similarity value is high, then a pair would be linked. The main idea is that the related issue reports and commits have similar descriptions.
– *Natural Language Text Similarity with Word Association (WA):* This approach also computes a text similarity value between issue reports and commits. However, this approach additionally associates words between issue reports and commits based on their contextual relationship.
– *Message Generation from Source Code (GS):* This approach uses comments of source code instead of commit messages. A prior study [49] used a comment generation technique. They used *javadoc comments* as the supervised data to train the technique, and therefore, we used the javadoc comments instead of using code

comment generation techniques to ensure that clean information is used. The procedure is the same as that in the natural language text similarity approach.
- *Loner Heuristics (LO):* This approach focuses on a scenario in which one commit addresses one issue report. Schermann et al. [63] proposed heuristics of the scenario to identify defect-fixing commits.
- *Phantom Heuristics (PH):* This approach focuses on a scenario in which a set of commits addresses one issue report. Schermann et al. [63] proposed heuristics of the scenario.
- *Modified Text Files (MT):* This approach considers not only commit messages but also modified text files. The main idea is to retrieve additional information from natural language text in modified files.
- *PU Learning (PU):* This approach uses PU (positive and unlabeled) learning [23]. As there might exist many unlabeled links between issue reports and commits, prior study [71] used the PU learning to predict positive links based on such unclear data. To predict positive links, we provided five features with the PU learning approach: the time difference, the time difference type, the cosine similarity of text, the proportion of modified source files, and the number of modified source files. Further details of the features are described in Appendix A.
- *Machine Learning (ML):* This approach applies machine learning models to predict links. Although the PU learning approach predicts positive links based on positive and unlabeled links, this approach predicts positive links based on positive links. We used two machine learning models: a random forest model [64] and a support vector machine model [66]. To predict positive links, we provided five features with the machine learning approach that are also used on the PU learning approach.

Note that, in this paper, we decided not to use the following four ILAs that have been proposed previously:

- *File Filtering Approach [26, 74]*
- *Developer Filtering Approach [63, 69, 74, 84]*
- *Code Similarity Approach [55]*
- *Deep Learning Approach [85]*

The file filtering and code similarity approaches need modified files information (patches). However, the Apache JIRA prohibited such information from being retrieving. The developer filtering approach is a common practice; however, we cannot use such information because of GDPR [24]. The deep learning approach was proposed by Xie et al. [85]. However, many settings are not clear in the paper, such as the details of deep learning architectures. Hence, we decided not to use these approaches in this paper.

5.3 Commit-Link Algorithm

After detecting defect-fixing commits, we need to detect defect-inducing commits. We call this process *commit-link algorithm (CLA)*. We used a basic procedure as our CLA:

1. Apply `cregit` [30] to the target repository. As described in Section 3, `cregit` [30] converts a Git repository into a *view repository* in which specified types of files (i.e., Java file) are converted into token per line files. Each token also has an AST type. Hence, we can easily ignore redundant tokens (e.g., comments).
2. Extract commit hash lists from the target repository. Remove the first commit hash. This is because the first commit is not related to source code in the Avro and ZooKeeper projects and it is difficult to track individual modifications in the Tez, Chukwa, and Knox projects.
3. Apply the `git show`[4] command to extract all changed lines (*added lines* and *deleted lines*) for each commit. Note that the `git show` command classifies all changed (added/deleted/modified) lines in a commit as the *added lines* or the *deleted lines*. A modified line is represented by an *added line* and a *deleted line*.
4. Extract the *deleted lines* for all Java files for each commit, but ignore the *added lines*. This is because the *added lines* are newly added lines in this defect-fixing commit. Hence, such lines do not have any information to detect defect-inducing commits.
5. Remove the *deleted lines* in non-source code (i.e., comments).
6. Apply the `git blame`[5] command to the remaining *deleted lines* to identify the commits where the *deleted lines* were added. We regard the extracted commits as defect-inducing commits.

5.4 Studied Defect Prediction Model

We used the *logistic regression model* as our defect prediction model. As our goal is not to construct an accurate defect prediction model, but to reduce the difference in defect prediction performance from the ground-truth defect prediction performance by using ILAs, we only chose logistic regression. The logistic regression model is frequently used for constructing defect prediction models [13, 32, 39]. This model learns the relationship between a dependent variable and independent variables. In defect prediction, a dependent variable is the flag of commits that indicates whether this commit is defective or clean; dependent variables are the features of commits.

To construct a logistic regression model, we used the scikit-learn implementation [65]. Because it is important to optimize the hyper-parameters of defect prediction models [79], we optimized the hyper-parameter of the logistic regression model. The scikit-learn implementation has two hyper-parameters that can be optimized: the regularization strength C and the norm of the penalty. We optimized these two hyper-parameters in the following ranges: 0 to 10 for C and *l1* and *l2* for the norm of the penalty. We might optimize other hyper-parameters; however, because of the long execution time, we only used these two hyper-parameters. We describe the execution time of defect prediction in Section 6.2. From empirical and theoretical viewpoints, the random search is one of the best optimization methods [15]. Hence, we used the random search to optimize the hyper-parameters of the logistic regression.

[4] https://git-scm.com/docs/git-show
[5] https://git-scm.com/docs/git-blame

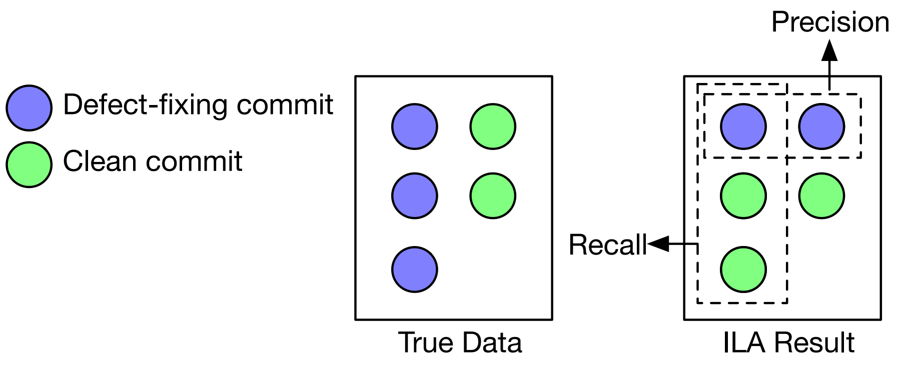

Fig. 4: An example to evaluate the accuracy of detecting defect-fixing commits.

5.5 Evaluation Measures

We evaluated two tasks: the accuracy of detecting defect-fixing commits by the ILAs and the accuracy of detecting defect-inducing commits by the defect prediction model. As each task has different outputs, we used different sets of evaluation measures. In addition, we used a statistical test. In the following, we explain the evaluation measures for each task and the statistical test.

5.5.1 Detecting Defect-Fixing Commits

We used four evaluation measures: precision, recall, F1, and true-positive (TP) rate. The precision indicates the proportion of true defect-fixing commits in all the defect-fixing commits that are decided by an ILA; the recall indicates the proportion of true defect-fixing commits that are identified by an ILA in all the true defect-fixing commits. Here, the true defect-fixing commits indicate the commits that are identified by the explicit links. Let us assume that we have three true defect-fixing commits and two clean commits, and an ILA detects one true defect-fixing commit and one clean commit as defect-fixing commits. In this case, the precision value would be 0.500 (1/2), and the recall value would be 0.333 (1/3) (Figure 4).

The TP rate is used on this task only. In this task, we deleted $X\%$ of the links and have the ILAs recover missing defect-fixing commits. The precision, recall, and F1 values were computed on all the true defect-fixing commits; however, the TP rate was computed on the missing defect-fixing commits only. This is because we want the TP rate to evaluate the accuracy of the ILAs on the missing defect-fixing commits. Let us assume that we have five missing defect-fixing commits. If an ILA detects two missing defect-fixing commits as defect-fixing commits, the TP rate would be 0.400.

5.5.2 Detecting Defect-Inducing Commits

We used six evaluation measures: area under the receiver operating characteristic curve (AUC), precision, recall, F1, Matthews correlation coefficient (MCC), and

Brier score. AUC and Brier score are threshold-independent measures, though the precision, recall, and F1 are threshold-dependent measures (we used 0.5 as the threshold). This is because Tantithamthavorn et al. [76] suggested using threshold-independent measures because threshold-dependent measures may result in different conclusions by different thresholds. However, we also used threshold-dependent measures because such measures show us various viewpoints on the results. We also used a threshold-dependent measure, MCC, because prior studies reported that MCC is durable to the skewness of defect data [19, 88].

5.5.3 The Scott-Knott ESD test

We used the Scott-Knott ESD test [80] as our statistical test to compare the evaluation measures across ILAs (using a 95% significance level). The Scott-Knott ESD test is an extended version of the Scott-Knott test. The Scott-Knott test is a clustering algorithm that ranks the distributions. If distributions are not statistically significantly different, these distributions are placed in the same rank. The Scott-Knott ESD test ranks the distributions with not only statistically significant differences but also Cohen's d effect size [20]. The distributions that are not statistically significantly different or with negligible effect size are placed in the same rank.

5.6 Preprocessing for Predicting Defect-Inducing Commits

To predict defect-inducing commits, we used the defect prediction model. Thus, we need to transform a commit into a numerical vector representation. The most common representation in commit-level defect prediction (a.k.a. *just-in-time defect prediction*) is metrics-based approaches such as using the *change metrics* [39, 41, 48, 52].

In this paper, we used the change metrics [39, 48] to transform a commit into a numerical vector representation and evaluate the ILAs. We used Commit Guru [62] to calculate the change metrics. We transformed the change metrics to remove correlated features and normalize the features following a previous study [48]:

- Exclude ND and REXP because they are strongly correlated with NF and EXP.
- LA is replaced by $\text{LA}_{\text{new}} = \text{LA}/\text{LT}$ if LT is not zero.
- LD is replaced by $\text{LD}_{\text{new}} = \text{LD}/\text{LT}$ if LT is not zero.
- LT is replaced by $\text{LT}_{\text{new}} = \text{LT}/\text{NF}$ if NF is not zero.
- NUC is replaced by $\text{NUC}_{\text{new}} = \text{NUC}/\text{NF}$ if NF is not zero.

Finally, we apply the z-score [47] to the processed change metrics. Note that we decided not to apply z-score to FIX because FIX is a binary metric.

5.7 Resampling Approach

When learning the model, the learning performance might be affected by imbalanced data [75]. Prior studies [1, 14, 77] recommend using the following resampling approaches: random under-sampling, SMOTUNED, and MAHAKIL. In particular, SMOTUNED and MAHAKIL are state-of-the-art approaches.

Table 5: Parameter values of the online change classification for each project (days).

Project	**Start gap**	**End gap**	**Gap**	**Unit (test interval)**	**Training interval**	**Iteration step size**
Avro	30	647	243	30	1530	51
Tez	30	450	51	30	990	33
ZooKeeper	30	606	211	30	1830	61
Chukwa	30	531	140	30	1590	53
Knox	30	544	147	30	1230	41

To remove the affection of imbalanced data, we compared the three approaches in defect prediction and selected one of them for our study. Because we used the resampling approach in RQ2, we evaluated the impact of these different approaches on the defect prediction performance in the same experimental setting as RQ2 except for the repetition times. Owing to the long execution time, we used 10 repetitions. Given the result, we found that SMOTUNED is the best resampling approach in our study. Hence, we employed SMOTUNED. We only applied SMOTUNED to training data because we must use raw test data for evaluation.

