# Detecting Skipped Commits in Continuous Integration Using Multi-objective Evolutionary Search

Islem Saidani, Ali Ouni, and Mohamed Wiem Mkaouer

**Abstract**—Continuous Integration (CI) consists of integrating the changes introduced by different developers more frequently through the automation of build process. Nevertheless, the CI build process is seen as a major barrier that causes delays in the product release dates. One of the main reasons for such delays is that some simple changes (*i.e.*, can be skipped) trigger the build, which represents an unnecessary overhead and particularly painful for large projects. In order to cut off the expenses of CI build time, we propose in this paper, SKIPCI, a novel search-based approach to automatically detect CI skipped commits based on the adaptation of Strength-Pareto Evolutionary Algorithm (SPEA-2). Our approach aims to provide the optimal trade-off between two conflicting objectives to deal with both skipped and non-skipped commits. We evaluate our approach and investigate the performance of both within and cross-project validation scenarios on a benchmark of 16,334 CI commits from 15 large-scale CI projects. The statistical tests revealed that our approach shows a clear advantage over the baseline approaches with average scores of 91% and 82% in terms of AUC for cross-validation and cross-project validations respectively. Furthermore, the features analysis reveals that the number of previously skipped commits, the commit purpose, and the terms appearing in the commit message are the most influential features to indicate skip proneness. Moreover, we deployed and evaluated the usefulness of SKIPCI with our industrial partner. Qualitative results demonstrate the effectiveness of SKIPCI in providing relevant CI skip commit recommendations to developers for two large software projects from practitioner's point of view.

**Index Terms**—Continuous Integration, Travis CI, Software Build, Search-Based Software Engineering, Genetic Programming

## 1 INTRODUCTION

CONTINUOUS integration (CI) is a leading edge of modern software development practices that is widely adopted in industry and open-source environments [1, 2]. CI systems, such as Travis CI[1], advocate to continuously integrate code changes, by automating the process of building and testing [3], which reduces the cost and risk of delivering defective changes. Nevertheless, the build process is seen as a critical problem that hinders CI adoption. In fact, in such context, the build process is typically time and resource-consuming and particularly painful for large projects, with various dependencies, in which their builds can take hours and even days [4]. The presence of such slow builds severely affects both the speed and cost of software development as well as the productivity of developers [5]. Such challenges motivated the research [6, 7] on developing techniques to speed up CI process and cut its expenses by detecting commits that do not require the system's build *e.g.* commits affecting non-source files.

The first attempt in addressing this problem [6], formulated the CI skip detection problem with rules that were manually defined through mining the historical commit changes. To raise the challenges related to making the manual procedure of defining the rules as well as the high false negative rate, Abdalkareem et al. [7] proposed an approach based on a supervised learning technique to automatically detect CI skip commits by training a Decision Tree (DT) classifier. Based on 23 commit-level features, the derived Machine Learning (ML) approach achieved satisfactory results within-project validation with 79% of F1-measure. When it comes to cross-project validation, the prediction performance degraded to the accuracy of random guessing with 55% of F1; which challenges the applicability of this approach on projects with little or no previous commit history. However, in practice, there is a lack of training data as the CI skip option (*e.g.* adding the '`[ci skip]`' or '`[skip ci]`' in commit messages[2] in order to explicitly skip the commit by CI systems) is usually miss-used. For instance, Abdalkareem et al. [6] have found that among all the Java projects hosted on GitHub, only eleven projects have sufficient data with a rate of CI skipped commits $\geq 10\%$. Moreover, by mining TravisTorrent dataset [8], we have found that in over 1,283 projects that use Travis CI, only three projects satisfied this condition. In addition, the generalizability of these results remains limited as this approach was evaluated on a small dataset (about 3,000 commits) with an average size of 300 commits per project. This suggests that the skip detection problem is not yet resolved.

Detecting a skipped commit in practice is not a trivial task as the main difficulty lies in the large and complex search space to be explored due to exponential number

- *I. Saidani and A. Ouni are with the Department of Software Engineering and IT, ETS Montreal, University of Quebec, QC, Canada E-mail: islem.saidani.1@ens.etsmtl.ca, ali.ouni@etsmtl.ca*
- *M. W. Mkaouer is with Rochester Institue of Technology, Rochester, NY, USA.E-mail: mwmvse@rit.edu*

*Manuscript received April 19, 2005; revised August 26, 2015.*

1. https://travis-ci.org/

2. https://docs.travis-ci.com/user/customizing-the-build/#skipping-a-build

of possible combinations of features and their associated threshold values. Hence, the CI skip commits detection problem is by nature a combinatorial optimization problem in order to find the optimal detection rules that should maximize as much as possible the detection accuracy. Additionally, taking into account the conflict between the minority (*i.e.*, skipped) and majority (*i.e.*, non-skipped) classes accuracies [9, 10], multi-objective formulation is well suited to search-based software engineering (SBSE) [11, 12].

Recently, a number of researchers have highlighted the successful use of Multi-Objective Genetic Programming (MOGP) as an efficient method for developing prediction models [12, 13, 14, 15, 16]. This technique is particularly effective with imbalanced problems as it allows the evolution of solutions with optimal balance between minority and majority classes, without need to rebalance the data [17, 18]. This is crucial to keep the generated models accurate to the original data [19] and hence human interpretable.

Motivated by the need for help in efficiently identifying commit changes that could be skipped in the CI pipeline, we introduce in this paper, SKIPCI, a novel approach that formulates the problem of CI skip detection as a search-based problem to (1) maximize the probability of detection (*i.e.*, skipped commits that are correctly classified) and (2) minimize the false alarms (*i.e.*, the commits incorrectly classified as skipped). In this approach, we adapt the Strength-Pareto Evolutionary Algorithm (SPEA-2) [20] with a tree-based solution representation, to generate the optimal detection rules that should cover as much as possible the accurately detected commits from the base of real world CI commits examples.

To evaluate our approach, we conducted an empirical study to investigate its performance on both within and cross-project validation scenarios on a benchmark of 16,334 CI commits from 15 large-scale CI projects. Results show that SKIPCI achieves a better performance over various baseline approaches with average AUC scores of 91% and 82% for cross-validation and cross-project validations, respectively. Furthermore, the features analysis reveals that the number of previously skipped commits, the commit purpose, and the terms appearing in the commit message are the most influential features to indicate skip proneness. Moreover, we deployed SKIPCI as a bot and integrate it in the CI pipeline with our industrial partner, a large company specialized for printers and digital document products and services. The results of a qualitative evaluation of SKIPCI with 14 developers indicate the relevance of SKIPCI in practice as compared to baseline techniques.

### 1.1 Contributions

The main contributions of the paper can be summarized as follows:

1) A novel formulation of the CI skip detection as a multi-objective optimization problem to handle imbalance nature of CI skipped commits data.
2) An evaluation of SKIPCI, on a benchmark of 16,334 Travis CI commits of fifteen projects that use Travis CI, shows that our approach outperforms other search-based algorithms as well as ML techniques by achieving an average F1 scores of 85% and 69% for cross-validation and cross-project validation respectively.
3) A qualitative analysis to discover which features are the most prominent to determine CI skipped commits using our proper rules. The results reveal that (1) the number of previously skipped commits, (2) features indicting the commit purpose and (3) terms appearing in the commit message are very influential in predicting CI skipped commits.
4) An industrial evaluation of SKIPCI. Specifically, we deployed our tool in the CI pipeline of two projects with our industrial partner and validated its relevance with 14 developers. The study's outcome indicates that the SKIPCI is able to effectively recommend CI skipped commits with a high acceptance rate compared to a baseline approach from practitioners perspective.

### 1.2 Replication Package

We provide our replication package containing all the materials to reproduce and extend our study [21]. In particular, we provide (1) our SKIPCI tool as a lightweight command line tool with the necessary documentation to run the tool, (2) the working data sets of our study and (3) the validation results along with (4) examples of the built models.

### 1.3 Paper Organization

The remainder of this paper is organized as follows. In Section 2, we motivate the formulation of CI skip detection with a real-world example. Then, we explain how we adapted SPEA-2 to our problem in Section 3. Section 4 describes the experimental setup of our empirical study while Section 5 presents the results of this evaluation. In Section 6, we deploy SKIPCI in an industrial setting and evaluate it from developers' perspectives. We discuss the implications of our findings in Section 7. Section 8 surveys the related work and Section 9 describes the threats to validity. Section 10 concludes the paper.

## 2 MOTIVATING EXAMPLE

Although the ML-based model [7] was able to improve the CI skip detection compared to the rule-based approach [6], it still misses cases of commits that should be skipped in CI. Figure 1a depicts a concrete example extracted from `Semantic MediaWiki`[3], a PHP framework, where the commit, despite not being a cosmetic change, it is worth to be skipped. Looking at the commit change, the developer has modified two files:

- The first file is a source file (PHP) in which the developer added a default parameter (or optional) called *$default* to the signature of `get` function. This parameter is the result to be returned in case the key (a parameter called `$key`) is not in the list (of type `SchemaList`). We clearly see that this change neither alter the function behavior nor the behavior of the caller functions as it replaces the empty list with a parameter set by default to `[]`. This explains why this file was the only source file to be affected by this change, even though this function is called in other files.

3. https://github.com/SemanticMediaWiki/SemanticMediaWiki/commit/8e46f76067196f5677ee88ec667daefedf611b4d

- The second file is a test file, in which the developer (*i*) changed the value to be tested in order to check whether calling a non-existent key would return an empty list, and (*ii*) added another assertion to verify whether calling a non-existent key would return `null` in case `$default` is set to null. Looking at the content of this file, we see that the input list for these tests contains only one element set to `'Foo'`, which means these tests can be skipped.

On the other hand, as seen in Figure 1b, this commit unnecessarily kicked off the CI process by taking alone about one hour and 20 minutes to be built on Travis CI, which confirms the usefulness of an automated tool for skipped commits detection, to avoid these unnecessary builds and delays. However, using existing approaches [6, 7], we have found that they failed to detect this commit to be skipped. Indeed, the rule-based approach [6] consists of five rules related mainly to non-source files and cosmetic changes, which explains why this approach is not suitable to this problem. The machine learning approach [7] has also failed to provide an adequate detection. Looking at the generated model, we found that the commit is classified as non-skipped based on three conditions including (*i*) having non-documentation files, (*ii*) a developer experience (*i.e.*, number of previously committed changes) greater than 1,700 and (*iii*) a number of changed files greater than one, which mismatches with the characteristics of the provided example. This implies that solving this problem is not a trivial task and requires a more suitable approach to generate the adequate skip detection rules that learn from both classes (1) skipped commits and (2) non-skipped commits. In fact, the main difficulty lies in the complex search space as the number of possible combinations of features and their associated thresholds is large. Hence, the CI skip detection can be formulated as a search-based optimization problem to explore this large search space, in order to find the optimal detection rules. Additionally, a practical tool should also provide the developers with human explainable detection models to help them gaining insights into the CI commits to be skipped, especially when the changes are not trivial as shown in this example. This cannot be provided by ML techniques as re-sampling affects the interpretability of the generated models.

In the next section, we describe SKIPCI and show how we formulated the CI skip detection problem as a multi-objective combinatorial optimization problem to address the above mentionned problems.

## 3 THE SKIPCI APPROACH

In this section, we describe our SKIPCI approach to automate the CI skip detection by adapting Strength-Pareto Evolutionary Algorithm (SPEA-2) [20].

### 3.1 Approach Overview

Figure 2 depicts an overview of SKIPCI. The first input is a set of collected examples of commits that were annotated by their original developers as skip commits. As output, SKIPCI generate optimal rules using then SPEA-2 algorithm. The search algorithm evolves toward finding the best trade-off between two objective functions to (1) maximize the true positive rate, and (2) minimize the false positive rate. Then, given a code change (*i.e.*, commit) which is composed of a number of changed files, SKIPCI provides the user with an explained recommendation whether to skip or not the submitted change in the CI pipeline, *i.e.*, trigger the build process.

### 3.2 MOGP adaptation

This section shows how MOGP is adopted to CI skip commits detection problem using SPEA-2. To ease the understanding of this formulation, we first describe the pseudo-code of SPEA-2 and then present the solution encoding, the objective functions to optimize, and the employed change operators.

#### 3.2.1 SPEA-2 overview

In this paper, we use SPEA-2 as an intelligent search algorithm, that has been widely adopted to solve many software engineering problems [22, 23, 24, 25, 26, 27], to identify CI skipped commits. As described in Algorithm 1, SPEA-2 starts with an initial population $P_0$ and an empty archive $\bar{P}_0$ (line 1). The external archive is a collection of high quality solutions to be maintained and used exclusively for mating of future generations. The archive size is typically, but not necessary, equal to the population size. Then, the following steps are performed in each iteration $t$. The fitness assignment (line 3), which is based on the *Pareto dominance* principle for both the population and the archive in order to quantify the quality of candidate solutions in the current population. Note that a non-dominated solution is a solution that has fitness values such that no other solution within the set has better fitness values across all objectives. During the environmental selection step (line 4), all non-dominated solutions from the population and archive are copied to the archive of the next generation: $\bar{P}_{t+1}$. If the non-dominated set size fits exactly into the archive ($|\bar{P}_{t+1}| = N$) the environmental selection step is completed. Otherwise, there can be two situations: Either the archive $\bar{P}_{t+1}$ is too small or too large. In the first case, the best $N - |\bar{P}_{t+1}|$ dominated individuals in the previous archive and population are copied to the new archive. In the second case, an archive truncation procedure is employed to iteratively remove individuals from $\bar{P}_{t+1}$ until $|\bar{P}_{t+1}| = N$. Next, if the stopping condition is not met then mating selection is performed (line 5). Binary tournament selection with replacement on $\bar{P}_{t+1}$ is used to fill the mating pool. Finally crossover and mutation operators are applied to the mating pool and set $\bar{P}_{t+1}$ to the resulting population. The generation counter is increased (line 7) and this process will be repeated until some condition is satisfied. (line 2).

