Detecting Continuous Integration Skip Commits Using Multi-Objective Evolutionary Search

Islem Saidani, *Student Member, IEEE*, 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 Skip 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 validations on a benchmark of 14,294 CI commits from 15 projects that use Travis CI system. The statistical tests revealed that our approach shows a clear advantage over the baseline approaches with average scores of 92% and 84% in terms of AUC for cross-validation and cross-project validations respectively. Furthermore, the features analysis reveals that documentation changes, terms appearing in the commit message and the committer experience are the most prominent features in CI skip detection. When it comes to the cross-project scenario, the results reveal that besides the documentation changes, there is a strong link between current and previous commits results. 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]. CI systems, such as Travis CI[1], advocate to continuously integrate code changes, by automating the process of building and testing [2], 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 [3]. The presence of such slow builds severely affects both the speed and cost of software development as well as the productivity of developers [4]. Such challenges motivated the research [5], [6] 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.

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

• Islem Saidani and Ali Ouni are with the Department of Software Engineering and IT, ETS Montreal, University of Quebec, H3C 3P8 Montreal, QC, Canada. E-mail: islem.saidani.1@ens.etsmtl.ca, ali.ouni@etsmtl.ca.
• Mohamed Wiem Mkaouer is with the Department of Software Engineering at Rochester Institue of Technology, Rochester, NY 14623 USA. E-mail: mwmvse@rit.edu.

Manuscript received 11 Jan. 2021; revised 24 Sept. 2021; accepted 28 Oct. 2021. Date of publication 19 Nov. 2021; date of current version 12 Dec. 2022. This work was supported by the Natural Sciences and Engineering Research Council of Canada under Grant RGPIN-2018-05960.
(Corresponding author: Ali Ouni.)
Recommended for acceptance by S. Nejati.
Digital Object Identifier no. 10.1109/TSE.2021.3129165

The first attempt in addressing this problem [5], 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.* [6] proposed an approach based on Decision Tree (DT) that achieved satisfactory results within-project validation with 89% in terms of Area Under the ROC Curve (AUC). When it comes to cross-project validation, the prediction performance degraded to the accuracy of random guessing with 74% in terms of AUC; 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 under-used. This suggests that the skip detection problem is not yet resolved.

Detecting a skip 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 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

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

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 [7], multi-objective formulation is well suited to search-based software engineering (SBSE) [8], [9]. Recently, a number of researchers have highlighted the successful use of Multi-Objective Genetic Programming (MOGP) as an efficient method for developing prediction models [9], [10], [11]. 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 [12], [13]. This is crucial to keep the generated models accurate to the original data [14] 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) [15] 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 on a benchmark of 14,294 CI commits from 15 projects that use Travis CI system. First, we answer the following research question: *RQ1:* How effective is SkipCI compared to other existing search-based algorithms? Our multi-objective formulation has shown its effectiveness compared to other SBSE approaches. Then, we compare SkipCI to five ML techniques within-project validation in *RQ2:* How does our approach perform compared to ML techniques? and considering cross-project validation in *RQ3:* How effective is our approach when applied on cross-projects? Results show that SkipCI achieves a better performance over various baseline approaches with average AUC scores of 92% and 88% for cross-validation and cross-project validations, respectively. Next, we investigate the most important features by leveraging our generated rules in *RQ4:* What features are most important to detect skipped commits? 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 CI skip proneness. Moreover, we deployed SkipCI in an industrial setting in *RQ5:* Are the CI skip commit recommendations provided by SkipCI useful for developers who use CI in practice? 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 14,294 Travis CI commits of 15 projects that use Travis CI, by (1) comparing our approach to search-based algorithms as well as ML techniques and (2) performing a qualitative analysis to discover which features are the most prominent to determine CI skipped commits using our proper rules.
3) 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.

1.2 Replication Package

We provide our replication package containing all the materials to reproduce and extend our study [16]. 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