5.8 Validation Schemes

We need to relieve data selection bias on the deleted links. If we used a set of deleted links, our result may be affected by which links are deleted. Therefore, we repeated the process of deleting links 100 times for each delete rate. For each process, we computed the evaluation measures of detecting defect-fixing commits in RQ1. Also, we used 20 of them in RQ2 to compute the evaluation measures of detecting defect-inducing commits. Finally, we computed the median evaluation measures across 100 repetitions in RQ1 and 20 repetitions in RQ2. When applying the Scott-Knott ESD test, we considered the values of an evaluation measure for 100/20 repetitions as a distribution for each ILA.

When evaluating the accuracy of detecting defect-inducing commits (just-in-time defect prediction), we also need to relieve data selection bias on the training data and test data. Cross-validation techniques or bootstrap-sampling techniques [80] are frequently used. However, just-in-time defect prediction is studied on sequential data. We must use past commits/changes to train the model without any information from the future commits/changes. Thus, we used *online change classification*, which satisfies this restriction. Online change classification was originally proposed by Tan et al. [75], and Kondo et al. [48] formalized the parameters. We provide a Python package of the online change classification [46]. Table 5 lists the parameter settings of the online change classification. We used the same process with prior work [48].

6 Results

6.1 RQ1: **Which Issue-Link Algorithm is the Best to Detect Defect-Fixing Commits?**

Motivation and Approach: In recent years, several prior studies [18, 63, 71, 72, 74, 85] focused on recovering missing links rather than detecting missing defect-fixing commits. A missing link indicates a link between a commit and an issue report that is not detected by the KE approach. A missing defect-fixing commit is a commit that fixes a defect but is not detected by the KE approach.

Our main aim is to evaluate the ability of the ILAs in terms of detecting **missing defect-fixing commits** rather than detecting missing links. This is because we want to contribute to defect prediction rather than recovering missing links.

In this experiment, we deleted the explicit links of 10% to 50% in steps of 10%. We considered the deleted explicit links as missing links and commits that are only linked with such missing links as missing defect-fixing commits. We evaluated how many missing defect-fixing commits are detected by the ILAs.

Results: Table 6 shows the median values of the evaluation measures for the 100 repetitions; the row indicates an ILA, and the column indicates an evaluation measure. The cells show not only the median values of the evaluation measures, but also the ranks in the parentheses that were computed by the Scott-Knott ESD test across the ILAs. The gold cells indicate the highest rank (= 1). Owing to space limitations, we only show the delete rates of 50% and 10%.

Observation 1) ***The TF approach generally statistically significantly outperformed the other ILAs in the case where the delete rate is 50%.*** Tables 6(a), 6(c), 6(e), and 6(g) list the results on the datasets with the delete rate of 50% in the Avro, Tez, ZooKeeper, and Chukwa projects. The TF approach achieved the highest rank on recall, F1, and TP rate in the delete rate of 50% except for F1 in the Chukwa project. In addition, the rank on F1 in the Chukwa project is the second rank. This result implies that the TF approach recovers the largest number of missing defect-fixing commits (recall and TP rate) in almost all projects, whereas the number of false-positive defect-fixing commits (not defect-fixing commits, but identified by the ILA) is moderate (F1). However, the TF approach ranked fourth or fifth in terms of precision in these four projects. Hence, even if the TF approach achieved the highest F1 rank, we need to be aware of false-positive defect-fixing commits when using the TF approach. Finally, the TF approach did not achieve high ranks in the Knox project on the three evaluation measures (Table 6(i)). Hence, the TF approach generally statistically significantly outperformed the other ILAs while projects exist in which the TF approach does not work well.

Observation 2) ***The TF approach statistically significantly outperformed the other ILAs in terms of the TP rate for all delete rates except for the Knox project.*** Except for the Knox project, all the results in Table 6 show that the TF approach achieved the highest rank in terms of the TP rate. We observed the same results in the other delete rates as well. This result implies that the TF approach can recover the most missing defect-fixing commits for not only the delete rate of 50% but for all the delete rates in many projects.

Table 6: The median values of the evaluation measures with the Scott-Knott ESD test results on detecting missing defect-fixing commits for each ILA. The delete rate is 50% (left) and 10% (right).

(a) The Avro project (50%)

ILA	Precision	Recall	F1	TP rate (deleted)
TF	0.757 (5)	0.821 (1)	0.788 (1)	0.824 (1)
TS	0.865 (4)	0.517 (4)	0.647 (3)	0.517 (3)
WA	0.117 (6)	0.271 (6)	0.164 (8)	0.123 (6)
GS	0.000 (7)	0.000 (10)	0.000 (9)	0.000 (9)
LO	0.817 (5)	0.236 (7)	0.366 (5)	0.471 (4)
PH	0.937 (3)	0.549 (3)	0.692 (2)	0.094 (7)
MT	0.598 (5)	0.372 (5)	0.458 (4)	0.374 (5)
PU	0.535 (5)	0.628 (2)	0.580 (3)	0.624 (2)
RF	0.993 (1)	0.103 (9)	0.188 (7)	0.055 (8)
SVM	0.986 (2)	0.127 (8)	0.225 (6)	0.104 (7)

(b) The Avro project (10%)

ILA	Precision	Recall	F1	TP rate (deleted)
TF	0.757 (5)	0.821 (2)	0.788 (3)	0.820 (1)
TS	0.865 (4)	0.517 (6)	0.647 (6)	0.519 (5)
WA	0.078 (8)	0.404 (7)	0.130 (8)	0.190 (8)
GS	0.000 (9)	0.000 (10)	0.000 (10)	0.000 (10)
LO	0.520 (7)	0.046 (9)	0.084 (9)	0.463 (6)
PH	0.940 (2)	0.917 (1)	0.929 (1)	0.163 (9)
MT	0.598 (6)	0.372 (8)	0.458 (7)	0.371 (7)
PU	0.882 (3)	0.682 (4)	0.769 (4)	0.675 (3)
RF	0.964 (1)	0.766 (3)	0.854 (2)	0.690 (2)
SVM	0.940 (2)	0.566 (5)	0.706 (5)	0.545 (4)

(c) The Tez project (50%)

ILA	Precision	Recall	F1	TP rate (deleted)
TF	0.847 (4)	0.776 (1)	0.810 (1)	0.777 (1)
TS	0.609 (6)	0.149 (6)	0.239 (5)	0.149 (6)
WA	0.071 (10)	0.443 (4)	0.123 (6)	0.296 (4)
GS	0.200 (9)	0.001 (10)	0.003 (10)	0.002 (9)
LO	0.888 (3)	0.159 (5)	0.270 (4)	0.318 (3)
PH	0.960 (1)	0.624 (2)	0.757 (2)	0.246 (5)
MT	0.255 (8)	0.058 (7)	0.095 (7)	0.057 (7)
PU	0.219 (7)	0.551 (3)	0.313 (3)	0.546 (2)
RF	1.000 (5)	0.004 (9)	0.007 (9)	0.001 (9)
SVM	0.955 (2)	0.040 (8)	0.077 (8)	0.031 (8)

(d) The Tez project (10%)

ILA	Precision	Recall	F1	TP rate (deleted)
TF	0.847 (4)	0.776 (2)	0.810 (3)	0.779 (1)
TS	0.609 (6)	0.149 (7)	0.239 (5)	0.150 (8)
WA	0.042 (9)	0.604 (4)	0.079 (7)	0.441 (5)
GS	0.200 (8)	0.001 (10)	0.003 (9)	0.000 (10)
LO	0.654 (5)	0.031 (9)	0.060 (8)	0.316 (6)
PH	0.962 (1)	0.945 (1)	0.954 (1)	0.454 (4)
MT	0.255 (7)	0.058 (8)	0.095 (6)	0.055 (9)
PU	0.254 (7)	0.566 (5)	0.350 (4)	0.560 (3)
RF	0.911 (2)	0.748 (3)	0.821 (2)	0.715 (2)
SVM	0.905 (3)	0.223 (6)	0.357 (4)	0.213 (7)

(e) The ZooKeeper project (50%)

ILA	Precision	Recall	F1	TP rate (deleted)
TF	0.908 (5)	0.771 (1)	0.834 (1)	0.774 (1)
TS	0.952 (4)	0.328 (6)	0.488 (4)	0.329 (4)
WA	0.046 (9)	0.445 (4)	0.084 (9)	0.341 (3)
GS	0.364 (8)	0.002 (10)	0.005 (10)	0.002 (8)
LO	0.846 (6)	0.090 (8)	0.163 (7)	0.181 (5)
PH	0.968 (3)	0.665 (2)	0.788 (2)	0.328 (4)
MT	0.505 (7)	0.331 (5)	0.400 (5)	0.331 (4)
PU	0.855 (6)	0.600 (3)	0.686 (3)	0.602 (2)
RF	1.000 (1)	0.062 (9)	0.117 (8)	0.037 (7)
SVM	0.990 (2)	0.120 (7)	0.214 (6)	0.104 (6)

(f) The ZooKeeper project (10%)

ILA	Precision	Recall	F1	TP rate (deleted)
TF	0.908 (6)	0.771 (2)	0.834 (3)	0.769 (1)
TS	0.952 (4)	0.328 (8)	0.488 (6)	0.331 (7)
WA	0.029 (10)	0.557 (6)	0.055 (8)	0.476 (6)
GS	0.364 (9)	0.002 (10)	0.005 (10)	0.000 (9)
LO	0.575 (7)	0.017 (9)	0.033 (9)	0.174 (8)
PH	0.970 (2)	0.960 (1)	0.965 (1)	0.596 (4)
MT	0.505 (8)	0.331 (7)	0.400 (7)	0.335 (7)
PU	0.933 (5)	0.681 (4)	0.787 (4)	0.679 (3)
RF	0.987 (1)	0.729 (3)	0.839 (2)	0.690 (2)
SVM	0.964 (3)	0.584 (5)	0.727 (5)	0.578 (5)

(g) The Chukwa project (50%)

ILA	Precision	Recall	F1	TP rate (deleted)
TF	0.730 (5)	0.706 (1)	0.718 (2)	0.704 (1)
TS	0.951 (4)	0.430 (3)	0.592 (4)	0.432 (3)
WA	0.271 (8)	0.323 (4)	0.297 (6)	0.164 (7)
GS	0.000 (9)	0.000 (8)	0.000 (9)	0.000 (10)
LO	0.709 (6)	0.109 (7)	0.188 (8)	0.217 (5)
PH	0.958 (3)	0.600 (2)	0.739 (1)	0.204 (6)
MT	0.663 (7)	0.272 (5)	0.385 (5)	0.268 (4)
PU	0.775 (5)	0.594 (2)	0.663 (3)	0.582 (2)
RF	1.000 (1)	0.142 (6)	0.248 (7)	0.054 (9)
SVM	0.984 (2)	0.149 (6)	0.259 (7)	0.111 (8)

(h) The Chukwa project (10%)

ILA	Precision	Recall	F1	TP rate (deleted)
TF	0.730 (5)	0.706 (2)	0.718 (4)	0.701 (1)
TS	0.951 (3)	0.430 (7)	0.592 (6)	0.439 (4)
WA	0.171 (8)	0.472 (6)	0.251 (8)	0.229 (7)
GS	0.000 (9)	0.000 (10)	0.000 (10)	0.000 (8)
LO	0.404 (7)	0.022 (9)	0.042 (9)	0.214 (7)
PH	0.963 (2)	0.938 (1)	0.950 (1)	0.366 (5)
MT	0.663 (6)	0.272 (8)	0.385 (7)	0.263 (6)
PU	0.896 (4)	0.652 (3)	0.753 (3)	0.630 (2)
RF	0.965 (1)	0.625 (4)	0.758 (2)	0.477 (3)
SVM	0.950 (3)	0.541 (5)	0.689 (5)	0.487 (3)

(i) The Knox project (50%)