#### 3.2.2 Adaptation

To adopt a search algorithm to a given problem, a set of elements need to be defined. In fact, it is insufficient to merely apply the search-based techniques *out of the box*, and problem-specific adaptations need to defined to ensure best performance such as (*i*) solution representation, (*ii*) solution evolution, and (*iii*) solution evaluation.

**Solution representation:** In MOGP, a candidate solution, *i.e.*, a detection rule, is represented as an IF – THEN rules with the following structure [14, 28, 29, 30]:

**Algorithm 1** Pseudo code of SPEA-2

```
Input: N: population size, $\bar{N}$: archive size, T: maximum number of generations
Output: A: non-dominated set
1: Initialization: Generate an initial population $P_0$, create an empty archive (external set) $\bar{P}_0 = \emptyset$. Set $t = 0$
2: while $t < T$ or another stopping criteria is not reached do
3:     Fitness assignment: Calculate fitness values of individuals in $P_t$ and $\bar{P}_t$
4:     Environmental selection: Copy all non-dominated individuals in $P_t$ and $\bar{P}_t$ to $\bar{P}_{t+1}$
       IF size of $\bar{P}_{t+1} > \bar{N}$ THEN reduce the size of $\bar{P}_{t+1}$
       ELSE IF if size of $\bar{P}_{t+1} < \bar{N}$ THEN fill $\bar{P}_{t+1}$ with dominated individuals from $P_t$ and $\bar{P}_t$
5:     Mating selection: IF the stopping condition is not satisfied THEN perform binary tournament selection with replacement on $\bar{P}_{t+1}$in order to fill the mating pool ELSE Stop.
6:     Variation: Apply crossover and mutation into $\bar{P}_{t+1}$
7:     t = t+1
8: end while
9: Termination: $A = \bar{P}_{t+1}$
```

**IF** *"Combination of metrics with their thresholds"* **THEN** *"RESULT"*.

The antecedent of the IF statement describes the conditions, *i.e.* pairs of metrics and their threshold values connected with mathematical operators (*e.g.*, $=, >, \geq, <, \leq$), under which a CI commit is said to be skipped or not. These pairs are combined using logic operators (OR, AND in our formulation). Figure 3 provides an example of a solution. This rule, represented by a binary tree, detects a skipped CI commit if it fulfills the situation where (1) the number of added lines in the changed files (`LA`) is less or equals to 146, (2) the commit does not change source files (`is_src`), and (3) the commit is not a bug fixing commit (`is_fix`).

**IF** `LA ≤146` AND `is_src =0` AND `is_fix =0` **THEN** Skip commit.

To generate the initial population, we start by randomly assigning a set of metrics and their thresholds to the different nodes of the trees. To control for complexity, each solution size, *i.e.* the tree's length, should vary between a lower and upper-bound limits based on the total number of metrics to use within the detection rule. More precisely, for each solution, we assign:

- For each leaf node one metric and its corresponding threshold. The latter is generated randomly between lower and upper bounds according to the values ranging of the related metric.
- Each internal node (function) is randomly selected between AND and OR operators.

**Solution Evolution:**

*Mutation:* In MOGP, this operator can be applied either to a terminal or a function node. It starts by randomly selecting a node in the tree. Then, if the selected node is a terminal, it is replaced by another terminal (other metric or other threshold value, or both); if it is a function (AND, OR operators) node, it is replaced by a new function. Then, the node and its sub-tree are replaced by the new randomly generated subtree. Figure 4 illustrates an example of a mutation process, in which we replace the terminal containing `is_fix` feature, by another terminal composed of the condition $NS \leq 20$. Thus, we obtain the new rule:

**IF** `LA ≤146` AND `is_src =0` AND `NS ≤20` AND `LT≤5` **THEN** Skip Commit.

*Crossover:* For MOGP, we use the standard single-point crossover operator where two parents are selected and a sub-tree is extracted from each one. Figure 5 depicts an example of the crossover process. In fact, rules P1 and P2 are combined to generate two new rules. For instance, the new rule C2 will be:

**IF** `NUC ≤ 2` OR `ic_src =0` **THEN** Skip Commit.

**Solution Evaluation:** Appropriate fitness function, also called objective function, should be defined to evaluate how good is a candidate solution. For the CI skip commit problem, we seek to optimize the two following objective functions:

1) Maximize the coverage of expected CI skipped commits over the actual list of detected skipped commits known as the True Positive Rate (TPR), or als the probability of detection (PD).

$$TPR(S) = \frac{\{Detected\ Skipped\ Commits\} \cap \{Expected\ Skipped\ Commits\}}{\{Detected\ Skipped\ Commits\}}$$

2) Minimize the coverage of actual non-skipped commits that are incorrectly classified as skipped also known as False Positive Rate (FPR), or the probability of false alarm (FP).

$$FPR(S) = \frac{\{Detected\ Skipped\ Commits\} \cap \{Expected\ non\text{-}skipped\ Commits\}}{\{Detected\ Skipped\ Commits\}}$$

Additionally, since SPEA-2 returns a set of optimal (*i.e.* non-dominated) solutions in the *Pareto front* without ranking, we extract a single best solution which is the nearest to the ideal solution known as *True Pareto* in which TPR value equals to 1 and FPR equals to 0. Formally, the distance is computed in terms of Euclidean distance [15, 29] as follows:

$$BestSol = \min_{i=1}^{n} \sqrt{(1 - TPR[i])^2 + FPR[i]^2}$$

where *n* represents the cardinality the Pareto front generated by SPEA-2.

## 3.3 Metrics for CI Skip commits detection

Table 1 lists the metrics, *i.e.*, features, used to learn and generate our detection rules. Besides the metrics used by Abdalkareem et al. [7], we also generated other features related to the history of commit results to better capture the salient characteristics of CI skip commits. The selected features can be categorized as follows:

**Statistics about the current commit:** These features give a different information about the current commit change size (*e.g.*, number of lines added/deleted, entropy), the

TABLE 1: Metrics used form CI Skip detection extracted from literature

| Category | Metric | Description (References) |
|---|---|---|
| *Statistics about current commit* | NS | # of changed sub-systems.([7, 35, 36]) |
| | ND | # of changed directories.([7, 35, 36, 37, 38]) |
| | NF | # of changed files ([7, 35, 36, 37, 38]) |
| | Entropy | Measures the distribution of the change across the different files. ([7, 35, 36, 37, 38]) |
| | LA | # of added lines([5, 7, 32, 33, 35, 36, 37, 38, 39]) |
| | LD | Number of deleted lines ([5, 7, 32, 33, 35, 36, 37, 38, 39]) |
| | day_week | Day of the week when the commit was performed. ([32]) |
| | CM | Measures the importance of terms appearing in the commit message using TF-IDF ([7, 40]) |
| | TFC | # of the changed files' types identified by their extension.([4, 7, 32]) |
| *Commit Purpose* | Classif | Feature Addition (1), Corrective (2), Merge (3), Perfective (4), Preventative (5), Non-Functional (6), None (7) ([41]) |
| | is_fix | Whether the commit is a bug fixing commit ([7, 35, 36, 41]) |
| | is_doc | Whether the commit is a documentation commit *i.e.* contains only doc files ([6, 7]) |
| | is_build | Tests whether the commits contains only build files ([6, 7]) |
| | is_meta | Whether the commit a meta commit (contains only meta files) ([6, 7, 42]) |
| | is_merge | The commit is a merge commit ([7]) |
| | is_media | Whether the commit a media commit *i.e.* contains only media files like images |
| | is_src | Whether the commit a source commit *i.e.* contains only source code files |
| | FRM | If the commit changes the formatting of the source code ([6, 7]) |
| | COM | If the commit modifies only source code comments([6, 7]) |
| | CFT | If the commit is a maintenance/refactoring activity ([7]) |
| *Link to Last Commit(s)* | proj_recent_skip | Number of recently commits that were skipped in the 5 past commits ([31]) |
| | comm_recent_skip | Number of recently commits that were skipped in the 5 past commits by the current committer([31]) |
| | prev_commit_res | Whether previous commit was skipped or not.([31]) |
| | same_committer | Whether it is the same committer of the last commit ([31]) |
| | NUC | # of unique last commits of the modified files ([7, 35, 36]) |
| | AGE | the average time interval between the current and the last time these files were modified ([7]) |
| | NDEV | # of developers that previously changed the touched file ([7, 41]) |
| | LT | Size of changed files before the commit ([7, 41]) |
| | EXP | # of commits made by the developer before the current commit ([7, 31]) |
| | SEXP | Subsystem experience measures the number of commits the developer made in the past to the subsystems that are modified by the current commit ([7]) |
| | REXP | Recent experience is measured as the total experience of the developer in terms of commits, weighted by their age ([7]) |

importance of terms in the commit message (*i.e.*, CM) that has proved its efficiency to predict CI skipped commits in the recent work of Abdalkareem et al. [7] and other general statistics (*e.g.*, the day of the week).

**Commit purpose:** These features provide insights into the commit skip proneness. For instance, merge commits, commits containing only media or those changing documentation files are most likely to be skipped.

**Link to last to commit(s):** In addition to the previously used metrics that are linked to last commits [7], such as the committer experience, we add and deduce other detailed metrics including the number of recently skipped commits (in the project and by the current committer), the metric indicating whether the last commit was skipped or not and whether the current commit is requested by the same committer of the previous one. These metrics are inspired from existing works of CI build failure detection [31, 32, 33] in which the authors found that there is a strong link between the current build and the previous ones, and it was helpful to predict the failure in practice. In this paper, we hypothesize that likely to CI build failure, skipped commits may come consecutively *e.g.* committing different changes of documentation or applying sequences of refactoring (*i.e.* modifying the code structure without changing its function [34]).

## 4 Experimental Study Design

In this section, we describe the design of our empirical study to evaluate our SkipCI approach. All the experimental material are provided in our replication package [21].

### 4.1 Research Questions

In this study, we define four Research Questions (RQs). In the first two RQs, we conduct a 10-fold cross validation by dividing the data of each project into 10 equal folds. We use 9 folds to train each algorithm, and use the remaining one fold to evaluate the predictive performance. Then, we perform cross-project validation in RQ3, and investigated the features influence in RQ4.

**RQ1 (SBSE validation).** *How effective is* SkipCI *compared to other existing search-based algorithms?*

**Motivation.** The aim in this question to evaluate the SPEA-2 formulation from an SBSE perspective as recommended by Harman and Jones [11]. First, we compare SPEA-2 against Random Search (RS) [43, 44], to evaluate the need for an intelligent method against the unguided search of solutions. Additionally, it is important to compare our multi-objective formulation mono-objective GP (GA) since if considering separate conflicting objectives fails to outperform aggregating them into a single objective function, then the proposed formulation is inadequate. Then, we evaluate the performance of SPEA-2 with three widely-used multi-objective algorithms [12, 12, 45, 46] namely Non-dominated Sorting Genetic Algorithm (NSGA-II) [47], NSGA-III [48] and Indicator-Based Evolutionary Algorithm (IBEA) [49] in terms of the quality of non-dominated solutions known as *Pareto front* in the objective space [50].

**Approach.** To answer **RQ1**, we first compute three widely used performance evaluation metrics namely **F1-score**, Area Under the ROC Curve (**AUC**) and the **Accuracy** metrics. The first measure is defined as follows:

$$\textit{F1-score} = 2 * \frac{Precision * Recall}{Precision + Recall} \in [0, 1] \tag{1}$$

In our study, the recall is the percentage of correctly classified CI skip commits over the total number of commits that are skipped, while the precision is the percentage of detected CI skip commits that are actually skipped. These metrics are computed as:

$$Recall = \frac{TP}{TP + FN} \in [0, 1] \tag{2}$$

$$Precision = \frac{TP}{TP + FP} \in [0, 1] \tag{3}$$

where TP is the number of the skipped commits that are correctly classified, TN is the number of non-skipped commits that are correctly classified while FP and FN represent the number of incorrectly classified commits for skipped and non-skipped commits respectively.

**AUC** measure assesses how well a model/rule performs on the minority and majority classes and is defined as follows [51]:

$$AUC = \frac{1 + \frac{TP}{TP+FN} - \frac{FP}{FP+TN}}{2} \in [0, 1] \tag{4}$$

The **Accuracy** refers to the proportion of correct predictions made by the model and defined as follows:

$$Accuracy = \frac{TP + TN}{TP + TN + FP + FN} \in [0, 1] \tag{5}$$

Moreover, since SPEA-2, NSGA-II, NSGA-III and IBEA are Pareto-based search-based approaches, it is necessary to evaluate the quality of solution sets with respect to Pareto dominance [52]. This evaluation is generally based on quality indicators that are well-adopted in SBSE community. In this paper, we select three quality indicators based on previous practical guides [53, 54, 55].

- **Hyper-volume (HV):** is the most employed metric according to a previous study by Riquelme et al. [54] which calculates the space covered by the non-dominated solutions. Note that a higher value of HV indicates a better performance of the algorithm.
- **Generational Distance (GD):** occupies the second ranking of the most used metrics [54]. GD calculates how far are the Pareto front solutions from the true Pareto front (in our case, it is the optimal detection rule having $TPR = 1$ and $FPR = 0$). The smaller is GD, the better is the algorithm
- **Spacing (SP):** captures another important aspect of quality *i.e.* uniformity of solutions [56]. In a nutshell, SP measures how evenly the members of a Pareto front are distributed. A value of 0 for SP means that all solutions are uniformly spaced *i.e.* the algorithm possess an optimal quality.

It is worth to mention that all the search-based algorithms and quality indicators used in this study are implemented using MOEA Framework[4], an open-source framework for developing and experimenting with search-based algorithms [50, 57].

4. http://moeaframework.org/

**RQ2 (Evaluation with ML).** *How does our approach perform compared to ML techniques?*

**Motivation.** After evaluating our approach in terms of SBSE performance, it is important to evaluate its efficiency in solving the problem against the state-of-art solution *i.e.* the Machine Learning (ML) based techniques used in the previous work ofAbdalkareem et al. [7]. This comparison is important to motivate the need of a search-based approach to improve the CI skip detection.

**Approach.** To answer **RQ2**, we evaluate the predictive performance of our MOGP formulation against five ML techniques, widely used software engineering research [5, 32, 39, 42, 58, 59, 59], namely Decision Tree (DT), Random Forest (RF), Naive Bayesian (NB), Logistic Regression (LR) and Support Vector Classification (SVC). As ML models are sensitive to the scale of the inputs, the data are normalized in the range $[0, 1]$ by using feature scaling. In addition, to mitigate the issue related to the imbalanced nature of the dataset, we rely on Synthetic Minority Oversampling Technique (SMOTE) method [60], to resample the training data. It is worth to mention that we only resample the training data in order to assess these algorithms in a real-world situation. To compare the predictive performance of SKIPCI with ML techniques, we use **AUC, F1-measure and Accuracy** that were defined previously.

**RQ3 (Cross-project evaluation).** *How effective is our approach when applied on cross-projects?*

**Motivation.** In **RQ3**, we investigate the extent to which CI Skip commit identifications can be generalized through a cross-project prediction. In fact, many projects do not have sufficient historical labeled data to build a classifier [7] (*e.g.*, small or new projects), which may prevent the project team from using a prediction tool. Cross-project validation the-state-of-art technique to solve the lack of training data in software engineering [58].

**Approach.** To evaluate our approach on the cross-project scenario, we trained each project based on the other studied projects and test our SKIPCI approach on the remaining one. To gain better insights into the performance of our approach, we compare it with the ML techniques used in RQ2 based on F1-measure, AUC and Accuracy in this RQ.

**RQ4 (Features analysis).** *What features are most important to detect skipped commits?*

**Motivation.** The goal of **RQ4** is to analyze the most influencing metrics to detect commits to be skipped. The perspective is for researchers interested in understanding how CI commits metrics can be related to building activities and for developers, who might want to identify the indications that can help them in their decisions to skip their committed changes.

**Approach.** We address **RQ4** by exploring and interpreting the valuable knowledge provided by our generated models, *i.e.*, detection rules. Since we use 10-fold cross-validation and cross-project validation, the analysis produces 11 optimal rules. Additionally, the same feature may occur multiple times in the obtained rules. Thus, to analyze the features importance, we consider that the higher the number of occurrences of a feature in the optimal rules,

TABLE 2: Statistics of the studied projects.

| Project | Language | Study period | # of commits | CI skip percentage(%) |
|---|---|---|---|---|
| tracee/contextlogger | Java | 2014-12-12 - 2017-03-14 | 217 | 29 |
| jMotif/SAX | Java | 2015-06-20 - 2018-02-20 | 427 | 19 |
| traneio/future | Java | 2017-02-05 - 2018-10-02 | 282 | 21 |
| ksclarke/solr-iso639-filter | Java | 2013-08-20 - 2015-06-16 | 423 | 40 |
| jMotif/GI | Java | 2015-06-20 - 2018-05-27 | 351 | 10 |
| GrammarViz2/grammarviz2_src | Java | 2014-08-22 - 2018-03-06 | 425 | 13 |
| eBay/parallec | Java | 2015-10-28 - 2018-01-13 | 145 | 50 |
| danimahardhika/candybar-library | Java | 2017-02-22 - 2018-02-05 | 287 | 70 |
| RWTH-i5-IDSG/steve | Java | 2015-10-07 - 2020-04-01 | 814 | 8 |
| mtsar/mtsar | Java | 2015-05-06 - 2019-07-17 | 392 | 33 |
| ankane/searchkick | Ruby | 2013-08-12 - 2020-03-31 | 1,841 | 24 |
| ankane/groupdate | Ruby | 2013-04-25 - 2020-03-18 | 646 | 20 |
| SemanticMediaWiki/SemanticMediaWiki | PHP | 2013-06-17 - 2020-04-19 | 7,947 | 14 |
| activerecord-hackery/ransack | Ruby | 2011-07-17 - 2020-04-04 | 1,491 | 15 |
| ankane/pghero | Ruby | 2016-02-19 - 2020-04-06 | 646 | 21 |

the more important is the feature in identifying CI skip commits. In addition, to give a more general view, we aggregate the results of features occurrences for each project and feature category (cf. Section 3.3). Note that as our approach produces 31 rules at each experimentation, we choose the best rule across these repetitions based on the obtained AUC scores.

### 4.2 Studied Projects

Our experiments are mainly based on the dataset provided by Abdalkareem et al. [7] which comprises ten open-source Java projects using Travis CI. Moreover, we extended this dataset based on the same filter by Abdalkareem et al. [7] while considering different programming languages, *i.e.*, projects that (*i*) use Travis CI system, the most popular CI service on GitHub [61], and (*ii*) have at least 10% of skipped commits on the master branch, which is required to feed our tool with sufficient historical CI skip records. The projects were gathered using the Big Query Google[5] to query on the GitHub data [62]. Our filters result in a total of 15 projects including the 10 projects studied in Abdalkareem et al. [7]. Note that we only consider commits from the date a given project starts using Travis CI. Table 2 provides some statistics about the studied projects. Our replication package is publicly available at [21].

### 4.3 Statistical Test methods Used

Due to the stochastic nature of search-based algorithms, DT and RF techniques, we compare the performance of these algorithms by running them 31 independent runs for each experimentation then we choose the rule/model with the median value as suggested in [63]. Additionally, in order to provide support for the conclusions derived from the obtained results, we use Wilcoxon signed rank test [64], a non-parametric statistical test method to detect significant performance differences with a 95% confidence level ($\alpha$ is set at 0.05). Vargha-Delaney A (VDA) [65] is also used to measure the effect size. This non-parametric method is widely recommended in SBSE context [66]. It indicates the probability that one technique will achieve better performance than another technique. When the A measure is 0.5, the two techniques are equal. When the A measure is above or below 0.5, one of the techniques outperforms the other [67]. Vargha-Delaney statistic also classifies the magnitude of the obtained effect size value into four different levels: *negligible, small, medium*, and *large* [28, 68].

5. https://bigquery.cloud.google.com

### 4.4 Parameter Tuning and Setting

One of the most important aspects of research on SBSE is *parameters tuning* which has a critical impact on the algorithm's performance [69]. This is also compulsory when using ML techniques [19]. There is no optimal parameters' setting to solve all problems, therefore, we used a trial-and-error method to select the hyper-parameters [12] to handle parameter tuning for search-based algorithms which is a common practice in SBSE [12]. These parameters are fixed as follows: population size = 100; maximum # of generations = 500; crossover probability =0.7; and mutation probability = 0.1.

As for ML techniques, we employed *Grid Search* [70], an exhaustive search-based tuning method widely used in practice. We report in Table 3, the parameters' setting for the ML techniques used in this study.

TABLE 3: Parameters' settings from ML techniques under comparison

| Algorithm | Parameters |
|---|---|
| RF | Max depth of the tree =10<br># of estimators = 400 |
| DT | Max depth of the tree=10 |
| NB | Used NB classifier= Gaussian NB |
| LR | Max iterations= 200<br>penalty = 'l2' |
| SVC | C=1<br>kernel='rbf'<br>Max iterations= 2000 |

## 5 Experimental Study Results

This section presents and discusses the experimental results to answer our research questions.

### 5.1 Results for SBSE comparison (RQ1)

Figure 6 plots the predictive performance of the search-based algorithms under comparison over 4,650 experiment instances (31 runs $\times$ 10 folds $\times$ 15 projects).

As shown in Figure 6, GA achieved in median 82%, 67% and 82% in terms of AUC, F1 and accuracy, respectively, while RS has shown low predictive performance of 55%, 37% and 40% in terms of AUC, F1 and accuracy, respectively. In comparison with the multi-objective formulation, we clearly see that MOGP techniques (IBEA, SPEA-2, NSGA-II, and NSGA-III) outperform the mono-objective algorithm (GA) by achieving high comparable scores with a median of 90%, 82% and 92% in terms of AUC, F1 and accuracy respectively. Moreover, the statistical tests' results (Table 4) reveal that over 31 runs (of each project and each validation iteration), MOGP techniques show significant improvement over GA and RS with large VDA effect sizes. These findings confirm the suitability of multi-objective formulation for the detection problem as it can provide a better compromise between TPR and FPR. Therefore, our problem formulation passes the "sanity check".

TABLE 4: Statistical tests results of SPEA-2 compared to other search-based techniques under cross-validation.

| | | **vs. IBEA** | **vs. NSGA-II** | **vs. NSGA-III** | **vs. GA** | **vs. RS** |
|---|---|---|---|---|---|---|
| AUC | *p-value* | 1 | 1 | 1 | $> 10^{-16}$ | $> 10^{-16}$ |
| | *A estimate* | 0.5 | 0.51 | 0.5 | 0.75 | 0.99 |
| | *Magnitude* | N | N | N | L | L |
| F1 | *p-value* | 1 | 1 | 1 | $> 10^{-16}$ | $> 10^{-16}$ |
| | *A estimate* | 0.5 | 0.5 | 0.5 | 0.74 | 0.95 |
| | *Magnitude* | N | N | N | L | L |
| Accuracy | *p-value* | 1 | 1 | 1 | $> 10^{-16}$ | $> 10^{-16}$ |
| | *A estimate* | 0.5 | 0.5 | 0.5 | 0.77 | 0.99 |
| | *Magnitude* | N | N | N | L | L |

L: Large, M: Medium, S: Small, N: Negligible

With regards to MOGP algorithms, Table 5 shows the results of comparison based on the Hyper-Volume (HV), Generational Distance (GD) and Spacing (SP) as described in Section 4.1. As reported in the table, the best scores of HV, GD and SP were obtained by SPEA-2. In fact, SPEA-2 obtained in median a HV score of 0.96, compared to 0.93 achieved by the remaining MOGP techniques. In terms of GD, SPEA-2 achieved a better distance between its generated solutions and the optimal one ( *i.e.* Pareto front) by reaching 0.10 compared to 0.11, 0.15 and 0.20 for NSGA-II, NSGA-III and IBEA, respectively. As for the SP, the median scores are barely distinguishable between all the algorithms, so they achieve a similar spacing between the generated solutions, even though SPEA-2 is slightly better with a median of 0.06 as compared to 0.05 for the other algorithms under comparison.

Overall, we observe that SPEA-2 provides the highest median performance among the compared MOGP algorithms, which motivates our choice to adopt it as a search method.

**Summary for RQ1.** *Our multi-objective formulation has shown its effectiveness compared to mono-objective and random search algorithms by reaching 90% of AUC, 82% of F1 and 92% of accuracy. SPEA-2 achieved also higher optimization performance among the studied MOGP algorithms, which motivates our choice to use it a search-based approach.*

TABLE 5: Results of Pareto-based comparison in terms of hyper-volume (HV), Generational Distance (GD), and SPacing (SP).

| | **SPEA-2** | **NSGA-II** | **NSGA-III** | **IBEA** |
|---|---|---|---|---|
| *HV* | **0.96** | 0.93 | 0.93 | 0.93 |
| *GD* | **0.10** | 0.11 | 0.15 | 0.20 |
| *SP* | **0.06** | 0.05 | 0.05 | 0.05 |

### 5.2 Results for ML comparison within project validation (RQ2)

Table 6 reports the average (of 10 cross-validation iterations) AUC, F1 and accuracy scores achieved by each of the studied approaches; while Table 7 shows the statistical tests' comparison using the Wilcoxon signed rank test and Vargha-Delaney A effect size.

With regards to AUC, we clearly see that, for 12 out of 15 projects, the best scores were obtained by SPEA-2 achieving on average 91% with an improvement of at least 5% over the best ML algorithm (SVC). Additionally, the statistical analysis underlines the significant difference with *medium* to *large* VDA effect sizes (cf. Table 7). For instance, in the `steve` project, our approach obtained a AUC score of 90% while we recorded scores of 82%, 79%, 71% for SVC, LR and NB respectively and 66% for NB and RF. Overall, the results for AUC reveal that SPEA-2 can reach the best balance between both minority (skipped commit) and majority (non-skipped) class accuracies, than all the ML techniques. It is worth noting that all ML techniques are trained using re-sampled training sets unlike in SPEA-2, which confirms that multi-objective formulation is more suited to handle the imbalance problem [28, 71].

Looking at F1-scores, we also observe that SPEA-2 achieved the best results for 13 out of 15 projects with an average score of 85% compared to 76% achieved by SVC the best performer among ML techniques. Additionally, the F1 scores of our approach range from 69% to 100%. The statistical tests' results reveal that SPEA-2 outperforms ML with medium (compared to SVC, LR, RF) to large (with DT, NB) effect sizes. Hence, F1-score results demonstrate a compelling superiority of SPEA-2 to identify more skipped commits.

As for the accuracy scores, the obtained results also show that SPEA-2 is a better performer than the five considered ML techniques with a significant improvement of 5% on average, and medium to large effect sizes as shown in Table 7. Additionally, the accuracy scores of our approach range from 86% to 100% while achieving in median a high score of 92%. For all the studied projects, the accuracy values of SPEA-2 exceed those of ML techniques.