ILA	Precision	Recall	F1	TP rate (deleted)
TF	0.867 (7)	0.451 (4)	0.594 (4)	0.452 (3)
TS	0.913 (6)	0.755 (1)	0.826 (1)	0.755 (1)
WA	0.290 (9)	0.412 (5)	0.340 (5)	0.156 (7)
GS	0.000 (10)	0.000 (10)	0.000 (9)	0.000 (10)
LO	0.929 (4)	0.187 (7)	0.311 (6)	0.376 (4)
PH	0.977 (3)	0.583 (3)	0.730 (3)	0.166 (6)
MT	0.400 (8)	0.002 (9)	0.004 (8)	0.002 (9)
PU	0.962 (5)	0.665 (2)	0.779 (2)	0.659 (2)
RF	1.000 (1)	0.090 (8)	0.164 (7)	0.047 (8)
SVM	0.990 (2)	0.213 (6)	0.350 (5)	0.189 (5)

(j) The Knox project (10%)

ILA	Precision	Recall	F1	TP rate (deleted)
TF	0.867 (6)	0.451 (7)	0.594 (5)	0.447 (5)
TS	0.913 (5)	0.755 (2)	0.826 (2)	0.759 (1)
WA	0.207 (9)	0.627 (6)	0.312 (6)	0.253 (8)
GS	0.000 (10)	0.000 (10)	0.000 (9)	0.000 (10)
LO	0.810 (7)	0.036 (8)	0.069 (7)	0.359 (6)
PH	0.982 (2)	0.930 (1)	0.955 (1)	0.300 (7)
MT	0.400 (8)	0.002 (9)	0.004 (8)	0.000 (9)
PU	0.956 (4)	0.730 (3)	0.827 (2)	0.725 (2)
RF	0.991 (1)	0.692 (4)	0.814 (3)	0.603 (4)
SVM	0.978 (3)	0.647 (5)	0.778 (4)	0.636 (3)

Observation 3) ***The TS approach statistically significantly outperformed the other ILAs in terms of the TP rate for all delete rates in the Knox project.*** We observed that the TF approach is the best approach in the Avro, Tez, ZooKeeper, and Chukwa projects in terms of the TP rate. However, in the Knox project, the TS approach achieved the highest rank in all the delete rates. Hence, the TS approach may recover the most missing defect-fixing commits in certain projects.

Observation 4) ***The PH achieved the highest rank (statistically significantly outperforming the ILAs that are placed at lower ranks) in terms of the recall, or F1 in 32 out of 40 cases between the delete rates of 10% and 40%.*** Tables 6(b), 6(d), 6(f), 6(h), and 6(j) list the results on the datasets with the delete rate of 10%. The PH achieved the highest rank on the recall and F1 in all the projects. We observed similar results between the delete rates of 10% and 40% (32 out of 40 cases[6]). This result implies that the PH detects many defect-fixing commits (recall) while keeping the number of false-positive defect-fixing commits moderate (F1). However, the PH achieved statistically significantly lower recall and F1 in the datasets with the delete rate of 50% except for one case and TP rate in all delete rates compared with the TF or TS approach. Hence, the PH potentially overlooks missing defect-fixing commits compared with the TF or TS approach.

Observation 5) ***The RF approach achieved the highest rank in terms of the precision in 22 out of 25 cases.*** Except for the Tez project with delete rates of 10%, 20%, and 50%, the RF approach achieved the highest rank on precision. This result implies that the RF approach prevents false-positive defect-fixing commits. Indeed, the median precision values are over 0.900. Hence, the RF approach could recover missing defect-fixing commits accurately. However, the recall values and ranks are low. Hence, the RF approach may overlook many defect-fixing commits. Note that the SVM approach achieved the highest or second highest rank in 19 out of 25 cases. Hence, the SVM approach could also recover missing defect-fixing commits accurately.

[6] Here a case indicates a cell in the table. The two evaluation measures (recall, and F1) for five projects with four deleted rates consist of 40 cases.

Summary of RQ1

We observed that the TF approach, TS approach, PH, RF approach, and SVM approach detected defect-fixing commits accurately. More specifically, we found the following:

– The TF approach recovered the most missing defect-fixing commits for all the delete rates except for the Knox project. The TS approach recovered the most missing defect-fixing commits for all the delete rates in the Knox project.
– The PH detects many defect-fixing commits while keeping the false-positive defect-fixing commits moderate compared with the TF approach on the datasets with the delete rates between 10% and 40% (32 out of 40 cases).
– The RF and SVM approaches achieved the highest or second highest precision in almost all cases, and therefore, these approaches can detect missing defect-fixing commits accurately.

6.2 RQ2: **Which Issue-Link Algorithm is the Best to Prevent a Defect Prediction Model From Being Affected by Missing Defect-Fixing Commits in Defect Prediction?**

Motivation and Approach: From the RQ1 results, we found that the ILAs can detect missing defect-fixing commits. In particular, the following ILAs are well performed:

– the time filtering approach;
– the natural language text similarity approach;
– the Phantom heuristics ;
– the random forest approach;
– the support vector machine approach.

We hypothesize that such ILAs can improve the reliance of defect prediction performance on a *low-quality dataset* by reducing the number of missing defect-fixing commits. A low-quality dataset indicates a dataset that has many missing defect-fixing commits. If there exist many missing defect-fixing commits, a defect prediction model may not learn sufficient numbers of defect-inducing commits that are detected by insufficient numbers of defect-fixing commits.

We prepared six possible scenarios: we randomly deleted 0% to 50% of links in steps of 10% as described in Section 4. In particular, we refer to the scenario where 0% of links are deleted as a *high-quality dataset scenario*; we refer to the other scenarios (where 10%, 20%, 30%, 40%, and 50% of links are deleted) as *low-quality dataset scenarios*.

We refer to the defect prediction performance on the high-quality dataset where the KE approach is used to detect defect-fixing commits as the *ground-truth defect prediction performance*. We computed the difference between the ground-truth defect prediction performance and the defect prediction performance on the low-quality

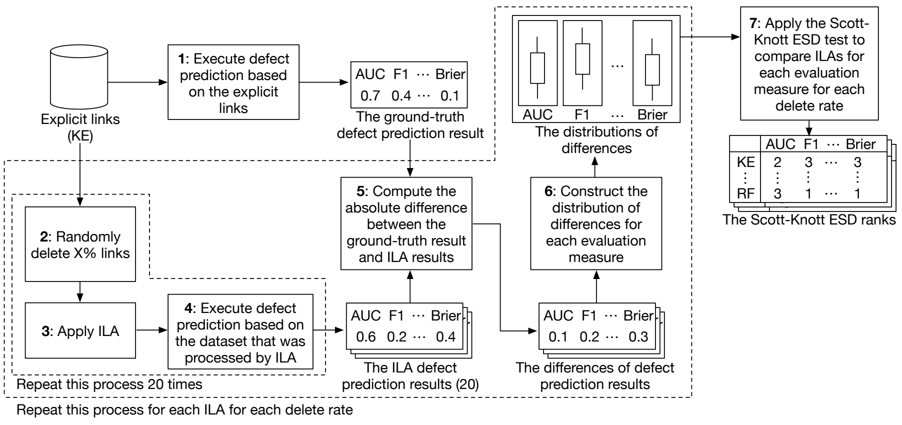
Explicit links (KE)
1: Execute defect prediction based on the explicit links
AUC F1 ··· Brier
0.7 0.4 ··· 0.1
The ground-truth defect prediction result
AUC F1 ··· Brier
The distributions of differences
7: Apply the Scott-Knott ESD test to compare ILAs for each evaluation measure for each delete rate
AUC F1 ··· Brier
KE 2 3 ··· 3
RF 3 1 ··· 1
The Scott-Knott ESD ranks
2: Randomly delete X% links
3: Apply ILA
4: Execute defect prediction based on the dataset that was processed by ILA
5: Compute the absolute difference between the ground-truth result and ILA results
6: Construct the distribution of differences for each evaluation measure
AUC F1 ··· Brier
0.6 0.2 ··· 0.4
The ILA defect prediction results (20)
AUC F1 ··· Brier
0.1 0.2 ··· 0.3
The differences of defect prediction results
Repeat this process 20 times
Repeat this process for each ILA for each delete rate

Fig. 5: Procedure of RQ2 approach for a studied project.

datasets where we use any ILAs. If such ILAs detect missing defect-fixing commits accurately and sufficiently, the difference would be smaller than using the KE approach only on the low-quality dataset scenarios. For example, if a defect prediction model on a low-quality dataset where an ILA is used matches the ground-truth defect prediction performance, such an ILA may detect all the missing defect-fixing commits. Note that we investigated not only an ILA, but also all combinations of the ILAs that are well performed in RQ1.

Figure 5 shows the procedure of the RQ2 approach for a studied project. We describe the steps in the following. The details are described in Section 5.

1. Execute defect prediction based on the explicit links. We used the explicit links (detected by the KE approach on the high-quality dataset) to detect defect-fixing commits and compute the *ground-truth defect prediction result* in terms of six evaluation measures (AUC, precision, recall, F1, MCC, and Brier score).

2. Randomly delete X% links. We randomly delete *X*% links (10% to 50% in steps of 10%) from the explicit links and prepare a low-quality dataset.

3. Apply ILA. We apply an ILA to the dataset to detect missing defect-fixing commits.

4. Execute defect prediction based on the dataset that was processed by ILA. We execute defect prediction based on the dataset. We repeat steps 2–4 20 times to relieve the data selection bias of deleted links. Eventually, we have 20 *ILA defect prediction results* for each evaluation measure.

5. Compute the absolute difference between the ground-truth result and ILA results. We compute the absolute difference between the ground-truth defect prediction result and the ILA defect prediction results. As we have 20 ILA defect prediction results, this process results in 20 differences for each evaluation measure.

6. Construct the distribution of differences for each evaluation measure. As each of the evaluation measures has 20 results, we consider these 20 results as a distribution

Table 7: The median absolute differences between the ground-truth result and the ILA results in the Avro project with the delete rate of 50%.

ILA	AUC	F1	Pre	Rec	MCC	Brier	COUNT
KE	0.028 (3)	0.027 (3)	0.018 (5)	0.061 (3)	0.039 (3)	0.018 (4)	0 (0)
TF	0.018 (2)	0.011 (1)	0.007 (1)	0.030 (1)	0.016 (1)	0.008 (1)	6 (5)
TS	0.031 (2)	0.014 (1)	0.008 (3)	0.030 (2)	0.019 (2)	0.008 (2)	6 (1)
PH	0.021 (2)	0.019 (1)	0.011 (3)	0.030 (2)	0.027 (2)	0.016 (4)	5 (1)
RF	0.044 (5)	0.035 (3)	0.021 (6)	0.061 (3)	0.047 (3)	0.014 (4)	0 (0)
SVM	0.026 (4)	0.025 (3)	0.016 (5)	0.061 (3)	0.032 (4)	0.014 (3)	1 (0)
TF,TS	0.021 (2)	0.018 (1)	0.012 (3)	0.030 (2)	0.026 (2)	0.010 (2)	6 (1)
TF,PH	0.016 (1)	0.015 (1)	0.010 (3)	0.030 (2)	0.022 (2)	0.009 (1)	6 (3)
TF,RF	0.020 (2)	0.018 (1)	0.014 (3)	0.030 (2)	0.026 (2)	0.011 (2)	6 (1)
TF,SVM	0.017 (2)	0.020 (1)	0.011 (3)	0.030 (2)	0.028 (2)	0.011 (2)	6 (1)
TS,PH	0.022 (2)	0.017 (1)	0.010 (2)	0.061 (2)	0.024 (2)	0.009 (2)	6 (1)
TS,RF	0.030 (3)	0.021 (1)	0.014 (3)	0.030 (3)	0.030 (2)	0.010 (2)	4 (1)
TS,SVM	0.019 (2)	0.016 (1)	0.010 (2)	0.061 (3)	0.023 (2)	0.009 (2)	5 (1)
PH,RF	0.029 (3)	0.018 (1)	0.011 (3)	0.061 (3)	0.023 (2)	0.017 (4)	3 (1)
PH,SVM	0.023 (3)	0.017 (1)	0.010 (3)	0.061 (3)	0.023 (2)	0.017 (4)	3 (1)
RF,SVM	0.033 (4)	0.015 (1)	0.012 (3)	0.061 (2)	0.022 (2)	0.021 (4)	4 (1)
TF,TS,PH	0.014 (1)	0.016 (1)	0.010 (2)	0.030 (2)	0.023 (2)	0.010 (2)	6 (2)
TF,TS,RF	0.025 (3)	0.021 (2)	0.014 (4)	0.030 (3)	0.031 (3)	0.008 (2)	3 (0)
TF,TS,SVM	0.021 (2)	0.017 (1)	0.011 (3)	0.030 (1)	0.025 (2)	0.012 (2)	6 (2)
TF,PH,RF	0.016 (1)	0.016 (1)	0.010 (2)	0.030 (1)	0.022 (2)	0.008 (2)	6 (3)
TF,PH,SVM	0.034 (3)	0.017 (1)	0.011 (3)	0.045 (2)	0.025 (2)	0.007 (2)	5 (1)
TF,RF,SVM	0.024 (3)	0.017 (1)	0.010 (3)	0.030 (2)	0.024 (2)	0.011 (2)	5 (1)
TS,PH,RF	0.017 (2)	0.017 (1)	0.010 (2)	0.030 (2)	0.025 (2)	0.007 (1)	6 (2)
TS,PH,SVM	0.023 (2)	0.015 (1)	0.010 (2)	0.030 (2)	0.023 (1)	0.008 (2)	6 (2)
TS,RF,SVM	0.024 (2)	0.019 (1)	0.012 (2)	0.045 (2)	0.027 (2)	0.009 (2)	6 (1)
PH,RF,SVM	0.027 (3)	0.010 (1)	0.007 (2)	0.045 (3)	0.015 (2)	0.018 (4)	3 (1)
TF,TS,PH,RF	0.024 (2)	0.015 (1)	0.010 (2)	0.030 (2)	0.023 (2)	0.009 (2)	6 (1)
TF,TS,PH,SVM	0.022 (2)	0.017 (1)	0.010 (3)	0.045 (2)	0.024 (2)	0.008 (2)	6 (1)
TF,TS,RF,SVM	0.025 (3)	0.017 (1)	0.012 (2)	0.030 (2)	0.024 (2)	0.006 (1)	5 (2)
TF,PH,RF,SVM	0.017 (2)	0.016 (1)	0.009 (3)	0.030 (2)	0.024 (2)	0.009 (2)	6 (1)
TS,PH,RF,SVM	0.023 (2)	0.016 (1)	0.010 (3)	0.030 (2)	0.023 (2)	0.005 (1)	6 (2)
TF,TS,PH,RF,SVM	0.024 (2)	0.020 (1)	0.013 (2)	0.030 (2)	0.028 (2)	0.009 (2)	6 (1)

of an evaluation measure. We repeat this process for each ILA for each delete rate (10% to 50%). Eventually, each ILA has a distribution for each evaluation measure for each delete rate.

7. Apply the Scott-Knott ESD test to compare ILAs for each evaluation measure for each delete rate. To identify the ILA that achieves the smallest differences, we apply the Scott-Knott ESD test to the distributions of all ILAs for each evaluation measure for each delete rate.

In these steps, the execution time of defect prediction (Strep 4) is remarkably long. In this paper, we repeated this step 20 times for 31 ILAs (all combinations of six ILAs), 5 studied projects, and 6 dataset scenarios. The execution time of one repetition for an ILA, a studied project, and a dataset scenario is about 761 seconds on a computational resource, which consists of 8 CPUs and 32 GB memory with parallel execution. Hence, if we repeated this process 100 times similar to RQ1, the total expected execution time would be $31 * 5 * 6 * 100 * 761 \simeq 819$ days. To reduce this execution time, we only repeat this process 20 times in this RQ.

Table 8: The sum of the cases where the rank is higher than the rank of the KE approach or the highest rank (the sum of the COUNT values).

ILA	The sum of the COUNT values
TS,PH,RF,SVM	111
TS,PH,RF	110
TF,RF,SVM	106
TS,RF	104
TF,TS,SVM	103
TF,TS,PH,SVM	101
TS,PH	100
TF,TS,PH,RF	98
TF,TS,PH	97
TF,TS,RF,SVM	96
TF,TS,PH,RF,SVM	95
TS	94
TF,PH,RF	94
TF,TS	93
TS,PH,SVM	93
TF,PH	91
TF,SVM	91
TF,PH,RF,SVM	91
TS,RF,SVM	90
TF,TS,RF	88
TF,PH,SVM	88
PH,RF,SVM	88
TF	84
TS,SVM	84
TF,RF	83
PH,SVM	83
PH,RF	81
RF,SVM	80
PH	73
RF	69
SVM	66
KE	56

Results: **Observation 6)** ***The combination of the TS, PH, RF, and SVM approaches achieved the highest rank or statistically significantly reduces the absolute differences the most compared with the KE approach.*** To understand this observation easily, we first describe the result in an experimental setting. Table 7 lists the median absolute differences between the ground-truth result and the ILA results in the Avro project with the delete rate of 50%. The values in parentheses show the ranks that were computed by the Scott-Knott ESD test across the ILAs. The gold cells indicate the cases where the rank is the highest (rank 1) across the ILAs. The cyan cells indicate the cases where the rank is higher than the rank of the KE approach. The COUNT column indicates the numbers of gold and cyan cells for each row; the values in parentheses indicate the number of gold cells only. We observed that 18 of the ILAs statistically significantly reduce the absolute differences for all the evaluation measures (i.e., the values in the COUNT column were six). Hence, these ILAs work well at reducing the absolute differences across the ILAs in this experimental setting.

Table 9: All the median absolute differences between the ground-truth result and the results of the combination of TS, PH, RF, and SVM approaches.

Project	Delete rate	AUC	F1	Pre	Rec	MCC	Brier	COUNT
Avro	10	0.016 (1)	0.014 (1)	0.008 (1)	0.030 (1)	0.020 (1)	0.010 (2)	6 (5)
	20	0.015 (2)	0.013 (1)	0.008 (1)	0.045 (1)	0.018 (1)	0.008 (2)	5 (4)
	30	0.014 (1)	0.012 (1)	0.006 (1)	0.030 (1)	0.018 (1)	0.008 (1)	6 (6)
	40	0.027 (2)	0.011 (2)	0.009 (2)	0.045 (1)	0.016 (2)	0.012 (3)	5 (1)
	50	0.023 (2)	0.016 (1)	0.010 (3)	0.030 (2)	0.023 (2)	0.005 (1)	6 (2)
Tez	10	0.009 (1)	0.016 (1)	0.013 (1)	0.047 (2)	0.016 (1)	0.003 (1)	5 (5)
	20	0.006 (1)	0.024 (2)	0.019 (4)	0.041 (3)	0.026 (2)	0.002 (2)	1 (1)
	30	0.010 (2)	0.017 (1)	0.020 (3)	0.029 (1)	0.020 (1)	0.004 (2)	5 (3)
	40	0.008 (2)	0.024 (1)	0.026 (2)	0.041 (2)	0.024 (1)	0.005 (3)	4 (2)
	50	0.010 (3)	0.029 (2)	0.022 (2)	0.053 (5)	0.032 (1)	0.005 (3)	2 (1)
ZooKeeper	10	0.008 (4)	0.009 (3)	0.010 (3)	0.020 (1)	0.013 (2)	0.007 (4)	2 (1)
	20	0.006 (1)	0.017 (2)	0.013 (1)	0.020 (2)	0.020 (3)	0.005 (1)	4 (3)
	30	0.006 (1)	0.010 (1)	0.012 (3)	0.017 (1)	0.011 (1)	0.008 (4)	6 (4)
	40	0.006 (1)	0.010 (1)	0.012 (2)	0.027 (1)	0.012 (1)	0.006 (2)	6 (4)
	50	0.008 (3)	0.013 (2)	0.015 (3)	0.027 (2)	0.017 (2)	0.009 (3)	6 (0)
Chukwa	10	0.014 (3)	0.026 (3)	0.021 (4)	0.026 (2)	0.036 (2)	0.009 (3)	0 (0)
	20	0.006 (1)	0.027 (2)	0.024 (2)	0.026 (2)	0.039 (2)	0.009 (3)	4 (1)
	30	0.014 (1)	0.026 (2)	0.028 (3)	0.026 (2)	0.039 (3)	0.010 (2)	3 (1)
	40	0.013 (3)	0.020 (1)	0.017 (3)	0.051 (3)	0.030 (1)	0.006 (3)	5 (2)
	50	0.017 (4)	0.028 (2)	0.028 (3)	0.026 (1)	0.043 (1)	0.007 (1)	6 (3)
Knox	10	0.007 (3)	0.013 (2)	0.007 (1)	0.041 (2)	0.020 (1)	0.007 (3)	4 (2)
	20	0.006 (1)	0.006 (1)	0.009 (2)	0.033 (1)	0.008 (1)	0.008 (3)	5 (4)
	30	0.007 (1)	0.009 (1)	0.007 (1)	0.041 (3)	0.015 (1)	0.004 (4)	5 (4)
	40	0.007 (2)	0.008 (1)	0.007 (1)	0.026 (1)	0.012 (1)	0.006 (2)	5 (4)
	50	0.005 (1)	0.016 (2)	0.007 (1)	0.041 (4)	0.026 (4)	0.005 (1)	5 (3)

Table 8 lists the summation of all the COUNT values for each ILA. As we used six evaluation measures in the five projects with five delete rates, the maximum summation value is 150. Indeed, we observed that the combination of TS, PH, RF, and SVM approaches achieved 111, which is the highest value. This result implies that this combination statistically significantly reduces the absolute differences compared with the KE approach or at least achieved the highest rank.

Observation 7) ***All ILAs statistically significantly reduced the absolute differences compared with the KE approach.*** Table 8 indicates that the KE approach, which is the baseline, achieved 56, which is the smallest value. Hence, the ILAs statistically significantly reduced the absolute differences compared with the KE approach.

Observation 8) ***The combination of the TS, PH, RF, and SVM approaches achieve better results in the lower-quality dataset scenarios while it may achieve worse results in the higher-quality dataset scenarios.*** Table 9 lists all the median absolute differences of the combination of TS, PH, RF, and SVM approaches with the Scott-Knott ESD test results. In the Chukwa project, the numbers of cyan and gold cells with the delete rate of 10% were zero. In addition, the numbers in the Tez project with the delete rate of 20% were one and one; the numbers in the ZooKeeper project with the delete rate of 10% were two and one. This result implies that the best combination of ILAs may be more suitable for the lower-quality dataset while it may achieve worse results in certain projects with higher-quality datasets.

Summary of RQ2

We observed that the combination of the TS, PH, RF, and SVM approaches is the best at reducing the absolute differences compared with the ground-truth defect prediction performance. More specifically,

– The combination achieved the smallest or smaller absolute differences from the ground-truth defect prediction performance the most frequently.
– The combination may be more suitable for the lower-quality dataset scenario.
– All the ILAs statistically significantly reduce the absolute differences compared with the KE approach.

7 Discussion

7.1 Can the RF Approach Detect Missing Defect-Fixing Commits in the High-Quality Dataset?

From the RQ1 result, the RF approach achieves the highest precision (e.g., 1.000); hence, we suppose that the RF approach can identify new defect-fixing commits that are not detected by the KE approach in the high-quality dataset scenario. Table 10 lists the links of newly identified defect-fixing commits and issue ids in the high-quality dataset. The non-green cells are actual links that were confirmed manually by two of the authors.