**Summary for RQ2.** SKIPCI *can achieve higher predictive performance than the studied ML techniques with a statistically significant difference within-validation, with an average of 90% and 85% in terms of AUC and F1-score, respectively, while reaching 93% of the overall classification accuracy.*

### 5.3 Results of cross-project validation (RQ3)

In this RQ, we compare SKIPCI with ML techniques under cross-project validation, using our evaluation metrics, the

TABLE 6: Performance of SPEA-2 vs. ML techniques under cross-validation.

| Project | AUC | | | | | | F1 | | | | | | Accuracy | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *DT* | *RF* | *LR* | *NB* | *SVC* | *SPEA-2* | *DT* | *RF* | *LR* | *NB* | *SVC* | *SPEA-2* | *DT* | *RF* | *LR* | *NB* | *SVC* | *SPEA-2* |
| candybar-library | 0.61 | 0.74 | 0.77 | 0.65 | 0.78 | **0.88** | 0.69 | 0.80 | 0.83 | 0.84 | 0.83 | **0.93** | 0.61 | 0.75 | 0.78 | 0.76 | 0.77 | **0.90** |
| contextlogger | **1.00** | **1.00** | **1.00** | 0.50 | **1.00** | **1.00** | **1.00** | **1.00** | **1.00** | 0.00 | **1.00** | **1.00** | **1.00** | **1.00** | **1.00** | 0.70 | **1.00** | **1.00** |
| future | 0.79 | 0.92 | 0.93 | 0.74 | 0.94 | **0.99** | 0.67 | 0.91 | 0.85 | 0.53 | 0.89 | **0.98** | 0.79 | 0.96 | 0.93 | 0.63 | 0.95 | **0.99** |
| GI | 0.72 | 0.86 | 0.96 | 0.50 | **0.99** | 0.98 | 0.50 | 0.83 | 0.88 | 0.00 | **0.95** | 0.94 | 0.91 | 0.97 | 0.97 | 0.90 | **0.99** | **0.99** |
| grammarviz2 | 0.85 | 0.91 | 0.91 | 0.74 | 0.88 | **0.95** | 0.54 | 0.80 | 0.82 | 0.47 | 0.81 | **0.91** | 0.74 | 0.95 | 0.93 | 0.88 | 0.95 | **0.97** |
| groupdate | 0.61 | 0.74 | 0.76 | 0.64 | 0.78 | **0.87** | 0.40 | 0.56 | 0.54 | 0.41 | 0.57 | **0.76** | 0.71 | 0.78 | 0.79 | 0.50 | 0.82 | **0.90** |
| mtsar | 0.57 | 0.71 | 0.71 | 0.58 | 0.73 | **0.86** | 0.46 | 0.61 | 0.61 | 0.52 | 0.62 | **0.81** | 0.59 | 0.72 | 0.72 | 0.47 | 0.74 | **0.87** |
| parallec | 0.83 | 0.95 | 0.93 | 0.54 | 0.93 | **0.99** | 0.80 | 0.93 | 0.90 | 0.09 | 0.92 | **0.98** | 0.79 | 0.93 | 0.93 | 0.54 | 0.93 | **0.99** |
| pghero | 0.68 | 0.78 | 0.78 | 0.49 | 0.78 | **0.87** | 0.47 | 0.59 | 0.63 | 0.06 | 0.62 | **0.75** | 0.72 | 0.80 | 0.79 | 0.75 | 0.82 | **0.90** |
| ransack | 0.70 | 0.79 | **0.84** | 0.74 | 0.82 | **0.84** | 0.45 | 0.61 | 0.62 | 0.44 | 0.63 | **0.69** | 0.71 | 0.88 | 0.86 | 0.69 | 0.87 | **0.90** |
| SAX | 0.85 | 0.92 | 0.93 | 0.50 | 0.91 | **0.96** | 0.70 | 0.89 | 0.87 | 0.00 | 0.83 | **0.93** | 0.86 | 0.95 | 0.94 | 0.80 | 0.95 | **0.97** |
| searchkick | 0.73 | 0.85 | 0.85 | 0.68 | **0.88** | 0.87 | 0.58 | 0.75 | 0.74 | 0.50 | **0.78** | 0.77 | 0.70 | 0.85 | 0.87 | 0.56 | 0.88 | **0.88** |
| SemanticMediaWiki | 0.83 | **0.91** | **0.91** | 0.84 | **0.91** | 0.90 | 0.58 | **0.75** | 0.73 | 0.54 | 0.73 | **0.75** | 0.83 | 0.91 | 0.90 | 0.78 | 0.90 | **0.92** |
| solr-iso639-filter | 0.69 | 0.76 | 0.74 | 0.63 | 0.77 | **0.85** | 0.61 | 0.69 | 0.64 | 0.57 | 0.70 | **0.82** | 0.67 | 0.77 | 0.72 | 0.59 | 0.76 | **0.86** |
| steve | 0.66 | 0.66 | 0.79 | 0.71 | 0.82 | **0.90** | 0.23 | 0.31 | 0.44 | 0.25 | 0.44 | **0.74** | 0.83 | 0.89 | 0.87 | 0.60 | 0.88 | **0.96** |
| **Median** | 0.72 | 0.85 | 0.85 | 0.64 | 0.88 | **0.90** | 0.58 | 0.75 | 0.74 | 0.44 | 0.78 | **0.82** | 0.74 | 0.89 | 0.87 | 0.69 | 0.88 | **0.92** |
| **Average** | 0.74 | 0.83 | 0.85 | 0.63 | 0.86 | **0.91** | 0.58 | 0.73 | 0.74 | 0.35 | 0.76 | **0.85** | 0.76 | 0.87 | 0.87 | 0.68 | 0.88 | **0.93** |

TABLE 7: Statistical tests' results of SPEA-2 compared to ML techniques under cross-validation.

| | | vs. DT | vs. RF | vs. LR | vs. NB | vs. SVC |
|---|---|---|---|---|---|---|
| **AUC** | *p-value* | $> 10^{-16}$ | $> 10^{-16}$ | $> 10^{-16}$ | $> 10^{-16}$ | $> 10^{-16}$ |
| | *A estimate* | 0.87 | 0.73 | 0.69 | 0.96 | 0.84 |
| | *Magnitude* | L | M | M | L | L |
| **F1** | *p-value* | $> 10^{-16}$ | $> 10^{-16}$ | $> 10^{-16}$ | $> 10^{-16}$ | $> 10^{-16}$ |
| | *A estimate* | 0.85 | 0.69 | 0.72 | 0.94 | 0.82 |
| | *Magnitude* | L | M | M | L | M |
| **Accuracy** | *p-value* | $> 10^{-16}$ | $> 10^{-16}$ | $> 10^{-16}$ | $> 10^{-16}$ | $> 10^{-16}$ |
| | *A estimate* | 0.88 | 0.68 | 0.73 | 0.95 | 0.85 |
| | *Magnitude* | L | M | L | L | M |

L: Large, M: Medium, S: Small, N: Negligible

Area Under the ROC Curve (AUC), F1-score, and accuracy, to measure the performance of our classifier. Table 8 presents the effectiveness of cross-project modeling compared to ML techniques while Table 9 reports the statistical tests' results. Note that we do not include the deterministic ML algorithms (*i.e.*, LR, NB and SVC) for the statistical tests' comparisons as we only recorded nine observations for each of them, under the cross-project scenario.

First, the results show that SPEA-2 achieves high scores of AUC with a median of 82% while the values range from 68-97%. We observe that 10 out of 15 projects showed high performance results ($\geq$ 80%). In particular, `contextlogger` achieves a significant AUC score of 97%. Additionally, we observe an improvement of 5% in median over ML techniques whose results have also decreased compared to their within-project scores. Moreover, the statistical tests' results show that the difference is significant with *large* effect sizes compared to RF and DT.

The same observations for AUC are also applied to F1-score for which we see that SPEA-2 is the best technique 14 out of 15 projects by achieving 66% on median compared to ML techniques that showed moderate F1-scores of 58%, 55% 52%, 37% and 33% for LR, SVC, RF, NB and DT respectively. The statistical analysis confirms the significant difference (with RF and DT) with large effect sizes, as reported in Table 9.

Looking at the accuracy results, we see that the scores are significantly improved compared to ML results, for 13 projects out of 15 by achieving in median 88% (and 86% on average) and the accuracy values range from 68-98%. Similarly to within validation, SPEA-2 obtained better accuracy results compared to ML with significant differences compared to DT and RF and large effect sizes.

However, compared to the within-project validation (RQ2), the results indicate that our approach can achieve a less significant performance with a *large* effect size, which may be explained by the fact under cross-project, the target project may have a low collinearity with the source project metrics thresholds. Overall, SKIPCI still be a very promising solution to mitigate the lack of data, especially for new software projects having no enough history.

**Summary for RQ3.** *Under cross-project scenario, our* SKIPCI *still outperforms state-of-the-art ML techniques with a significant improvement by achieving an average of 82%, 69% and 86% in terms of AUC, F1 and accuracy, respectively in identifying CI skip commits. While cross-project results are lower than the within-project results (RQ2), our approach is still a promising solution that can be practically used when little/no data is available for new projects.*

## 5.4 Results for Features' Influence (RQ4)

In the following, we report the results of features analysis within and cross-project validations.

### 5.4.1 Within-project results

Figure 7 depicts the results of feature categories' occurrences for each project while Table 10 summarizes the ranking of the most occurring features among all the studied projects considering the cross-validation scenario.

Broadly speaking, the results reveal a slight variation between features' categories in terms of their occurrences in the built SPEA-2 models. Among all projects, the most important features are those related to the *link to last commits* and *commit purpose* that appear each one the most in at least 8 out of 15 projects.

**Link to Last Commits** has shown to be a strong indicator of the commit likelihood to be skipped. For example, from Table Table 10, we observe that the metric *proj_recent_skip*,

TABLE 8: Performance of SPEA-2 vs. ML techniques under cross-project validation.

| Project | AUC | | | | | | F1 | | | | | | Accuracy | | | | | |
|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|---|
| | *DT* | *RF* | *LR* | *NB* | *SVC* | *SPEA-2* | *DT* | *RF* | *LR* | *NB* | *SVC* | *SPEA-2* | *DT* | *RF* | *LR* | *NB* | *SVC* | *SPEA-2* |
| candybar-library | 0.62 | 0.71 | 0.75 | 0.69 | 0.62 | **0.77** | 0.69 | 0.79 | 0.76 | 0.84 | 0.62 | **0.87** | 0.61 | 0.72 | 0.71 | 0.65 | 0.74 | **0.80** |
| contextlogger | 0.39 | 0.74 | 0.94 | 0.51 | 0.61 | **0.97** | 0.38 | 0.62 | 0.91 | 0.52 | 0.61 | **0.96** | 0.24 | 0.63 | 0.94 | 0.70 | 0.45 | **0.98** |
| future | 0.39 | 0.37 | 0.47 | 0.47 | 0.47 | **0.84** | 0.27 | 0.25 | 0.25 | 0.34 | 0.47 | **0.66** | 0.28 | 0.26 | 0.55 | 0.73 | 0.25 | **0.84** |
| GI | 0.52 | 0.66 | **0.86** | 0.70 | 0.54 | **0.86** | 0.19 | 0.25 | 0.48 | 0.20 | **0.54** | 0.52 | 0.14 | 0.39 | 0.79 | 0.50 | 0.18 | **0.88** |
| grammarviz2 | 0.39 | 0.69 | 0.87 | 0.71 | 0.55 | **0.88** | 0.18 | 0.34 | 0.65 | 0.26 | 0.55 | **0.78** | 0.21 | 0.48 | 0.87 | 0.58 | 0.24 | **0.95** |
| groupdate | 0.52 | 0.73 | 0.77 | 0.71 | 0.58 | **0.81** | 0.33 | 0.53 | 0.56 | 0.37 | 0.58 | **0.64** | 0.39 | 0.73 | 0.74 | 0.75 | 0.37 | **0.83** |
| mtsar | 0.59 | 0.61 | 0.65 | 0.59 | 0.58 | **0.68** | 0.51 | 0.52 | 0.53 | 0.53 | **0.58** | **0.58** | 0.54 | 0.60 | **0.69** | 0.68 | 0.47 | 0.68 |
| parallec | 0.61 | 0.87 | 0.89 | 0.85 | 0.67 | **0.92** | 0.67 | 0.88 | 0.89 | 0.75 | 0.67 | **0.92** | 0.61 | 0.87 | 0.89 | 0.85 | 0.67 | **0.92** |
| pghero | 0.49 | 0.72 | 0.76 | 0.64 | 0.55 | **0.80** | 0.32 | 0.54 | 0.58 | 0.38 | 0.55 | **0.63** | 0.42 | 0.75 | 0.77 | 0.53 | 0.33 | **0.82** |
| ransack | 0.52 | 0.73 | 0.72 | 0.61 | 0.52 | **0.77** | 0.26 | 0.42 | 0.48 | 0.27 | 0.52 | **0.58** | 0.42 | 0.65 | 0.81 | 0.82 | 0.24 | **0.88** |
| SAX | 0.46 | 0.67 | **0.86** | 0.74 | 0.57 | 0.85 | 0.30 | 0.43 | 0.69 | 0.37 | 0.57 | **0.70** | 0.23 | 0.50 | 0.83 | 0.61 | 0.33 | **0.88** |
| searchkick | 0.67 | 0.71 | 0.76 | 0.52 | 0.53 | **0.82** | 0.50 | 0.54 | 0.60 | 0.40 | 0.53 | **0.70** | 0.67 | 0.70 | 0.75 | 0.73 | 0.32 | **0.86** |
| SemanticMediaWiki | 0.70 | 0.79 | 0.80 | 0.61 | 0.65 | **0.85** | 0.39 | 0.46 | 0.54 | 0.31 | 0.65 | **0.62** | 0.69 | 0.71 | 0.81 | 0.78 | 0.40 | **0.90** |
| solr-iso639-filter | 0.65 | 0.70 | **0.77** | 0.69 | 0.52 | 0.74 | 0.65 | 0.68 | **0.71** | 0.58 | 0.52 | **0.71** | 0.61 | 0.66 | **0.80** | 0.66 | 0.43 | 0.78 |
| steve | 0.43 | 0.52 | 0.73 | 0.62 | 0.51 | **0.77** | 0.13 | 0.16 | 0.42 | 0.15 | 0.51 | **0.52** | 0.20 | 0.44 | 0.87 | 0.77 | 0.16 | **0.93** |
| **Median** | 0.52 | 0.71 | 0.77 | 0.64 | 0.55 | **0.82** | 0.33 | 0.52 | 0.58 | 0.37 | 0.55 | **0.66** | 0.42 | 0.65 | 0.80 | 0.70 | 0.33 | **0.88** |
| **Average** | 0.53 | 0.68 | 0.77 | 0.64 | 0.56 | **0.82** | 0.38 | 0.49 | 0.60 | 0.42 | 0.56 | **0.69** | 0.42 | 0.61 | 0.79 | 0.69 | 0.37 | **0.86** |

TABLE 9: Statistical tests' results of SPEA-2 under cross-projects compared to its achieved within-project results, RF and DT (*).

| | | vs. Cross-Validation | vs. DT | vs. RF |
|---|---|---|---|---|
| **AUC** | *p-value* | $> 10^{-16}$ | $> 10^{-16}$ | $> 10^{-16}$ |
| | *A estimate* | 0.19 | 0.98 | 0.83 |
| | *Magnitude* | L | L | L |
| **F1** | *p-value* | $> 10^{-16}$ | $> 10^{-16}$ | $> 10^{-16}$ |
| | *A estimate* | 0.22 | 0.83 | 0.77 |
| | *Magnitude* | L | L | L |
| **Accuracy** | *p-value* | $> 10^{-16}$ | $> 10^{-16}$ | $> 10^{-16}$ |
| | *A estimate* | 0.2 | 0.98 | 0.88 |
| | *Magnitude* | L | L | L |

L: Large, M: Medium, S: Small, N: Negligible
(*) LR, NB and SVC were executed once for each experiment instance since they are deterministic techniques. Hence we cannot compare the statistical differences with them.