Observation 9) ***The RF approach identified several missing defect-fixing commits.*** For example, in the Avro project, the RF approach identified the missing defect-fixing commit `7eaba5aa` that links with `AVRO-555`. The commit message includes `AVR0-555`, which uses `0` instead of `O`. It is easy for humans to detect this missing defect-fixing commit; however, it is difficult for tools to interpret `AVR0` as `AVRO`. In the Tez project, the RF approach identified the missing defect-fixing commit `a0d63ed05` that links with `TEZ-3001`. This commit message includes an issue id of `TEZ-2496` that is not labeled `Bug`. However, a comment of `TEZ-3001` describes that the patch of `TEZ-2496` will fix the issue of `TEZ-3001`. The RF approach can identify such a complex link as well.

7.2 Why Does the TF Approach Generally Work Well in Detecting Defect-Fixing Commits?

Observation 10) ***Developing software products is conducted in a short time interval over multiple occurrences.*** Figure 6 shows the number of appearances of each issue report label for each month between the initial commit month to the end of 2013 in the Avro project. We classified issue report labels into each month based on the dates of their linked commits. We show their labels (e.g., `Bug`, `Improvement`, and `New Feature`) as line plots. Note that we allow the same issue report label to be

Table 10: The links of newly identified missing defect-fixing commits and issue ids by the RF approach in the high-quality dataset. The non-green cells are actual links that were confirmed manually by two of the authors.

Answer	Issue ID	Commit Hash
Avro	AVRO-4	9b14a2a7
	AVRO-14	e6d1fca4
	AVRO-262	1acc9913
	AVRO-401	79c09800
	AVRO-555	7eaba5aa
	AVRO-656	50768496
	AVRO-718	e623053d
	AVRO-746	34d6f3ac
	AVRO-900	d7dbac1a
	AVRO-1077	5e8664c1
	AVRO-1123	196011b4
	AVRO-1131	eaa43dbc
	AVRO-1140	ab5eb854
	AVRO-1251	267bda89
	AVRO-1320	258f800d
	AVRO-1540	0478e9ce
Tez	TEZ-243	a167e861e
	TEZ-254	778ad2438
	TEZ-258	c35702d6f
	TEZ-739	ea345bab5
	TEZ-740	9138f7906
	TEZ-846	816e2e5a9
	TEZ-924	aca83090e
	TEZ-2479	4e57a922b
	TEZ-2479	c1d334b4d
	TEZ-2924	40e864d14
	TEZ-3001	a0d63ed05
	TEZ-3423	cbd4eacb0
	TEZ-1963	4bc64b5c3
	TEZ-2046	a6bfc1ad3
	TEZ-2046	5b2f011f1
ZooKeeper	ZOOKEEPER-55	1d2a7863
	ZOOKEEPER-76	f1f13a37
	ZOOKEEPER-138	5d56e6d2
	ZOOKEEPER-181	03f0f816
	ZOOKEEPER-554	e8d31e67
	ZOOKEEPER-1943	86ebdc9a
Chukwa	CHUKWA-117	47f2f79
	CHUKWA-411	9b5ee68
Knox	KNOX-71	ad693b0e7

counted repeatedly if such an issue report links to two or more commits. The vertical dotted lines indicate the month in which a new version was released as described on the Avro release page [2].

We observed that each label has spikes around release dates. This result implies that developers commit their addition/modification/deletion related to issue reports in a short time interval. In addition, we observed many commits that correspond to issue reports labeled `Bug` are gathered around a short time interval through our manual analysis on the Git repository. We observed the same tendency on the Tez

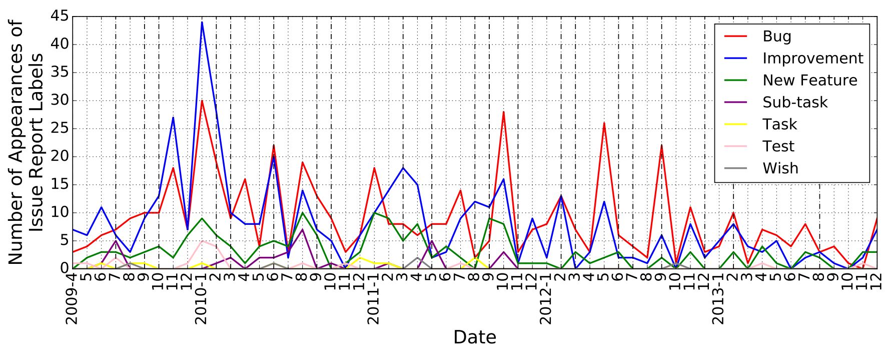

Fig. 6: The number of appearances of each issue report label for each month between the initial commit month to the end of 2013 in the Avro project.

and ZooKeeper projects[7]. This may be a reason why the TF approach generally works well.

7.3 Which Time Interval is the Best to Detect Defect-Fixing Commits?

Observation 11) ***The 10-minute time interval is the best setting to detect defect-fixing commits in our studied projects.*** Table 11 indicates the performance of the TF approach in terms of detecting defect-fixing commits in different time intervals. The gold cells indicate over 0.7. The values in the parentheses show the ranks that were computed by the Scott-Knott ESD test for each evaluation measure across five time intervals. The delete rate is 50%. We observed two findings: the smaller the time interval, the better the performance in terms of precision; the larger the time interval, the better the performance in terms of recall and the TP rate (deleted). This is because these evaluation measures are the trade-off. Hence, we focused on the harmonic evaluation measure, F1. As the TF approach does not work well in the Knox project (RQ1), we only studied the other four projects.

The time intervals of 10 and 30 minutes achieved rank 1 once in the Scott-Knott ESD test. The time interval of 5 minutes achieved rank 1 twice. However, the time interval of 5 minutes achieved rank 4 in the Tez project. Although the time interval of 10 minutes achieved rank 1 once, it achieved rank 2 in the other projects. Therefore, we concluded that the time interval of 10 minutes is well balanced. This result implies that the 10-minute time interval detects many defect-fixing commits while keeping the number of false-positive defect-fixing commits low.

[7] The Chukwa project does not show the dates of all release dates. The TF approach does not work well in the Knox project. Hence, we ignored these projects in this analysis.

Table 11: The performance of the time filtering approach in terms of detecting defect-fixing commits in five different time intervals with the delete rate of 50%.

(a) The Avro project

Time Interval (Min.)	Precision	Recall	F1	TP rate (deleted)
5	0.839 (1)	0.774 (5)	0.805 (1)	0.777 (5)
10	0.757 (2)	0.821 (4)	0.788 (2)	0.824 (4)
30	0.626 (3)	0.860 (3)	0.724 (3)	0.860 (3)
60	0.528 (4)	0.877 (2)	0.660 (4)	0.876 (2)
120	0.465 (5)	0.897 (1)	0.612 (5)	0.895 (1)

(b) The Tez project

Time Interval (Min.)	Precision	Recall	F1	TP rate (deleted)
5	0.894 (1)	0.660 (5)	0.760 (4)	0.664 (5)
10	0.847 (2)	0.776 (4)	0.810 (2)	0.777 (4)
30	0.768 (3)	0.858 (3)	0.811 (1)	0.858 (3)
60	0.706 (4)	0.884 (2)	0.785 (3)	0.884 (2)
120	0.622 (5)	0.909 (1)	0.738 (5)	0.909 (1)

(c) The ZooKeeper project

Time Interval (Min.)	Precision	Recall	F1	TP rate (deleted)
5	0.955 (1)	0.700 (5)	0.808 (2)	0.701 (5)
10	0.908 (2)	0.771 (4)	0.834 (1)	0.774 (4)
30	0.784 (3)	0.816 (3)	0.800 (3)	0.816 (3)
60	0.716 (4)	0.843 (2)	0.774 (4)	0.844 (2)
120	0.653 (5)	0.856 (1)	0.741 (5)	0.856 (1)

(d) The Chukwa project

Time Interval (Min.)	Precision	Recall	F1	TP rate (deleted)
5	0.828 (1)	0.652 (5)	0.729 (1)	0.651 (5)
10	0.730 (2)	0.706 (4)	0.718 (2)	0.704 (4)
30	0.605 (3)	0.760 (3)	0.674 (3)	0.760 (3)
60	0.560 (4)	0.783 (2)	0.653 (4)	0.783 (2)
120	0.515 (5)	0.810 (1)	0.630 (5)	0.810 (1)

(e) The Knox project

Time Interval (Min.)	Precision	Recall	F1	TP rate (deleted)
5	0.893 (1)	0.392 (5)	0.545 (5)	0.394 (5)
10	0.867 (2)	0.451 (4)	0.594 (4)	0.452 (4)
30	0.808 (3)	0.575 (3)	0.672 (3)	0.572 (3)
60	0.759 (4)	0.637 (2)	0.692 (1)	0.636 (2)
120	0.683 (5)	0.679 (1)	0.681 (2)	0.677 (1)

7.4 Do ILAs Affect the Effort-Aware Defect Prediction Performance Measures?

Motivation and Approach: Just-in-time defect prediction models help in identifying whether a commit is likely to be defective. If such a commit is identified as defective, developers use their *test effort* to inspect this commit to modify the defect; however, their test effort is limited. Hence, considering the test effort is also important to evaluate defect prediction models.

To evaluate this perspective, we can use *effort-aware measures* [39, 58]. Generally, developers need more test effort to inspect more commits. In addition, large-size commits (e.g., including many added/deleted lines) need more test effort to be inspected than small-size commits. Hence, effort-aware measures use the number of commits and lines that are inspected by developers to evaluate defect prediction models. In this discussion, we investigate whether ILAs affect this perspective in just-in-time defect prediction.

We computed effort-aware measures for the results of RQ2. Similar to RQ2, we investigated the difference between the ground truth result and the ILA results. We use four effort-aware evaluation measures that were applied by the prior study [58]: IFA, PII@L, CostEffort@L, and Norm(P_{opt}).

IFA measures the number of commits that need to be inspected before the first defect-inducing commit is identified. The smaller IFA implies that defect prediction models identify defect-inducing commits at an early time. PII@L and CostEffort@L measure the number of commits that need to be inspected and the number of identified defect-inducing commits, respectively, when developers can inspect L lines of code. We used the same L as the prior study [58]: 20%, 1000, and 2000. Norm(P_{opt}) indicates the similarity between the prediction result and the optimized case where defect prediction models perfectly predict defect-inducing commits according to the number of lines of code [39]. The range is between 0 to 1; the higher the value, the better the prediction result is implied.

Results: **Observation 12)** ***The KE approach is the best approach in terms of the effort-aware evaluation measures.*** In RQ2, the KE approach is the worst approach. However, no ILAs achieved a higher sum of the COUNT values than the KE approach in terms of the effort-aware evaluation measures (Table 12). As a result, all ILAs may not work well in terms of the effort-aware evaluation measures. However, the difference between the largest sum of the COUNT values and the smallest one is 32 (166-134) in this analysis while that in RQ2 is 55 (111-56). Also, because we used eight effort-aware evaluation measures (PII@L and CostEffort@L have three variants), the maximum value is 200 while that in RQ2 is 150. If we consider this difference, the difference ratio between the largest value and the smallest value in RQ2 is two times larger than this analysis. Hence, the difference between the best approach and the worst approach in terms of the effort-aware evaluation measures may be small. Future studies are necessary to investigate the relationship between ILAs and effort-aware evaluation measures.

7.5 The False-positive/negative Defect-fixing Commits in the Ground Truth Data

As we described in Section 5.1, we found that our ground truth data (defect-fixing commits) are accurate through manual inspection. However, there exist a few false-positive/negative defect-fixing commits.