TABLE 10: Top-Features Analysis for all projects (within-project scenario).

| Category | TOP-1 | # of projects |
|---|---|---|
| *Commit Purpose* | is_doc | 1 |
| | is_merge | 1 |
| | is_build | 1 |
| | classif | 1 |
| | is_fix | 1 |
| *Link to last commits* | proj_recent_skip | 2 |
| | comm_recent_skip | 1 |
| | AGE | 1 |
| | EXP | 1 |
| | LT | 1 |
| *Statistics of current commit* | CM | 2 |
| | LA | 1 |
| | LD | 1 |

*i.e.*, number of recently skipped commits in the project, is the appearing feature in two projects including `future` and `steve`. A closer examination reveals that this feature has a clear indication of whether a commit should be skipped or not. For example, in `steve` project, our tool suggests that to a label a commit as skipped, it should have at least 3 recently skipped commits. This condition covers alone 92% of the commits in this project. A similar observation can be applied in `future` project, in which we also observed that having at least 2 recently commits alone would detect the skipped commits in this project with an accuracy of 80%. Similarly to *proj_recent_skip* feature, *comm_recent_skip* also seems to be relevant and appears the most in `SAX` in which 69% of the skipped commits are preceded by at least one skipped change that was skipped by the same committer; which correlates favorably with our earlier observation about *proj_recent_skip*. These results provide evidence that, similar to build failures [31], commits are skipped in sequence. *AGE* is also a prominent feature in `parallec` in which we found that 61% of the skipped commits have on average 0.69 elapsed days between the current and the last time the changed file were modified, which means that many commits can be skipped in the same day. Thus, it is clear that developers tend to skip commits consecutively maybe because simple changes are generally performed during a specific period of the development *e.g.* after a release. Committer experience *i.e. EXP* appears the most in one project indicating that the experienced developers are probably more familiar with CI features. A similar observation is applied to *LT*. Abdalkareem et al. [7] have also found the committer experience could help to accurately detect skipped commits.

**Commit Purpose** features have a good probability to indicate whether this commit should be skipped as they appear the most in 5 out of 15 projects. For instance, in `solr-iso639-filter` project, the condition $is\_doc = 1$ can detect alone the skipped commits with 62% of accuracy, and in `steve` project 100% of the merged commits are not skipped, which is consistent with the real world situation. In addition, other features indicating the commit purpose, *e.g.* *is_build*, seem to be important. Abdalkareem et al. [7] also pointed out that these features (*i.e.* CI skip rules as mentioned in the paper) can be strong indicators for CI skip detection.

**Statistics about the current commit** are also important in detecting CI skipped commits. First, terms appearing in the commit message can be useful as *CM* is the most occurring feature in two projects including `contextlogger` and `grammarviz2` projects. For example, in `contextlogger` 90% of skipped commit messages contain "Update Readme.md". This finding was previously confirmed by Abdalkareem et al. [7]. In addition, features providing statistics about the current commit change size can also be useful to detect skipped commits. For example, in the project `searchkick` for which the number of deleted lines $LD$ is the most important feature, 89% of skipped commits have $LD \leq 10$ which indicates that small changes are more likely to be skipped. The same observation is applied to the number of added lines *i.e. LA*.

#### 5.4.2 Cross-project results

The top-features analysis under cross-project scenario is presented in Table 11. This table clearly indicates that statistics linked to last commits are the most important features across the studied projects. Specifically, metrics denoting the number of recently skipped commits from the current committer are dominant in 9 out of the 15 studied projects, while recent skipped commits from all developers in the project are dominant in 5 out of the 15 projects. This strengthens our previous findings claiming that if the commit is skipped, the next commit is more likely to be skipped as well. The recent experience of the developer `REXP` appear also on the top list of one project which indicates that the more familiar is the developer with the source code, the more he is able to recognize the type of changes that can be skipped.

Another important observation to consider is that none of the features related to commit purpose category is present in the top-features list which is may be explained by the fact developers do not usually skip even simple changes *i.e.* CI skip option is miss-used. Looking at the data, we have found a non-neglected number of cases where developers do not skip commits that are worth to be skipped (*e.g.* documentation commits). Additionally, the statistics of the current commit is also not present in the top-features list, which indicates that these metrics are less likely to be generalized, as CI skip commits depend mainly on the specific context of the project (*e.g.* the day of the week when developers usually skip commits can differ from a project to another).

TABLE 11: Top-Features Analysis for all projects (cross-project validation).

| Category | TOP-1 | # of projects |
|---|---|---|
| ***Link to last commits*** | comm_recent_skip | 9 |
| | proj_recent_skip | 5 |
| | REXP | 1 |

**Summary for RQ4.** *Within-project validation, the feature analysis reveals that (1) the number of previously skipped commits, (2) features indicating the commit purpose and (3) terms appearing in the commit message are prominent features in CI skip detection. When it comes to the cross-project rules, the results show that there is a strong link between current and previous commits results.*

## 6 Industrial Case Study

While in RQ1-RQ3, we have shown the efficiency of SkipCI to detect CI skipped commits, we aim in this section to assess the applicability of our approach in practice *i.e.* from developers' perspectives. We first present the case study design then we report the obtained results.

### 6.1 Case Study Design

We designed our industrial case study to address the following research question:

**RQ5. (Industrial validation).** *Are the CI skip commit recommendations provided by* SkipCI *useful for CI developers in practice?*

#### 6.1.1 Case selection

We conducted a user study with our industrial partner, a large company producing digital document products, services and printers. We evaluate SkipCI during a period of two weeks, *i.e.*, 10 working days on two large and long-lived software systems developed by our industrial partner. We denote both projects as *Project-1* and *Project-2* in this paper for confidentiality reasons. Both projects use a proprietary customized CI system that supports high configurability. Developers working on both projects are used to integrate their code changes more than once per day. As case study participants, we reached out 36 active developers working on both projects inviting them to participate in our experiments. We received positive responses from 14 developers who volunteered to participate, where 8 developers are from project-1, and 6 developers are from project-2.

Participants were first asked to fill out a pre-study questionnaire that consists of seven control questions. The questionnaire helped us to collect background and demographic information such as their role within the company and within the project, their experience with the studied project, their academic degree, their proficiency in CI practice, their familiarity with the CI skip commit feature, and their experience with software quality assurance. The list of questionnaires and the obtained results can be found in our online replication package. All the participants had a minimum of 6 years experience post-graduation and were working as active programmers with strong backgrounds in CI practices, and software quality assurance. All participants are familiar with the studied projects (71% have over 3 years, and 29% have over 2 years experience with the concerned projects). Moreover, all of our participants hold an academic degree related to computer science and/or software engineering (50% Bachelor, 35.7% Masters, 14.3% Ph.D.).

#### 6.1.2 Study setup and analysis method

As a first step to prepare and integrate the SkipCI bot into the CI pipeline, we collected data about the history of commits and builds for both projects, from the last 3 years from the version control system and the CI tool. Such data allowed us to generate the set of required features to train our model to generate the CI skip detection rules. Thereafter, we deployed SkipCI and integrated it into the CI service pipeline for both projects in the form of a Bot. The SkipCI bot is triggered whenever a new commit is pushed and shows a pop-up notification message recommending whether the current commit could be (1) skipped or (2) not skipped. To keep track of the developers' decisions for our evaluation, we integrated a routine in our SkipCI bot to record the commit ID, the recommended action, the developer decision, and her/his comments (if any).

With the sake of comparing the results of our tool with a baseline approach, and better understanding the participants behavior, we also considered a random recommender tool (that we refer to as RANDOM in this paper) that generates CI skip commits at a random way. Then, for each project, we split the concerned participants into two groups (A and B) of equal sizes (*i.e.*, 7 developers each), in such a way that each group will use one of the tools, SKIPCI or RANDOM. Hence, in total, we had four groups (2 groups per project) as shown in Table 12.

TABLE 12: Participants partition statistics.

| Project | Tool | Group | # of developers | # of commits |
|---|---|---|---|---|
| Project-1 | SKIPCI | A1 | 4 | 61 |
| | RANDOM | B1 | 4 | 52 |
| Project-2 | SKIPCI | A2 | 3 | 43 |
| | RANDOM | B2 | 3 | 48 |

During the study period, for each commit, the developer receives a recommendation from either SKIPCI or RANDOM (depending on her/his group) indicating to skip the commit or not. The developer can either "*accept*" or "*decline*" the skip recommendations. Moreover, we configured both tools to show a pop-up notification asking the developer to optionally leave her/his justification about his accept/decline decision, immediately after she/he makes a decision. To avoid potential biases in our experiments, the individual developers were not aware of the specific tools being compared (*i.e.*, SKIPCI and RANDOM).

In total, the 14 participants performed 204 commits (113 for Project-1 and 91 commits for Project-2) during the study period of 10 business days. The number of performed commits by each group for each project are reported in Table 12. After the study period, we collected the experiments records of the accepted and rejected recommendations for each developer for both tools. To analyze the collected data and answer *RQ5*, we compute for each project, and each group, the ratio of both *accepted* and *rejected* recommendations by the developers, with respect to the total number of recommendations provided from each tool.

Moreover, to further understand how to improve SKIPCI, we performed an online interview with four developers from both groups A1 and A2 who assessed the SKIPCI bot (2 developers from each project). The interview consists of a structured discussion guided by three main questions:

- **Q1:** *How important is the CI Skip option in CI practice?*
- **Q2:** *How efficient is our CI skip recommendation bot in the context of your project?*
- **Q3:** *What additional features or improvements do you recommend to further improve* SKIPCI*?*

In the next subsection, we present and discuss the obtained results for our user case study.

## 6.2 Case study results and discussion

Table 13 summarizes the results of deploying SKIPCI during 10 working days with our industrial partner on both projects. We observe that developers accepted most of the "Skip" recommendations provided by SKIPCI in both projects (*i.e.*, groups A1 and A2), with 88.9% and 90.9% in project-1 and project-2, respectively. We also observe that SKIPCI recommended to skip 18 out of the 61 commits from project 1 (29.5%), and 11 out of the 48 commits from project-2 (25.6%). On the other side, developers tend to accept only a small number of "Skip" recommendations provided by RANDOM, with 17.9% and 13.6% of the total skip recommendations from project-1 and project-2, respectively.

As for the "Non-skip" recommendations, we also observe SKIPCI recommended not to skip 43 out of 61 commits (70.4%) for project-1, and not to skip 32 out 43 commits (74.4%) for project-2, that were accepted by developers with over 90% for both projects. Looking at the RANDOM recommendations, we found that developers accepted 66.7% and 65.4% of non-skip recommendations from project-1 and project-2, respectively. While the results of random recommendations may seem quite high, their results could be justified by the fact that developers tend to generally run the build after each commit in the context of CI.

By looking at the comments left by the SKIPCI participants when justifying their decisions, developers of both projects highlighted in their comments that they found the CI skip commit relevant because it can save time for trivial changes that do not need to build the entire system. For instance, one developer wrote in a comment in response to our recommendation:

💬 *"I agree! So I actually do not see benefit of starting the CI build immediately, there are only non-code changes were made in my last commit."*

Another developer mentioned in his comment:

💬 *"[...] of course this commit and my previous one should be skipped, my changes are only on markdown and JSON files. It's bad for this commit to wastefully use server time."*

Furthermore, another developer who declined a CI skip recommendation from SKIPCI commented:

💬 *"I am fine with skipping this commit, but this time I want to merge my changes to the master branch. My changes can only be merged when the CI build has successfully run [...] this will never happen if I skip the CI build"*.

In another declined CI skip recommendation, the developer left the following comment:

💬 *"Well I don't agree, even very few lines have been changed related to my function call redirection and my variable rename changes, I am inclined to run the build to be on the safe side anyway."*

Furthermore, to gain insights and better understand how to improve SKIPCI, we interviewed four developers (2 developers from each project) who assessed SKIPCI within the two-week study period, as described in Section 6.1.2. The four developers recognized and highly value the importance of considering the CI skip feature provided by the CI systems given the waste of time of resources for unnecessary builds. The four developers were also very positive on the relevance of our SKIPCI bot in the context of their projects. In response to possible improvement of SKIPCI, one of the interviewees indicated that:

💬 *"[...] I found the bot very useful to me, basically what I used to do is to skip the entire build pipeline if certain condition is met based on my "safe list". I manually defined a basic, yet*

TABLE 13: Case study results.

| Project | Group | Tool | Total commits | Recommendation | # commits | Results* | |
|---|---|---|---|---|---|---|---|
| Project-1 | A1 | SKIPCI | 61 | Skip | 18 | 11.1% 🟥 | 🟦 88.9% |
| | | | | Non-skip | 43 | 9.3% 🟥 | 🟦 90.7% |
| | B1 | RANDOM | 52 | Skip | 28 | 82.1% 🟥 | 🟦 17.9% |
| | | | | Non-skip | 24 | 33.3% 🟥 | 🟦 66.7% |
| Project-2 | A2 | SKIPCI | 43 | Skip | 11 | 9.1% 🟥 | 🟦 90.9% |
| | | | | Non-skip | 32 | 9.4% 🟥 | 🟦 90.6% |
| | B2 | RANDOM | 48 | Skip | 22 | 86.4% 🟥 | 🟦 13.6% |
| | | | | Non-skip | 26 | 34.6% 🟥 | 🟦 65.4% |

* 🟦 *Accepted*, 🟥 *Rejected*

*simple, script to check my conditions like "if the changed files are in a given directory, then trigger the CI build, else skip it". However, I have to manually go through the change diff to look at each individual file [...] and I often ignore or change my defined conditions. I wish to have more control over the proposed CI skip recommendation tool, based on what I'm doing and based on how the project evolves."*

Two other developers pointed out the importance of providing more details by SKIPCI to justify the recommended decisions along with a summary of the changes in either a textual manner or in a user-friendly visualization to help them taking the right decision while giving more trust and transparency to our tool. Another developer also recommended to add a user-friendly configuration layer on top of the tool that allows the developer to customize the current conditions and add her/his own conditions to the tool.