For example, the commit `cf3318e1b` in the Tez project is labeled a defect-fixing commit. However, this is a false-positive defect-fixing commit. The commit message includes two issue ids: `TEZ-1594` labeled `Sub-task` and `TEZ-8` labeled `Bug`. As the KE approach links this commit to these two issue reports, this commit is referred to

Table 12: The sum of the cases where the rank is higher than the rank of the KE approach or the highest rank (the sum of the COUNT values).

ILA	The sum of the COUNT values
KE	166
TF,TS,SVM	157
TF,PH,SVM	156
TF,TS,PH	155
SVM	154
TS,PH	154
TF,PH,RF,SVM	153
TF,TS	152
TF,PH	152
TF,TS,PH,SVM	152
TF,TS,RF,SVM	152
RF,SVM	149
TF,PH,RF	149
TS,PH,SVM	149
TF,RF	148
TS,PH,RF,SVM	148
RF	147
TF,SVM	145
TS,RF	145
PH,RF	145
TF,RF,SVM	144
PH,RF,SVM	144
TS,SVM	143
TF,TS,PH,RF,SVM	143
TF	142
PH	142
TF,TS,RF	142
TF,TS,PH,RF	142
TS,RF,SVM	141
TS,PH,RF	140
PH,SVM	138
TS	134

as a defect-fixing commit; however, the link to `TEZ-8` is a false-positive link because `TEZ-8` is not directly related to this commit. Hence, this commit is a false-positive defect-fixing commit.

The commit `0b74bd5e` in the Avro project is an example of a false-negative defect-fixing commit. This commit does not include any issue ids in its commit message. However, `CHANGES.txt`, which is a changed file, includes an issue id labeled `Bug`. Hence, this commit should be a defect-fixing commit.

Finally, this manual inspection provides us with an interesting suggestion. Our basic restriction (Section 5.2) may exclude defect-fixing commits that do not fix source code (i.e., *noise*). For example, the commit message of the commit `c89e352e0` in the Tez project includes an issue id, `TEZ-2885` labeled `Bug`. Hence, prior KE approaches may link this commit and the issue report. However, this commit only modifies `CHANGES.txt` while the actual defect-fixing commit is the commit `6eb2cb551`. This may occur if developers forget to modify `CHANGES.txt`. This kind of defect-fixing commit should be excluded from the defect prediction research. Because our basic restriction excludes these commits, we suggest researchers and practitioners use the basic restriction at least. This kind of restriction is employed by prior work [50].

7.6 Answer to This Paper: Which ILAs Should We Use?

In summary, researchers and practitioners need to select the ILAs according to their particular purpose. If researchers and practitioners want to evaluate the defect prediction models in the low-quality dataset scenario, we recommend using the best ILA in terms of the absolute differences with the ground-truth defect prediction performance: the combination of the TS, PH, RF, and SVM approaches. This is because, in the low-quality dataset scenarios, this ILA can reduce the absolute differences of defect prediction performance from that in the high-quality dataset scenario with the KE approach (RQ2).

If researchers and practitioners investigate the defect-fixing commits, we recommend using the ILA that achieves the highest precision: the RF approach because researchers and practitioners do not need to worry about false-positive defect-fixing commits (RQ1).

If researchers and practitioners want to identify almost all missing defect-fixing commits, we recommend using the TF approach or TS approach because these ILAs achieve the highest TP rate in different projects (RQ1) while being a very simple approach. However, the precision value is lower than those of the other accurate ILAs. Researchers and practitioners need to consider false-positive defect-fixing commits.

Finally, in defect prediction, we recommend using the basic restriction to exclude noise of defect-fixing commits. In particular, considering the dates of the commit and the issue report is a simple but effective approach to detect defect-fixing commits.

Summary

After applying the KE approach, we recommend using the ILAs as the following criteria to recover missing defect-fixing commits:

- If the aim is to evaluate the defect prediction models in the low-quality dataset scenario, we recommend using the combination of the TS, PH, RF, and SVM approaches.
- If the aim is to investigate the defect-fixing commits accurately, we recommend using the RF approach.
- If the aim is to identify almost all missing defect-fixing commits, we recommend using the TF approach or the TS approach.
- Even if no aims, we recommend using the basic restriction at least.

8 Threats to Validity

8.1 Construct Validity

The reliability of the issue reports in the studied issue-tracking system (i.e., JIRA) is a threat in this study. Ramler et al. [61] described this challenge. Herzig et al. [34] reported that 39% of files that are labeled as defective are not defective on average. Bachmann et al. [12] reported that some defects are only reported on the mailing list. In addition, such defects are very important because the core developers in the Apache projects use the mailing list. Future studies are necessary to investigate the quality of issue-tracking system to improve the reliability of our findings.

Defect-fixing commits could include addition/modification/deletion that is not related to defect fixing. For example, Mills et al. [51] reported that around 63.1% (848/1,344) of modified files in defect-fixing commits are not related to defect fixing. If we removed such files from the defect-fixing commits, this might result in different defect prediction performance for each ILA. Future studies are necessary to investigate whether our results are consistent with removing such files.

We used the results of the keyword extraction approach as our ground-truth data. The keyword extraction approach uses a regular expression to extract issue ids from commit messages. Unfortunately, this process may induce false-positive/negative defect-fixing commits. Such commits would affect our experimental results, though we manually inspected the accuracy of ground truth data and found the accuracy is high.

To execute the natural language text similarity approach, we removed the issue ids from commit messages on the missing defect-fixing commits to make our experimental setting closer to a practical situation. However, commit messages also frequently include issue report titles because of their commit rule [5]. This JIRA title may make our experimental setting artificial and unfair. We kept the JIRA title because we assume that developers forget to add issue ids only.

In this paper, we mainly focus on the ILAs while defect prediction includes several factors such as the process of detecting defect-inducing commits. Hence, the results of our study are restricted by our experimental setting in defect prediction. Future studies are necessary to investigate the relationship between ILAs and the other factors in defect prediction.

8.2 External Validity

To generalize our results, we applied our experiments to five open-source software projects on the Apache Software Foundation. These studied projects contain two domains. However, all the projects are Apache projects and have high-quality commit messages. Future studies are necessary to investigate whether our results generalize to other projects.

We carefully chose our studied ILAs from prior studies that were collected by our systematic literature study with the snowballing approach. However, we decided not to use a few ILAs and we may have overlooked a few prior studies that proposed ILAs. Future studies are necessary to investigate such ILAs with software projects in which we can use all necessary information.

8.3 Internal Validity

We summarized our validation technique and ILAs as Python packages [45, 46]. In addition, we made the replication package [44]. Researchers and practitioners may easily repeat our experiments.

We manually investigated the correctness of the identified links in Section 7.1. This investigation was conducted by two of the authors, and we double-checked the result. However, the result may include mistakes. In addition, we manually inspected the accuracy of the ground truth data. This manual inspection has been carefully done. However, it may include mistakes.

To remove the merge commits, we used the `--no-merges` option of the git log command [31]. Hence, the accuracy to identify the merge commits depends on this option.

9 Conclusion

The impact of false-positive/negative defect-inducing commits on the defect prediction performance is important when evaluating defect prediction models. To reduce the number of false-positive/negative defect-inducing commits, many prior studies have proposed ILAs to detect defect-fixing commits accurately [10, 12, 17, 18, 26, 49, 55, 63, 69, 71–74, 82, 84, 85]. However, challenges still exist, such as dataset inconsistency and small comparisons. Our work is the first large-scale study to evaluate the ILAs on the same experimental settings. In addition, we summarized the prior ILAs as our related work through our systematic literature study.

In the following, we summarize the main recommendations. We recommend selecting ILAs according to the particular purpose.

Recommendation 1: For researchers and practitioners who need to evaluate defect prediction models in the low-quality dataset scenario, we recommend using the combination of the natural language text similarity, Phantom heuristics, random forest, and support vector machine approaches. Our experiments in RQ2 show that the studied ILAs prevented the defect prediction model from being affected by missing defect-fixing commits. In particular, the combination of the natural language text similarity, Phantom heuristics, random forest, and support vector machine approaches achieved the most similar defect prediction performance with the ground-truth defect prediction performance across all the studied ILAs. If researchers and practitioners use this combination, they would not need to worry about the impact of missing defect-fixing commits to defect prediction performance.

Recommendation 2: For researchers and practitioners who need defect-fixing commits without false-positive defect-fixing commits, we recommend using the random forest approach. Our experiments in RQ1 have shown that the random forest approach achieved the highest precision. Hence, using this ILA rarely induces false-positive defect-fixing commits while reducing missing defect-fixing commits.

Recommendation 3: For researchers and practitioners who need defect datasets without missing defect-fixing commits, we recommend using the time filtering approach or the natural language text similarity approach. Our experiments in

RQ1 have shown that the time filtering approach and the natural language text similarity approach performed the highest TP rate compared with the other accurate ILAs in different projects. Hence, using the time filtering approach or the natural language text similarity approach reduces the number of missing defect-fixing commits. In addition, these ILAs are very simple. Note that when using these approaches, researchers and practitioners need to consider false-positive defect-fixing commits.

Recommendation 4: Considering the dates of the commit and the issue report help exclude noise of defect-fixing commits. Our manual inspection discussed in Section 7.5 shows that our basic restriction can exclude defect-fixing commits that do not fix source code. Hence, using our basic restriction that uses the dates excludes such noise of defect-fixing commits for defect prediction.

Acknowledgment

This work has been supported by JSPS KAKENHI Japan (Grant Numbers: JP19J23477 and JP18H03222) and JSPS International Joint Research Program with SNSF (Project "SENSOR"). We would like to thank Editage (`www.editage.com`) for English language editing.

References

1. Agrawal, A., Menzies, T.: Is "better data" better than "better data miners"? In: Proceedings of the 40th International Conference on Software Engineering (ICSE), pp. 1050–1061. IEEE (2018)
2. Apache Software Foundation: Apache Avro^{TM} Releases. URL `https://avro.apache.org/releases.html`
3. Apache Software Foundation: Avro. URL `https://avro.apache.org/`
4. Apache Software Foundation: Chukwa. URL `http://chukwa.apache.org/`
5. Apache Software Foundation: HowToContribute. URL `https://cwiki.apache.org/confluence/display/ZOOKEEPER/HowToContribute`
6. Apache Software Foundation: Knox. URL `https://knox.apache.org/`
7. Apache Software Foundation: Tez. URL `https://tez.apache.org/`
8. Apache Software Foundation: ZooKeeper. URL `https://zookeeper.apache.org/`
9. Ayari, K., Meshkinfam, P., Antoniol, G., Di Penta, M.: Threats on building models from cvs and bugzilla repositories: The mozilla case study. In: Proceedings of the 2007 Conference of the Center for Advanced Studies on Collaborative Research, p. 215–228. IBM Corp. (2007)
10. Bachmann, A., Bernstein, A.: Data retrieval, processing and linking for software process data analysis. Tech. Rep. IFI-2009.0003b (2009)
11. Bachmann, A., Bernstein, A.: Software process data quality and characteristics: a historical view on open and closed source projects. In: Proceedings of the Joint International and Annual ERCIM Workshops on Principles of Software Evolution (IWPSE) and Software Evolution (Evol) Workshops (IWPSE-Evol), pp. 119–128. ACM (2009)

12. Bachmann, A., Bird, C., Rahman, F., Devanbu, P., Bernstein, A.: The missing links: bugs and bug-fix commits. In: Proceedings of the 18th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE), pp. 97–106. ACM (2010)
13. Basili, V.R., Briand, L.C., Melo, W.L.: A validation of object-oriented design metrics as quality indicators. IEEE Transactions on software engineering **22**(10), 751–761 (1996)
14. Bennin, K.E., Keung, J., Phannachitta, P., Monden, A., Mensah, S.: Mahakil: Diversity based oversampling approach to alleviate the class imbalance issue in software defect prediction. IEEE Transactions on Software Engineering **44**(6), 534–550 (2017)
15. Bergstra, J., Bengio, Y.: Random search for hyper-parameter optimization. Journal of machine learning research **13**(10), 281–305 (2012)
16. Bird, C., Bachmann, A., Aune, E., Duffy, J., Bernstein, A., Filkov, V., Devanbu, P.: 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 (ESEC/FSE), pp. 121–130. ACM (2009)
17. Bird, C., Bachmann, A., Rahman, F., Bernstein, A.: Linkster: enabling efficient manual inspection and annotation of mined data. In: Proceedings of the 19th ACM SIGSOFT International Symposium on Foundations of Software Engineering (FSE), pp. 369–370. ACM (2010)
18. Bissyandé, T.F., Thung, F., Wang, S., Lo, D., Jiang, L., Réveillère, L.: Empirical evaluation of bug linking. In: Proceedings of the 17th European Conference on Software Maintenance and Reengineering, pp. 89–98. IEEE (2013)
19. Boughorbel, S., Jarray, F., El-Anbari, M.: Optimal classifier for imbalanced data using matthews correlation coefficient metric. PloS one **12**(6), e0177,678 (2017)
20. Cohen, J.: Statistical power analysis for the behavioral sciences. Academic press (2013)
21. Čubranić, D., Murphy, G.C.: Hipikat: Recommending pertinent software development artifacts. In: Proceedings of the 25th International Conference on Software Engineering (ICSE), pp. 408–418. IEEE (2003)
22. Da Costa, D.A., McIntosh, S., Shang, W., Kulesza, U., Coelho, R., Hassan, A.E.: A framework for evaluating the results of the szz approach for identifying bug-introducing changes. IEEE Transactions on Software Engineering **43**(7), 641–657 (2016)
23. Elkan, C., Noto, K.: Learning classifiers from only positive and unlabeled data. In: Proceedings of the 14th International Conference on Knowledge Discovery and Data Mining (SIGKDD), p. 213–220. ACM (2008)
24. EU: Regulation (eu) 2016/679 of the european parliament and of the council of 27 april 2016 on the protection of natural persons with regard to the processing of personal data and on the free movement of such data, and repealing directive 95/46/ec (general data protection regulation) (text with eea relevance). The Official Journal of the European Union (OJ) **L 119**, 1–88 (2016)
25. Fan, Y., Xia, X., Alencar da Costa, D., Lo, D., Hassan, A.E., Li, S.: The impact of changes mislabeled by szz on just-in-time defect prediction. IEEE Transactions

on Software Engineering (2019). To appear
26. Fischer, M., Pinzger, M., Gall, H.: Analyzing and relating bug report data for feature tracking. In: Proceedings of the 10th Working Conference on Reverse Engineering (WCRE), pp. 90–99. IEEE (2003)
27. Fischer, M., Pinzger, M., Gall, H.: Populating a release history database from version control and bug tracking systems. In: Proceedings of the 2003 International Conference on Software Maintenance (ICSM), pp. 23–32. IEEE (2003)
28. Fu, W., Nair, V., Menzies, T.: Why is differential evolution better than grid search for tuning defect predictors? arXiv preprint arXiv:1609.02613 (2016)
29. Fukushima, T., Kamei, Y., McIntosh, S., Yamashita, K., Ubayashi, N.: An empirical study of just-in-time defect prediction using cross-project models. In: Proceedings of the 11th Working Conference on Mining Software Repositories (MSR), pp. 172–181. ACM (2014)
30. German, D.M., Adams, B., Stewart, K.: cregit: Token-level blame information in git version control repositories. Empirical Software Engineering **24**(4), 2725–2763 (2019)
31. Git community: git-log - Show commit logs. URL https://git-scm.com/docs/git-log
32. Gyimóthy, T., Ferenc, R., Siket, I.: Empirical validation of object-oriented metrics on open source software for fault prediction. IEEE Transactions on Software engineering **31**(10), 897–910 (2005)
33. Herbold, S., Trautsch, A., Trautsch, F.: Issues with szz: An empirical assessment of the state of practice of defect prediction data collection. arXiv preprint arXiv:1911.08938 (2019)
34. Herzig, K., Just, S., Zeller, A.: It's not a bug, it's a feature: how misclassification impacts bug prediction. In: Proceedings of the 2013 International Conference on Software Engineering (ICSE), pp. 392–401. IEEE Press (2013)
35. Herzig, K., Zeller, A.: The impact of tangled code changes. In: Proceedings of the 10th Working Conference on Mining Software Repositories (MSR), pp. 121–130. IEEE (2013)
36. Jung, Y., Oh, H., Yi, K.: Identifying static analysis techniques for finding non-fix hunks in fix revisions. In: Proceedings of the ACM First International Workshop on Data-intensive Software Management and Mining, pp. 13–18. ACM (2009)
37. Kamei, Y., Fukushima, T., McIntosh, S., Yamashita, K., Ubayashi, N., Hassan, A.E.: Studying just-in-time defect prediction using cross-project models. Empirical Software Engineering **21**(5), 2072–2106 (2016)
38. Kamei, Y., Shihab, E.: Defect prediction: Accomplishments and future challenges. In: Proceedings of the 23rd International Conference on Software Snalysis, Evolution, and Reengineering (SANER), pp. 33–45 (2016)
39. Kamei, Y., Shihab, E., Adams, B., Hassan, A.E., Mockus, A., Sinha, A., Ubayashi, N.: A large-scale empirical study of just-in-time quality assurance. IEEE Transactions on Software Engineering **39**(6), 757–773 (2013)
40. Kawrykow, D., Robillard, M.P.: Non-essential changes in version histories. In: Proceedings of the 33rd International Conference on Software Engineering (ICSE), pp. 351–360. IEEE (2011)

41. Kim, S., Whitehead Jr, E.J., Zhang, Y.: Classifying software changes: Clean or buggy? IEEE Transactions on Software Engineering **34**(2), 181–196 (2008)
42. Kim, S., Zhang, H., Wu, R., Gong, L.: Dealing with noise in defect prediction. In: Proceedings of the 33rd International Conference on Software Engineering (ICSE), pp. 481–490. IEEE (2011)
43. Kim, S., Zimmermann, T., Pan, K., James Jr, E., et al.: Automatic identification of bug-introducing changes. In: Proceedings of the 21st IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 81–90. IEEE (2006)
44. Kondo, M.: MKmknd/EMSE2021_ILA. DOI 10.5281/zenodo.5712318. URL https://doi.org/10.5281/zenodo.5712318
45. Kondo, M.: MKmknd/ILA. DOI 10.5281/zenodo.5573591. URL https://doi.org/10.5281/zenodo.5573591
46. Kondo, M.: MKmknd/ILA_Validation. DOI 10.5281/zenodo.5612161. URL https://doi.org/10.5281/zenodo.5612161
47. Kondo, M., Bezemer, C.P., Kamei, Y., Hassan, A.E., Mizuno, O.: The impact of feature reduction techniques on defect prediction models. Empirical Software Engineering **24**(4), 1925–1963 (2019)
48. Kondo, M., German, D.M., Mizuno, O., Choi, E.H.: The impact of context metrics on just-in-time defect prediction. Empirical Software Engineering **25**(1), 890–939 (2020)
49. Le, T.D.B., Linares-Vásquez, M., Lo, D., Poshyvanyk, D.: Rclinker: automated linking of issue reports and commits leveraging rich contextual information. In: Proceedings of the 23rd International Conference on Program Comprehension (ICPC), pp. 36–47. IEEE (2015)
50. McIntosh, S., Kamei, Y.: Are fix-inducing changes a moving target? a longitudinal case study of just-in-time defect prediction. IEEE Transactions on Software Engineering **44**(5), 412–428 (2018)
51. Mills, C., Pantiuchina, J., Parra, E., Bavota, G., Haiduc, S.: Are bug reports enough for text retrieval-based bug localization? In: 2018 IEEE International Conference on Software Maintenance and Evolution (ICSME), pp. 381–392. IEEE (2018)
52. Mockus, A., Votta, L.G.: Identifying reasons for software changes using historic databases. In: Proceedings of the 2000 International Conference on Software Maintenance (ICSM), pp. 120–130 (2000)
53. Neto, E.C., da Costa, D.A., Kulesza, U.: The impact of refactoring changes on the szz algorithm: An empirical study. In: Proceedings of the 25th International Conference on Software Analysis, Evolution and Reengineering (SANER), pp. 380–390. IEEE (2018)
54. Neto, E.C., da Costa, D.A., Kulesza, U.: Revisiting and improving szz implementations. In: Proceedings of the 2019 International Symposium on Empirical Software Engineering and Measurement (ESEM), pp. 1–12. IEEE (2019)
55. Nguyen, A.T., Nguyen, T.T., Nguyen, H.A., Nguyen, T.N.: 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. ACM (2012)

56. Nguyen, H.A., Nguyen, A.T., Nguyen, T.N.: Filtering noise in mixed-purpose fixing commits to improve defect prediction and localization. In: Proceedings of the 24th International Symposium on Software Reliability Engineering (ISSRE), pp. 138–147. IEEE (2013)
57. Nguyen, T.H., Adams, B., Hassan, A.E.: A case study of bias in bug-fix datasets. In: 2010 17th Working Conference on Reverse Engineering, pp. 259–268. IEEE (2010)
58. Ni, C., Xia, X., Lo, D., Chen, X., Gu, Q.: Revisiting supervised and unsupervised methods for effort-aware cross-project defect prediction. IEEE Transactions on Software Engineering (2020)
59. Pan, K., Kim, S., Whitehead, E.J.: Toward an understanding of bug fix patterns. Empirical Software Engineering **14**(3), 286–315 (2009)
60. Rahman, F., Posnett, D., Herraiz, I., Devanbu, P.: Sample size vs. bias in defect prediction. In: Proceedings of the 9th Joint Meeting on Foundations of Software Engineering, pp. 147–157. ACM (2013)
61. Ramler, R., Himmelbauer, J.: Noise in bug report data and the impact on defect prediction results. In: Proceedings of the 2013 Joint Conference of the 23rd International Workshop on Software Measurement and the 8th International Conference on Software Process and Product Measurement, pp. 173–180. IEEE (2013)
62. Rosen, C., Grawi, B., Shihab, E.: Commit guru: Analytics and risk prediction of software commits. In: Proceedings of the 10th Joint Meeting on Foundations of Software Engineering (ESEC/FSE), pp. 966–969. ACM (2015)
63. Schermann, G., Brandtner, M., Panichella, S., Leitner, P., Gall, H.: Discovering loners and phantoms in commit and issue data. In: Proceedings of the 23rd International Conference on Program Comprehension (ICPC), pp. 4–14. IEEE (2015)
64. scikit-learn developers: 3.2.4.3.1. sklearn.ensemble.RandomForestClassifier. URL https://scikit-learn.org/stable/modules/generated/sklearn.ensemble.RandomForestClassifier.html
65. scikit-learn developers: sklearn.linear_model.LogisticRegression. URL https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.LogisticRegression.html
66. scikit-learn developers: sklearn.linear_model.SGDClassifier. URL https://scikit-learn.org/stable/modules/generated/sklearn.linear_model.SGDClassifier.html
67. scikit-learn developers: sklearn.metrics.cohen_kappa_score. URL https://scikit-learn.org/stable/modules/generated/sklearn.metrics.cohen_kappa_score.html
68. scikit-learn developers: sklearn.metrics.pairwise.cosine_similarity. URL https://scikit-learn.org/stable/modules/generated/sklearn.metrics.pairwise.cosine_similarity.html
69. Śliwerski, J., Zimmermann, T., Zeller, A.: When do changes induce fixes? In: Proceedings of the 2005 International Workshop on Mining Software Repositories (MSR), pp. 1–5. ACM (2005)
70. Sparck Jones, K.: A statistical interpretation of term specificity and its application in retrieval. Journal of documentation **28**(1), 11–21 (1972)