Overall, the outcomes of this survey are aligned with the motivations of this paper advocating for using interpretable rule-based recommendations based on various features that could be learned from the CI build and project history considering both (1) skipped and (2) non-skipped CI commits (*i.e.*, the minority and majority classes).

From this user study, we learn that to improve SKIPCI, we have to include three aspects. First, we need to deploy along with the tool, a set of interpretable and configurable rules so that developers can understand and customize the tool based on their preferences/experience on what should or should not be skipped. Second, the tool needs to provide more details along with the recommended skip decisions to justify whether the commit should be skipped or not. Third, the rules should be re-trained and updated regularly as the project evolves through learning from the recent decisions taken by the developers. Moreover, based on our interactions with developers in this industrial case study, we advocate that the CI skip feature should be used *wisely* and with *caution*. The developer needs to make the necessary verification based on her/his current code changes in the commit, before taking the decision to skip or not. This is indeed an important aspect, as it remains to some extent hard to understand the semantics of source code changes.

## 7 IMPLICATIONS

In this section, we discuss the implications of our findings.

### 7.1 For CI developers

**Developers can efficiently identify their CI skipped commits.** The usefulness of our SKIPCI approach has been shown through its achieved results within and cross-project validations as well as our industrial case study. Nevertheless, we believe that the key innovation of our approach is its ability to provide the user with a comprehensible justification for the detection of CI skip commits especially when the changes made in the commit are non-trivial. For instance, Figure 8 illustrates an example of a detection rule generated by SKIPCI to detect CI skipped commits in the `Semantic MediaWiki`[6] project with a high AUC score of 89%. Using this rule, we can detect the CI commit described early in Section 2 as skipped since its characteristics satisfy the conditions of the rule, *e.g.*, having a number of changed files NF $\leq$ 215 (*i.e.*, NF which is equal to 2 in the example). Moreover, it is worth noting that, thanks to the flexibility of MOGP techniques, it may be possible to reduce the complexity of the generated detection rules (*e.g.*, tree size and/or depth) in order to generate more comprehensible justification by considering this objective in the fitness function (or as a constraint in the solution encoding), but at the cost of scarifying the accuracy as these objectives are in conflict [28].

**Usage scenario of our tool.** Figure 9 provides a typical usage scenario of SKIPCI in practice. SKIPCI is triggered when a developer commits a change to the repository (1). At this point, the tool check whether the project contains enough data by verifying whether the rate of CI skipped commits $\geq 10\%$. Based on this condition, SKIPCI provides the options to apply:

- *Within-project prediction (2)*: In this case, the user is invited to choose whether to generate new rule (2.1) or to load the previously generated rule (2.2). After the generation/loading of the detection rule, our tool analyzes the changes made in the commit files to determine whether the commit can be skipped or not (2.3).
- *Cross-project prediction (3)*: When the project does not have enough training data (*e.g.*, small/new project or miss-use of CI skip option), this does not prevent it from using our tool. Indeed, SKIPCI generates, based on our dataset, a set of optimal detection rules that can be loaded (3.1) to predict the current CI commit outcome (3.2). Then, all the outputs of these rules are combined

6. https://github.com/SemanticMediaWiki/SemanticMediaWiki

to a single detection model using majority voting (3.3). If the majority of rules predict the commit as skipped, then it is predicted as skipped and the best rule will be displayed for the justification.

Finally, SKIPCI provides an explained recommendation to the user (4) in order to guide him in his decision to (not) skip the committed change.

### 7.2 For researchers

**Can the predictive performance be improved with the use of SMOTE?** So far, we showed that SKIPCI provides an effective improvement over the state-of-the-art without rebalancing. However, one can argue that the use of resampling can further improve the identification of CI skipped commits of SKIPCI, even though the latter has shown less sensitivity to the class imbalance problem as pointed out in prior research [17, 18, 71, 72]. Thus, we conduct a set of additional experiments to rebalance the input data prepared in Section 4.2(10 cross-validation) using SMOTE and re-run the MOGP learning process on the new balanced data. We use the same approach described in Section 3 to generate our detection rules. To provide a comprehensive comparison, we compute the F1-score, AUC score and the overall accuracy. Figure 10 shows the obtained results with (in red) and without (in blue) using SMOTE.

We observe that by using SMOTE, SKIPCI can achieve in median 91%, 86% and 95% in terms of AUC, F1 and accuracy respectively, which represents an improvement of 1%, 4% and 3%, respectively but with no statistically significant differences. This suggests that using an external approach to artificially rebalance the dataset can slightly improve the classification performance of SKIPCI. However, the sampling techniques have their own drawbacks. In fact, sampling can lead to over-fitting and affect the interpretability of the generated rules [19] as the original and the balanced training corpora have different characteristics. Furthermore, rebalancing can also be computationally expensive [72]. For these reasons, in this paper, we advocate that using MOGP alone is a better classification strategy when the data is imbalanced.

**Researchers can investigate periodicity in CI skip.** Our features analysis results (RQ4) reveal that many CI skipped commits occurred consecutively and features related to historical statistics about the commits are strong indicators of the current commit outcome. Hence, we argue that it is important to track skipped commits in future research. We also encourage researchers to investigate what software engineering activities may link with such skip periods, *e.g.*, automated refactoring etc.

### 7.3 For tool builders

**Further tool support in CI.** Our industrial case study reveals the importance of the SKIPCI tool from developers' perspectives. As discussed in Section 6.2, one of the developers indicates that he is using his own simplified defined conditions to help him decide whether to run or skip the build for a given commit. This recalls the importance of providing efficient, lightweight and practical tool support to further improve the CI pipeline. Indeed, the current CI practice is still in its infancy, as CI technologies are experiencing an exponential growth in both open-source and commercial projects. Further support from tool builders is needed to adequately respond to the developers' needs and cut with the expenses of CI build in modern software engineering.

## 8 RELATED WORK

### 8.1 Detection of CI skipped commits

Abdalkareem et al. [6] are the first to examine the CI commits that can be skipped and determine the reasons behind it. Additionally, they proposed a rule-based technique to automatically detect and label the commits to be skipped by using 8 rules. Based on a corpus of ten open-source Java projects that use Travis CI, this technique has reached a moderate score of 58% in terms of F1. Another major problem to highlight is that this approach is based solely on Java projects. In this paper, we extend the metrics used in this work while considering two other programming languages namely Ruby and PHP.

A related effort for improving CI skip detection, by Abdalkareem et al. [7], proposed a ML-based technique to raise the issue related to the moderate performance of the rule-based approach. By conducting an empirical study based on the same dataset of [6], the used DT classifier achieved higher F1 of 79% within project validation and 55% under cross-project validation. Additionally, they investigated the most prominent metrics and found that the number of developers who changed the modified files and the terms used in the written commit messages are the best indicators of CI skip commits. In this paper, we have shown that using MOGP techniques can further enhance the predictive performance without sacrificing the interpretability of the rules by applying resampling.

### 8.2 Approaches to speed up CI build process

There is a considerable amount of research efforts that attempted to reduce the expense of CI process by detecting CI build failure. Hassan and Wang [32] leveraged the history of Ant, Maven and Gradle build systems in order to train Random Forest models. Their approach achieved over 90% in terms of AUC on average. Xia and Li [39] employed and compared nine ML classifiers to train CI build failure prediction models. Another work by Jin and Servant [73] proposed a novel technique that detects the CI build failure by separating the first failure from the remaining sequence of failures which has been shown its effectiveness compared to other existing solutions.

Recently, Saidani et al. [28, 74] proposed a MOGP technique to predict CI build failures by adapting the NSGA-II algorithm [47]. The proposed approach achieved a significant improvement of 8% in terms of AUC over ML techniques in a highly imbalanced dataset. Most of these existing works have found that metrics related to past build outcome can be effective to predict the current one. However, these approaches do not detect the existence of CI commits that are not necessary to trigger the build process, and therefore, these approaches could be further improved, when augmented with efficient CI skip commits detection techniques. Therefore, in this paper, SKIPCI aims at taking one step forward to speed up the CI build process and reduce its costs by identifying skipped commits in CI. It is worth noting that SKIPCI could be used jointly with existing CI build failure prediction techniques [28, 39, 73] to provide better support to CI developers while reducing CI build costs and enhancing developer's experience in CI environments.

## 9 THREATS TO VALIDITY

**Threats to internal validity** are concerned with the factors that could have affected the validity of our results. The main concern could be related to the stochastic nature of search-based algorithms, RF and DT. To address this issue, we repeated each experimentation 31 times and considered the median scores values used to evaluate the predictive performance. Threats to internal validity could also be related to our industrial case study that aimed at deploying and evaluating our SKIPCI approach in practice. The opinions/decisions of the practitioners involved in our study may be divergent or influenced by the used tool when it comes to accepting or rejecting a CI skip commit recommendation, which can impact our results. To mitigate this threat, we compared the results of our tool to a baseline tool with random recommendations. Furthermore, our invited participants achieved a higher response rate of 38.9% (14 out of 36 invited developers) than the average rate in software engineering research surveys [75].

**Threats to construct validity** are mainly related to the rigor of the study design. First, we relied on three standard performance metrics namely F1-measure, AUC and accuracy that are widely employed in predictive models comparison [76] and also considered three performance measures, *i.e.*, hyper-volume (HV), generational distance (GD) and spacing (SP) in order to compare multi-objective algorithms from SBSE perspective. Second, although we used different search-based and ML algorithms, there exist other techniques. As a future work, we plan to extend our empirical study with other baseline techniques. Third, a threat to construct validity could be related to the annotated set of skipped commits used in our dataset, as we cannot exclude that developers of our studied projects could have missed some skipped commits, or even included some false positives. Another threat to construct validity could be related to parameters' tuning as setting different parameters can lead to different results for search-based as well as ML techniques. We mitigated this issue by applying several trial and error iterations to tune search-based algorithms and relied on Grid Search [70] method to find the optimal settings of ML techniques.

**Threats to external validity** are concerned with the generalizability of results since the experiments were based on fifteen open-source projects that use Travis CI, and two projects from our industrial partner. However, it is worth to recall that CI skip option is generally miss-used in practice and hence labeled training data is not always available. Additionally, while in our approach we focus on Travis CI, the most popular CI system [61], our approach can be applied to other CI systems. Future replications of this study are necessary to confirm our findings.

## 10 CONCLUSIONS AND FUTURE WORK

This paper proposed a novel search-based approach for CI Skip detection, SKIPCI, in which we adapted SPEA-2 to generate optimal detection rules with a tree-like representation in order to find the best trade-off between two conflicting objective functions to (1) maximize the true positive rate, and (2) minimize the false positive rate.

An empirical study conducted on a benchmark of 16,334 Travis CI commits of fifteen projects that use Travis CI shows that SKIPCI outperforms Random Search, mono-objective Genetic Algorithm and three other multi-objective algorithms which indicates that our adaptation is more suited than other search-based techniques. Considering two validation scenarios namely cross-validation and cross-project validation, the statistical analysis of the obtained results reveals that SKIPCI is advantageous over five Machine Learning (ML) techniques confirming that our formulation is better to solve the problem. Moreover, our experience with the industrial partner demonstrates the effectiveness of SKIPCI in providing relevant recommendations to developers from two different projects. Regarding the features analysis, we found that (1) the number of previously skipped commits, (2) features indicating the commit purpose and (3) terms appearing in the commit message are the prominent in CI skip detection.

Our future research agenda includes performing a larger empirical study with other open-source projects while considering other metrics, *e.g.* semantic similarity, to improve the detection accuracy.

## ACKNOWLEDGMENTS

A special thanks to the engineers who participated in this study for their time and feedback. This research has been partially funded by the Natural Sciences and Engineering Research Council of Canada (NSERC).

## REFERENCES

[1] Paul M Duvall, Steve Matyas, and Andrew Glover. *Continuous integration: improving software quality and reducing risk*. Pearson Education, 2007.

[2] Bogdan Vasilescu, Yue Yu, Huaimin Wang, Premkumar Devanbu, and Vladimir Filkov. Quality and productivity outcomes relating to continuous integration in github. In *10th Joint Meeting on Foundations of Software Engineering*, pages 805–816, 2015.

[3] Martin Fowler. Continuous integration. https://www.martinfowler.com/articles/continuousIntegration.html, 2006.

[4] Taher Ahmed Ghaleb, Daniel Alencar da Costa, and Ying Zou. An empirical study of the long duration of continuous integration builds. *Empirical Software Engineering*, pages 1–38, 2019.

[5] Yang Luo, Yangyang Zhao, Wanwangying Ma, and Lin Chen. What are the factors impacting build breakage? In *Web Information Systems and Applications Conference*, pages 139–142, 2017.

[6] Rabe Abdalkareem, Suhaib Mujahid, Emad Shihab, and Juergen Rilling. Which commits can be ci skipped? *IEEE Transactions on Software Engineering*, 2019.

[7] Rabe Abdalkareem, Suhaib Mujahid, and Emad Shihab. A machine learning approach to improve the detection of ci skip commits. *IEEE Transactions on Software Engineering*, 2020.

[8] M. Beller, G. Gousios, and A. Zaidman. Travistorrent: Synthesizing travis ci and github for full-stack research on continuous integration. In *IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pages 447–450, 2017.

[9] Urvesh Bhowan, Mark Johnston, and Mengjie Zhang. Evolving ensembles in multi-objective genetic programming for classification with unbalanced data. In *Annual conference on Genetic and evolutionary computation (GECCO)*, pages 1331–1338, 2011.

[10] Ruchika Malhotra and Megha Khanna. An exploratory study for software change prediction in object-oriented systems using hybridized techniques. *Automated Software Engineering*, 24(3):673–717, 2017.

[11] Mark Harman and Bryan F Jones. Search-based software engineering. *Information and soft. Technology*, 43(14):833–839, 2001.

[12] Mark Harman, S Afshin Mansouri, and Yuanyuan Zhang. Search-based software engineering: Trends, techniques and applications. *ACM Computing Surveys (CSUR)*, 45(1):11, 2012.

[13] Urvesh Bhowan, Mark Johnston, and Mengjie Zhang. Developing new fitness functions in genetic programming for classification with unbalanced data. *IEEE Transactions on Systems, Man, and Cybernetics, Part B (Cybernetics)*, 42(2):406–421, 2011.

[14] Marouane Kessentini and Ali Ouni. Detecting android smells using multi-objective genetic programming. In *International Conference on Mobile Software Engineering and Systems*, pages 122–132, 2017.

[15] Ali Ouni, Marouane Kessentini, Houari Sahraoui, Katsuro Inoue, and Kalyanmoy Deb. Multi-criteria code refactoring using search-based software engineering: An industrial case study. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 25 (3):23, 2016.

[16] Ruchika Malhotra, Megha Khanna, and Rajeev R Raje. On the application of search-based techniques for software engineering predictive modeling: A systematic review and future directions. *Swarm and Evolutionary Computation*, 32:85–109, 2017.

[17] Urvesh Bhowan, Mark Johnston, Mengjie Zhang, and Xin Yao. Evolving diverse ensembles using genetic programming for classification with unbalanced data. *IEEE Transactions on Evolutionary Computation*, 17(3):368–386, 2012.

[18] Urvesh Bhowan, Mark Johnston, Mengjie Zhang, and Xin Yao. Reusing genetic programming for ensemble selection in classification of unbalanced data. *IEEE Transactions on Evolutionary Computation*, 18(6):893–908, 2013.

[19] Chakkrit Tantithamthavorn, Shane McIntosh, Ahmed E Hassan, and Kenichi Matsumoto. The impact of automated parameter optimization on defect prediction models. *IEEE Transactions on Software Engineering*, 45(7):683–711, 2018.

[20] Eckart Zitzler, Marco Laumanns, and Lothar Thiele. Spea2: Improving the strength pareto evolutionary algorithm. *TIK-report*, 103, 2001.

[21] SKIPCI Dataset: https://github.com/stilab-ets/SkipCI, 2021.

[22] Jorge Novo, J Santos, and Manuel G Penedo. Multiobjective differential evolution in the optimization of topological active models. *Applied Soft Computing*, 13(6):3167–3177, 2013.

[23] Fuqing Zhao, Wenchang Lei, Weimin Ma, Yang Liu, and Chuck Zhang. An improved spea2 algorithm with adaptive selection of evolutionary operators scheme for multiobjective optimization problems. *Mathematical Problems in Engineering*, 2016, 2016.

[24] Vincent J Amuso and Jason Enslin. The strength pareto evolutionary algorithm 2 (spea2) applied to simultaneous multi-mission waveform design. In *International Waveform Diversity and Design Conference*, pages 407–417, 2007.

[25] María José Gacto, Rafael Alcalá, and Francisco Herrera. A multi-objective evolutionary algorithm for an effective tuning of fuzzy logic controllers in heating, ventilating and air conditioning systems. *Applied Intelligence*, 36(2):330–347, 2012.

[26] Sergio Garcia and Cong T Trinh. Comparison of multi-objective evolutionary algorithms to solve the modular cell design problem for novel biocatalysis. *Processes*, 7(6):361, 2019.

[27] Sokratis Sofianopoulos and George Tambouratzis. Studying the spea2 algorithm for optimising a pattern-recognition based machine translation system. In *IEEE Symposium on Computational Intelligence in Multicriteria Decision-Making (MDCM)*, pages 97–104, 2011.

[28] Islem Saidani, Ali Ouni, Moataz Chouchen, and Mohamed Wiem Mkaouer. Predicting continuous integration build failures using evolutionary search. *Information and Software Technology*, 128: 106392, 2020.

[29] Ali Ouni, Marouane Kessentini, Houari Sahraoui, and Mounir Boukadoum. Maintainability defects detection and correction: a multi-objective approach. *Automated Software Engineering*, 20(1): 47–79, 2013.

[30] Ali Ouni, Marouane Kessentini, Katsuro Inoue, and Mel O Cinnéide. Search-based web service antipatterns detection. *IEEE Transactions on Services Computing*, 10(4):603–617, 2017.

[31] Ansong Ni and Ming Li. Cost-effective build outcome prediction using cascaded classifiers. In *IEEE/ACM 14th International Conference on Mining Software Repositories (MSR)*, pages 455–458, 2017.

[32] Foyzul Hassan and Xiaoyin Wang. Change-aware build prediction model for stall avoidance in continuous integration. In *ACM/IEEE International Symposium on Empirical Software Engineering and Measurement*, pages 157–162, 2017.

[33] Zheng Xie and Ming Li. Cutting the software building efforts in continuous integration by semi-supervised online auc optimization. In *IJCAI*, pages 2875–2881, 2018.

[34] Martin Fowler. *Refactoring: improving the design of existing code*. Addison-Wesley Professional, 2018.

[35] Wei Fu and Tim Menzies. Revisiting unsupervised learning for defect prediction. In *Proceedings of the 2017 11th Joint Meeting on Foundations of Software Engineering*, pages 72–83, 2017.

[36] Yasutaka Kamei, Emad Shihab, Bram Adams, Ahmed E Hassan, Audris Mockus, Anand Sinha, and Naoyasu Ubayashi. A large-scale empirical study of just-in-time quality assurance. *IEEE Transactions on Software Engineering*, 39(6):757–773, 2012.

[37] Shane McIntosh and Yasutaka Kamei. 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, 2017.

[38] Yibiao Yang, Yuming Zhou, Jinping Liu, Yangyang Zhao, Hongmin Lu, Lei Xu, Baowen Xu, and Hareton Leung. Effort-aware just-in-time defect prediction: simple unsupervised models could be better than supervised models. In *24th ACM SIGSOFT International Symposium on Foundations of Software Engineering*, pages 157–168, 2016.

[39] Jing Xia and Yanhui Li. Could we predict the result of a continuous integration build? an empirical study. In *IEEE International Conference on Software Quality, Reliability and Security Companion (QRS-C)*, pages 311–315, 2017.

[40] Eddie Antonio Santos and Abram Hindle. Judging a commit by its cover. In *13th International Workshop on Mining Software Repositories-MSR*, volume 16, pages 504–507, 2016.

[41] Christoffer Rosen, Ben Grawi, and Emad Shihab. Commit guru: analytics and risk prediction of software commits. In *10th Joint Meeting on Foundations of Software Engineering*, pages 966–969, 2015.

[42] Mark Santolucito, Jialu Zhang, Ennan Zhai, and Ruzica Piskac. Statically verifying continuous integration configurations. *Technical Report*, 2018.

[43] Mark Harman, Phil McMinn, Jerffeson Teixeira De Souza, and Shin Yoo. Search based software engineering: Techniques, taxonomy, tutorial. In *Empirical software engineering and verification*, pages 1–59. 2010.

[44] Dean C Karnopp. Random search techniques for optimization problems. *Automatica*, 1(2-3):111–121, 1963.

[45] Mark Harman. The current state and future of search based software engineering. pages 342–357, 2007.

[46] Wiem Mkaouer, Marouane Kessentini, Adnan Shaout, Patrice Koligheu, Slim Bechikh, Kalyanmoy Deb, and Ali Ouni. Many-objective software remodularization using nsga-iii. *ACM Transactions on Software Engineering and Methodology (TOSEM)*, 24(3):17, 2015.

[47] Kalyanmoy Deb, Amrit Pratap, Sameer Agarwal, and TAMT Meyarivan. A fast and elitist multiobjective genetic algorithm: NSGA-II. volume 6, pages 182–197, 2002.

[48] Kalyanmoy Deb and Himanshu Jain. An evolutionary many-objective optimization algorithm using reference-point-based non-dominated sorting approach, part i: solving problems with box constraints. *IEEE Transactions on Evolutionary Computation*, 18(4): 577–601, 2013.

[49] Francesco di Pierro, Soon-Thiam Khu, and Dragan A Savic. An investigation on preference order ranking scheme for multiobjective evolutionary optimization. *IEEE Transactions on Evolutionary Computation*, 11(1):17–45, 2007.

[50] David Hadka. Moea framework user guide. 2014.

[51] Jair Cervantes, Xiaoou Li, and Wen Yu. Using genetic algorithm to improve classification accuracy on imbalanced data. In *IEEE International Conference on Systems, Man, and Cybernetics*, pages 2659–2664, 2013.

[52] Tao Chen, Miqing Li, and Xin Yao. How to evaluate solutions in pareto-based search-based software engineering? a critical review and methodological guidance. *arXiv preprint arXiv:2002.09040*, 2020.

[53] Shuai Wang, Shaukat Ali, Tao Yue, Yan Li, and Marius Liaaen. A practical guide to select quality indicators for assessing pareto-

based search algorithms in search-based software engineering. In *38th International Conference on Software Engineering*, pages 631–642, 2016.

[54] Nery Riquelme, Christian Von Lücken, and Benjamin Baran. Performance metrics in multi-objective optimization. In *Latin American Computing Conference (CLEI)*, pages 1–11, 2015.

[55] Miqing Li and Xin Yao. Quality evaluation of solution sets in multiobjective optimisation: A survey. *ACM Computing Surveys (CSUR)*, 52(2):1–38, 2019.

[56] Hong-yun Meng, Xiao-hua Zhang, and San-yang Liu. New quality measures for multiobjective programming. In *International Conference on Natural Computation*, pages 1044–1048. Springer, 2005.

[57] David Hadka. Moea framework. http://moeaframework.org/,. Accessed: 2020-12-01.

[58] Jing Xia, Yanhui Li, and Chuanqi Wang. An empirical study on the cross-project predictability of continuous integration outcomes. In *Web Information Systems and Applications Conference*, pages 234–239, 2017.

[59] Ansong Ni and Ming Li. Poster: Acona: Active online model adaptation for predicting continuous integration build failures. In *IEEE/ACM 40th International Conference on Software Engineering: Companion (ICSE-Companion)*, pages 366–367, 2018.

[60] Nitesh V Chawla, Kevin W Bowyer, Lawrence O Hall, and W Philip Kegelmeyer. Smote: synthetic minority over-sampling technique. *Journal of artificial intelligence research*, 16:321–357, 2002.

[61] Michael Hilton, Timothy Tunnell, Kai Huang, Darko Marinov, and Danny Dig. Usage, costs, and benefits of continuous integration in open-source projects. In *31st IEEE/ACM International Conference on Automated Software Engineering*, ASE 2016, pages 426–437, 2016. ISBN 978-1-4503-3845-5.

[62] GitHub. Github activity data. https://console.cloud.google.com/bigquery?p=bigquery-public-data&d=github_repos&page=dataset,. Accessed: 2020-12-01.

[63] Andrea Arcuri and Lionel Briand. A practical guide for using statistical tests to assess randomized algorithms in software engineering. In *International Conference on Software Engineering*, pages 1–10, 2011.

[64] Frank Wilcoxon, SK Katti, and Roberta A Wilcox. Critical values and probability levels for the wilcoxon rank sum test and the wilcoxon signed rank test. *Selected tables in mathematical statistics*, 1:171–259, 1970.

[65] András Vargha and Harold D Delaney. A critique and improvement of the cl common language effect size statistics of mcgraw and wong. *Journal of Educational and Behavioral Statistics*, 25(2): 101–132, 2000.

[66] Shiva Nejati and Gregory Gay. *11th International Symposium Search-Based Software Engineering*, volume 11664. 2019.

[67] Stephen W Thomas, Hadi Hemmati, Ahmed E Hassan, and Dorothea Blostein. Static test case prioritization using topic models. *Empirical Software Engineering*, 19(1):182–212, 2014.

[68] Simone Scalabrino, Giovanni Grano, Dario Di Nucci, Rocco Oliveto, and Andrea De Lucia. Search-based testing of procedural programs: Iterative single-target or multi-target approach? In *International Symposium on Search Based Software Engineering*, pages 64–79, 2016.

[69] Andrea Arcuri and Gordon Fraser. On parameter tuning in search based software engineering. In *International Symposium on Search Based Software Engineering*, pages 33–47. Springer, 2011.

[70] Scikit-learn.org. Parameter estimation using grid search with scikit-learn. available online:. https://scikit-learn.org/stable/modules/grid_search.html,, 2006. Accessed: 2020-12-01.

[71] Urvesh Bhowan, Mengjie Zhang, and Mark Johnston. Genetic programming for classification with unbalanced data. In *European Conference on Genetic Programming*, pages 1–13, 2010.