71. Sun, Y., Chen, C., Wang, Q., Boehm, B.: Improving missing issue-commit link recovery using positive and unlabeled data. In: Proceedings of the 32nd IEEE/ACM International Conference on Automated Software Engineering (ASE), pp. 147–152. IEEE Press (2017)
72. Sun, Y., Wang, Q., Li, M.: Understanding the contribution of non-source documents in improving missing link recovery: An empirical study. In: Proceedings of the 10th ACM/IEEE International Symposium on Empirical Software Engineering and Measurement, pp. 1–10. ACM (2016)
73. Sun, Y., Wang, Q., Yang, Y.: Frlink: Improving the recovery of missing issue-commit links by revisiting file relevance. Information and Software Technology **84**, 33–47 (2017)
74. Sureka, A., Lal, S., Agarwal, L.: Applying fellegi-sunter (fs) model for traceability link recovery between bug databases and version archives. In: Proceedings of the 18th Asia-Pacific Software Engineering Conference (APSEC), pp. 146–153. IEEE (2011)
75. Tan, M., Tan, L., Dara, S., Mayeux, C.: Online defect prediction for imbalanced data. In: Proceedings of the 37th International Conference on Software Engineering (ICSE), pp. 99–108. IEEE (2015)
76. Tantithamthavorn, C., Hassan, A.E.: An experience report on defect modelling in practice: Pitfalls and challenges. In: Proceedings of the 40th International Conference on Software Engineering: Software Engineering in Practice (ICSE-SEIP), p. 286–295. ACM (2018)
77. Tantithamthavorn, C., Hassan, A.E., Matsumoto, K.: The impact of class rebalancing techniques on the performance and interpretation of defect prediction models. IEEE Transactions on Software Engineering **46**(11), 1200–1219 (2018)
78. Tantithamthavorn, C., McIntosh, S., Hassan, A.E., Ihara, A., Matsumoto, K.: The impact of mislabelling on the performance and interpretation of defect prediction models. In: Proceedings of the 37th International Conference on Software Engineering (ICSE), pp. 812–823. IEEE (2015)
79. Tantithamthavorn, C., McIntosh, S., Hassan, A.E., Matsumoto, K.: Automated parameter optimization of classification techniques for defect prediction models. In: Proceedings of the 38th International Conference on Software Engineering, pp. 321–332 (2016)
80. Tantithamthavorn, C., McIntosh, S., Hassan, A.E., Matsumoto, K.: An empirical comparison of model validation techniques for defect prediction models. IEEE Transactions on Software Engineering **43**(1), 1–18 (2017)
81. Thomas W., S.: lscp: A lightweight source code preprocesser. URL `https://github.com/doofuslarge/lscp`
82. Tu, H., Menzies, T.: Better data labelling with emblem (and how that impacts defect prediction). arXiv preprint arXiv:1905.01719 (2020)
83. Wohlin, C.: Guidelines for snowballing in systematic literature studies and a replication in software engineering. In: Proceedings of the 18th International Conference on Evaluation and Assessment in Software Engineering, pp. 1–10 (2014)
84. Wu, R., Zhang, H., Kim, S., Cheung, S.C.: 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 (ES-EC/FSE), pp. 15–25. ACM (2011)
85. Xie, R., Chen, L., Ye, W., Li, Z., Hu, T., Du, D., Zhang, S.: Deeplink: A code knowledge graph based deep learning approach for issue-commit link recovery. In: Proceedings of the 26th International Conference on Software Analysis, Evolution and Reengineering (SANER), pp. 434–444. IEEE (2019)
86. Yang, X., Lo, D., Xia, X., Zhang, Y., Sun, J.: Deep learning for just-in-time defect prediction. In: Proceedings of the International Conference on Software Quality, Reliability and Security (QRS), pp. 17–26. IEEE (2015)
87. Yedida, R., Menzies, T.: On the value of oversampling for deep learning in software defect prediction. IEEE Transactions on Software Engineering (2021)
88. Zhang, F., Zheng, Q., Zou, Y., Hassan, A.E.: Cross-project defect prediction using a connectivity-based unsupervised classifier. In: Proceedings of the 38th International Conference on Software Engineering (ICSE), pp. 309–320. ACM (2016)

Appendix

A Overview of ILAs

In this Appendix, we describe the details of each of the ILAs.

A.1 Keyword Extraction (KE)

The keyword extraction approach is the simplest and most popular approach to link issue reports with commits. Our implementation is as follows:

1. Extract issue ids from commit messages by using regular expressions.
2. Link issue reports to commits whose commit messages include issue ids.

Especially, our studied projects adhere to rules to write commit messages. For example, developers need to write "`ZOOKEEPER-jiraNumber: jiraTitle`" as the commit message in the ZooKeeper project [5]. Hence, the link rates are very high.

A.2 Time Filtering (TF)

The time filtering approach compares created/updated/resolved dates of issue reports or its comment dates and commit or author dates of commits. Our implementation is as follows:

1. Extract issue resolution dates and commit dates from an issue-tracking system and a software repository.
2. Subtract the date of a commit date from an issue resolution date to compute the difference (*time interval*).
3. If the time interval is less than a certain threshold, such a pair is linked.

This approach is frequently used in previous studies [55, 63, 69, 72, 74, 84]. For example, Wu et al. [84] reported that 93% of comments were posted within 24 hours after pushing the associated commits to the repository.

A.3 Natural Language Text Similarity (TS)

The natural language text similarity approach uses the similarity of text between issue reports and commits. If a commit message is similar to a description and comments of an issue report, such a pair would be linked.

In this approach, the preprocessing of text is important. A previous study [84] used the following preprocessing:

1. Remove stop words (e.g., remove "the").
2. Conduct stemming analysis (e.g., "played" would be "play").
3. Replace words into a common synonym.

In our implementation, we improved the preprocessing as follows:

1. Make text lower case
2. Tokenize words
3. Remove punctuation
4. Remove stop words
5. Replace words into a common synonym
6. Conduct stemming analysis

After this preprocessing, we converted the text into numerical vectors based on the TFIDF vectorization [70]. Finally, we computed the cosine similarity [68] between issue reports and commits and if the similarity value is over a certain threshold, we tagged such pairs as linked pairs. The threshold is 0.3 that was decided by our preliminary study.

The input text is:

– Description and comments from an issue report
– Commit message from a commit

A.4 Natural Language Text Similarity with Word Association (WA)

The natural language text similarity approach still has a challenge: wording between issue reports and commits are different. For example, let us assume we have an issue with a registration system. Then we might discuss a password system as well; however, the code fix would be applied to the registration system only. In this case, we need to make an association between "password" and "registration." *The natural language text similarity with word association approach* addresses this challenge. The original paper describes the detail concept [55]. The procedure that we used in this paper is as follows:

1. Extract the description and all the comments for each issue report and parse them by *lscp (lightweight source code preprocessor)* [81].
2. Extract all commit messages for all commits and parse them by lscp.
3. Execute the keyword extraction approach to prepare the training data.
4. Compute the formula (1) to (3) in the original paper [55] for each word pair.
5. If the value of formula (3) is over the threshold, such a word pair is considered as an associated word pair. The threshold is 0.5 that is decided by our preliminary study.

A.5 Message Generation from Source Code (GS)

Commit messages are not enough information. Hence a previous study [49] used a code comment generation technique to add more information. We call this approach as *the message generation from source code approach*. Recently, code comment generation techniques based on deep learning techniques become a popular research area; and therefore, we can also use such techniques. These techniques use *javadoc comments* as the supervised data in Java; and therefore, we use the javadoc comments instead of using code comment generation techniques in order to add clean information.

The procedure is the same as the natural language text similarity approach. One difference is that we used the javadoc comments instead of commit messages for each commit. Such comments were extracted from all modified files on the target commit.

A.6 Loner Heuristics (LO)

Schermann et al. [63] proposed two new scenarios called *Loner* and *Phantom scenarios*. The Loner scenario indicates the case where only one commit addresses one issue report; such an issue report links with only one commit. The Phantom scenario indicates the case where a set of commits addresses an issue report; such an issue report links with multiple commits. The original paper [63] extracted heuristics of such scenarios respectively in order to improve the accuracy of detecting defect-fixing commits.

We studied these heuristics as ILAs. Note that we excluded the heuristics about developers from these heuristics. The procedure of *the Loner heuristics* is as follows:

1. Execute the keyword extraction approach and remove the links that were detected by the keyword extraction approach (we call the output as (1)).
2. Apply the time filtering approach to (1) (we call the output as (2)). Since not only the time filtering approach but also other heuristics are applied, we used 30 minutes as the time interval.
3. If the pairs in (2) meet the following conditions, the pairs remain; otherwise, the pairs are excluded:
   - A pair has one issue report and one commit.
   - The issue report in the pair is not reopened.

We considered the remained pairs as the output of the Loner heuristics. Note that since it is difficult to extract the data whether an issue report is reopened, we skipped to check whether the issue report is reopened in our implementation.

A.7 Phantom Heuristics (PH)

The procedure of *the Phantom heuristics* is as follows:

1. Execute the keyword extraction approach and classify commits into linked commits and non-linked commits.
2. Make pairs between linked commits and non-linked commits (we call the output as (1)).
3. Exclude pairs of (1) if the dates of a linked commit and a non-linked commit are not within the interval (we call the output as (2)). The interval is five days.
4. Exclude pairs of (2) if the overlap of the modified files is less than `DUPLICATE_RATE` (the original paper [63] and we use 66% as `DUPLICATE_RATE`).

We considered the remained pairs as the output of the Phantom heuristics.

A.8 Modified Text Files (MT)

Sun et al. [72] stated that we have focused on commit messages; however, we did not focus on natural language files in the source code repository such as CHANGE.txt. *The modified text files approach* regards such files as a representation of a commit. The algorithm is the same as the natural language text similarity approach. However, we used different input:

- Description and comments from issue reports
- Natural language texts from commits (i.e., files with .txt or .md extension)

We used a different threshold value for the cosine similarity results. The threshold is 0.2 that was decided by our preliminary study.

A.9 PU Learning (PU)

PU learning (positive and unlabeled learning) is a learning method [23]. We can build a model based on positive examples and unlabeled examples in this learning method. Since there might exist many unlabeled (false-negative) links between issue reports and commits, prior study [71] used the PU learning to predict positive links based on such unclear data.

The procedure of the PU learning approach as an ILA is as follows:

1. Extract links by the keyword extraction approach.
2. Extract five features (the proportion of modified source files, the number of modified source files, the time difference, the time difference type, the cosine similarity of text).
3. Normalize all features except for binary features by z-score [47].
4. Train a PU model.
5. Classify all links by the PU model.

The proportion of modified source files is the proportion of modified Java files in a commit. The number of modified source files is the number of modified Java files in a commit. The time difference is the time difference of a commit date and an issue resolved date in seconds. The time difference type is a binary value; if an issue resolved date is after a commit date, it would be one; otherwise, it would be zero. The cosine similarity of text is the cosine similarity values that are computed on the natural language text similarity approach.

A.10 Machine Learning (ML)

The PU learning approach used a PU model to predict links. However, we can also apply other machine learning models to this task. We used two machine learning models: a random forest model [64] and a support vector machine model [66] instead of a PU model. The procedure is the same as the PU learning approach. The only difference is to use a PU model or machine learning models. We call this approach as *the machine learning approach*.