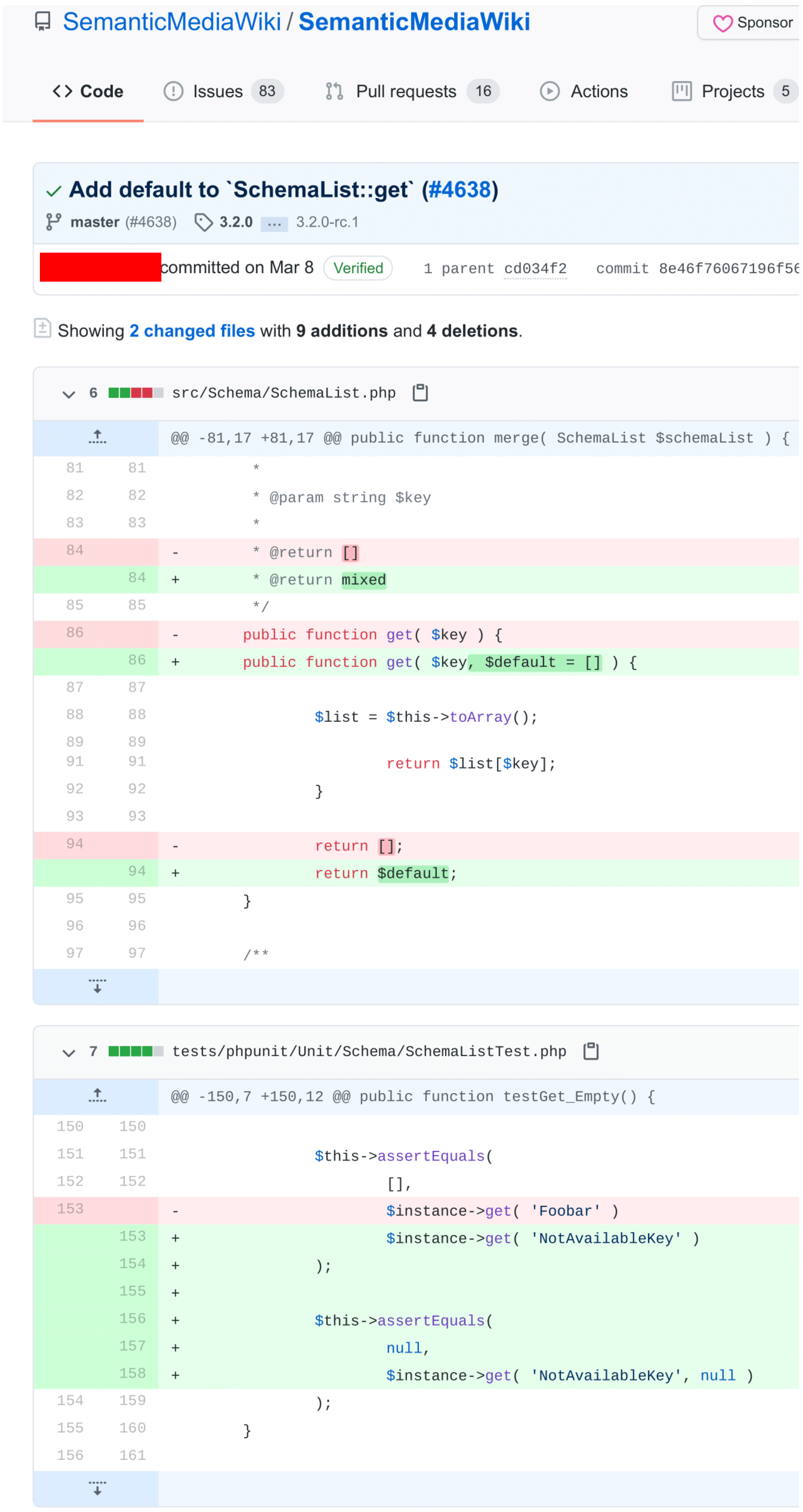
[72] Urvesh Bhowan, Mark Johnston, and Mengjie Zhang. Differentiating between individual class performance in genetic programming fitness for classification with unbalanced data. In *2009 IEEE Congress on Evolutionary Computation*, pages 2802–2809. IEEE, 2009.

[73] Xianhao Jin and Francisco Servant. A cost-efficient approach to building in continuous integration.

[74] Islem Saidani, Ali Ouni, Moataz Chouchen, and Mohamed Wiem Mkaouer. On the prediction of continuous integration build failures using search-based software engineering. In *Annual Genetic and Evolutionary Computation Conference (GECCO)*, pages 313–314, 2020.

[75] Janice Singer, Susan E Sim, and Timothy C Lethbridge. Software engineering data collection for field studies. In *Guide to Advanced Empirical Software Engineering*, pages 9–34. Springer, 2008.

[76] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. *The elements of statistical learning: data mining, inference, and prediction*. Springer Science & Business Media, 2009.



SemanticMediaWiki / **SemanticMediaWiki**

Sponsor

**Add default to `SchemaList::get` (#4638)**

master (#4638) 3.2.0 ... 3.2.0-rc.1

[redacted] committed on Mar 8 Verified 1 parent cd034f2 commit 8e46f76067196f5

Showing **2 changed files** with **9 additions** and **4 deletions**.

6 src/Schema/SchemaList.php

```
@@ -81,17 +81,17 @@ public function merge( SchemaList $schemaList ) {
          *
          * @param string $key
          *
-         * @return []
+         * @return mixed
          */
-        public function get( $key ) {
+        public function get( $key, $default = [] ) {

                $list = $this->toArray();

                        return $list[$key];
                }

-               return [];
+               return $default;
        }

        /**
```

7 tests/phpunit/Unit/Schema/SchemaListTest.php

```
@@ -150,7 +150,12 @@ public function testGet_Empty() {

                $this->assertEquals(
                        [],
-                       $instance->get( 'Foobar' )
+                       $instance->get( 'NotAvailableKey' )
+               );
+
+               $this->assertEquals(
+                       null,
+                       $instance->get( 'NotAvailableKey', null )
                );
        }

```

(a) The code change.

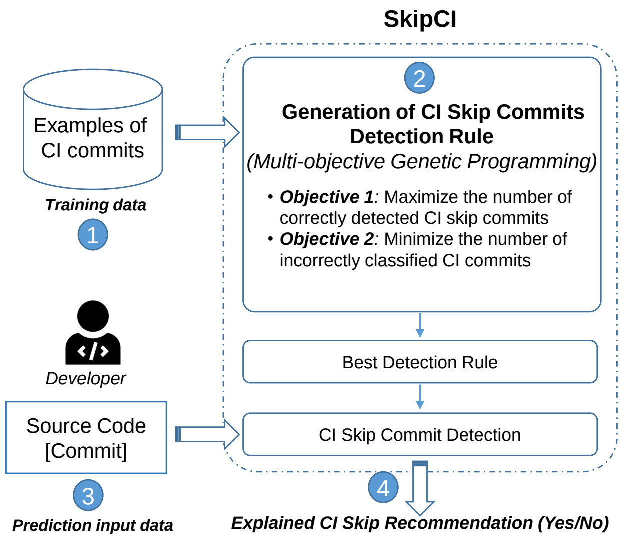

SkipCI
Examples of CI commits
Training data
1
2
Generation of CI Skip Commits Detection Rule
(Multi-objective Genetic Programming)
• Objective 1: Maximize the number of correctly detected CI skip commits
• Objective 2: Minimize the number of incorrectly classified CI commits
Developer
Best Detection Rule
Source Code [Commit]
CI Skip Commit Detection
3
Prediction input data
4
Explained CI Skip Recommendation (Yes/No)

Fig. 2: Approach overview.

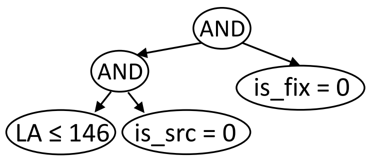


Fig. 3: Example of solution representation.

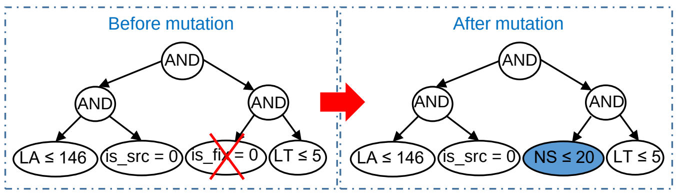


Fig. 4: An example of mutation operation.

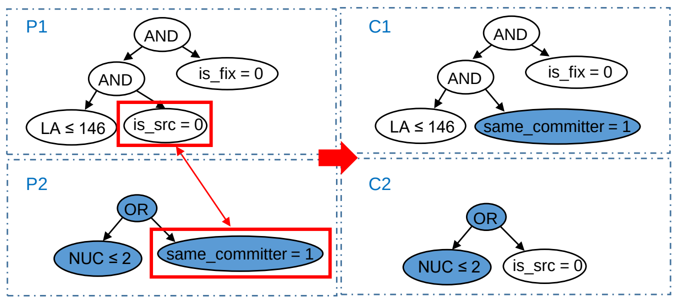


Fig. 5: An example of crossover operator.

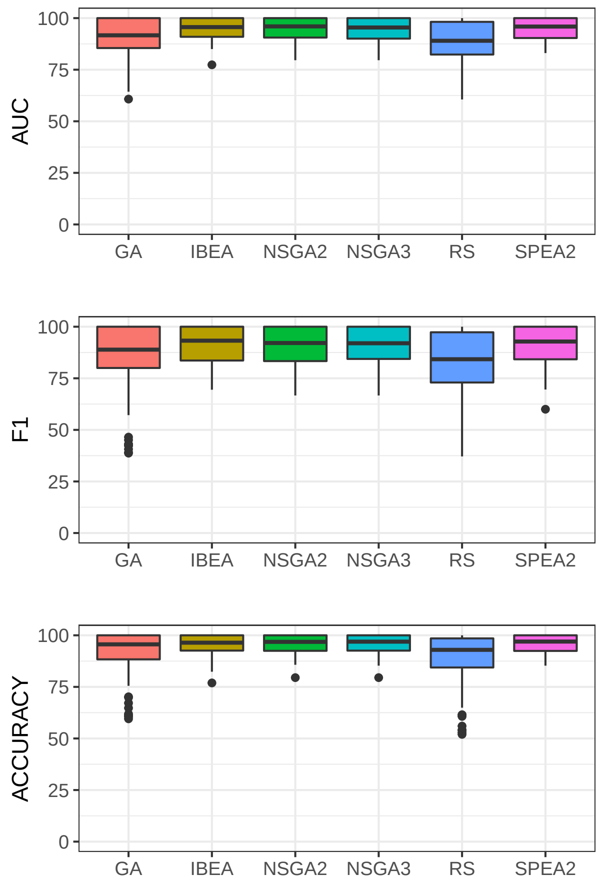


Fig. 6: Results of the search algorithms for the 4,650 experiment instances (31 runs, 10 validation iterations, 15 projects).

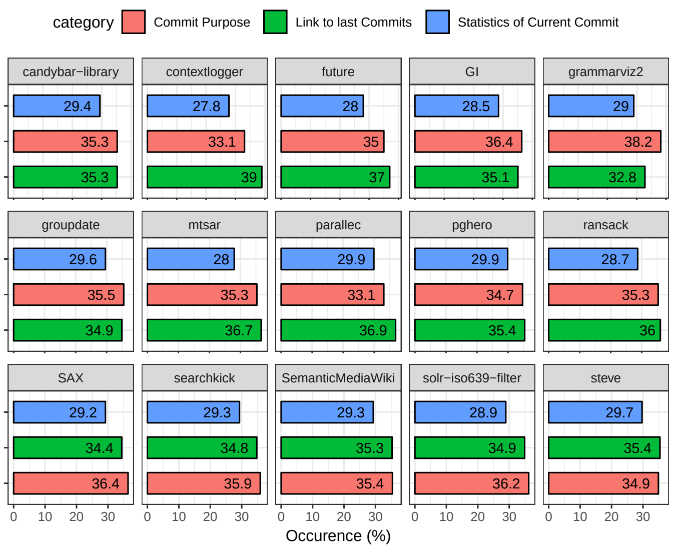


Fig. 7: Feature categories' occurrences for each project (within-project scenario).

disc_rule2.pdf

Fig. 8: An illustrative example of CI skip detection rule for the `Semantic MediaWiki` project.

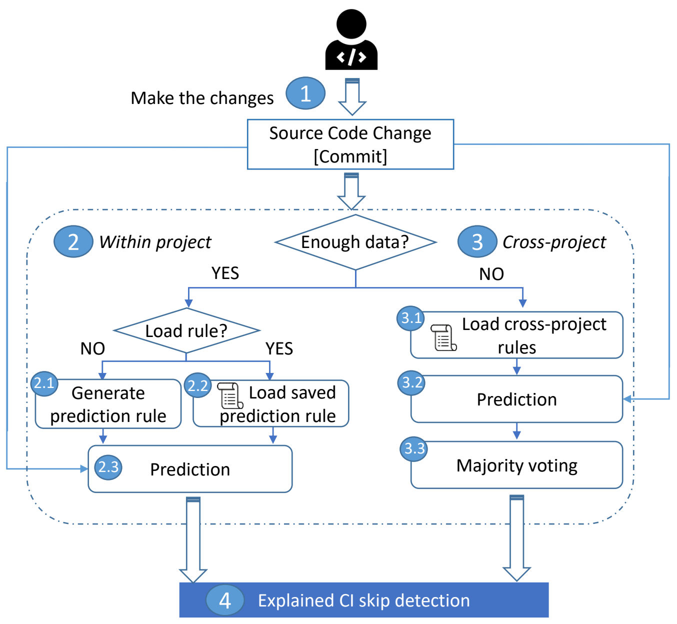
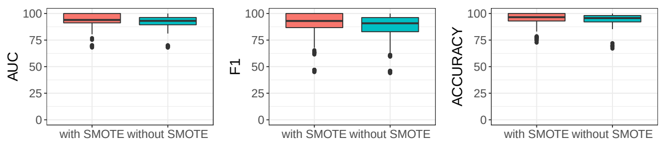

Make the changes
1
Source Code Change [Commit]
2 Within project
Enough data?
3 Cross-project
YES
NO
Load rule?
NO
YES
2.1 Generate prediction rule
2.2 Load saved prediction rule
2.3 Prediction
3.1 Load cross-project rules
3.2 Prediction
3.3 Majority voting
4 Explained CI skip detection

Fig. 9: A usage workflow of our approach.



(a) AUC (b) F1 (c) Accuracy

Fig. 10: Comparison of SKIPCI results with (red) and without (blue) using SMOTE.