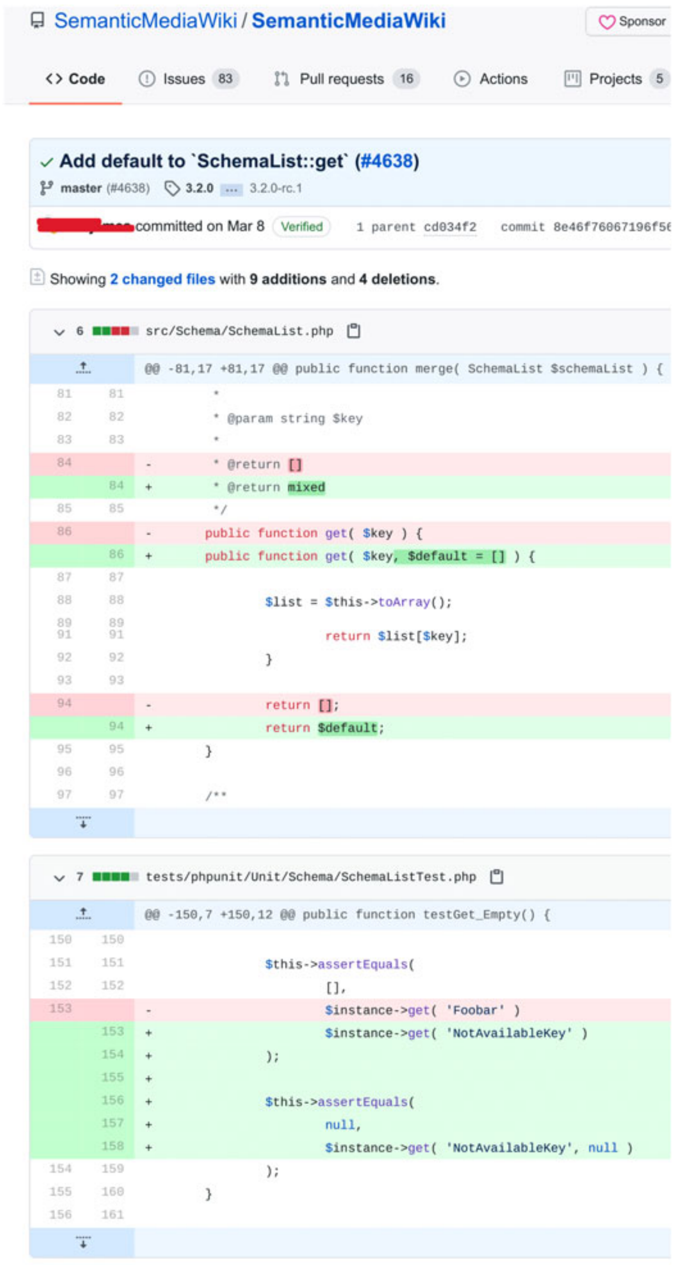
The remainder of this paper is organized as follows. In Section 2, we motivate the formulation of CI skip detection with a real-world example. Then, we explain how we adapted SPEA-2 to our problem in Section 3. Section 4 describes the experimental setup of our empirical study while Section 5 presents 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 [6] was able to improve the CI skip detection compared to the rule-based approach [5], it still misses cases of commits that should be skipped in CI. Fig. 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.
- The second file is a test file, in which the developer (*i*) changed the value to be tested in order to check

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



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 )
                );
        }

```

The code change.

Fig. 1. A motivating example extracted from the `SemanticMediaWiki` project.

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.

However, using existing approaches [5], [6], we have found that they failed to detect this commit to be skipped. Indeed, the rule-based approach [5] consists of five rules related mainly to non-source files (e.g.,documentation) and cosmetic changes (e.g., source code formatting), which explains why this approach cannot detect more sophisticated cases as this example shows. Hence, this approach is not suitable to the CI skip problem.

The machine learning approach [6] 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, when the changes are sophisticated, the problem resolution requires exploring a large search space composed of high number of possible combinations of features and their associated thresholds. 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 [14] since the original and the balanced training corpora have different characteristics. Furthermore, rebalancing can also be computationally expensive [17]. In this paper, we advocate that a MOGP technique is more suitable since it allows the evolution of solutions with optimal balance between minority and majority classes, without need to rebalance the data. In this way, the generated rules can provide useful explanations to the user.

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 detection of CI skipped commits by adapting Strength-Pareto Evolutionary Algorithm (SPEA-2) [15].

3.1 Approach Overview

Fig. 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 the 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 Multi-Objective Genetic Programming Adaptation

This section shows how MOGP is adopted to CI skip commits detection problem using SPEA-2. To ease the understanding

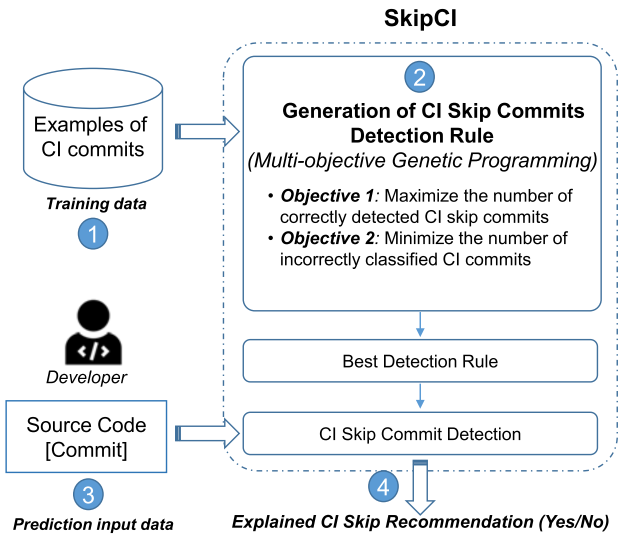

SkipCI
Examples of CI commits
Training data
1
2
Generation of CI Skip Commits Detection Rule
(Multi-objective Genetic Programming)
• Objective 1: Maximize the number of correctly detected CI skip commits
• Objective 2: Minimize the number of incorrectly classified CI commits
Developer
Best Detection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

Fig. 2. Approach overview.

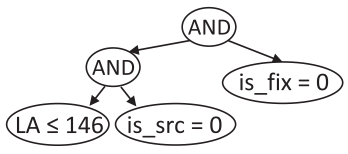


Fig. 3. Example of solution representation.

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 [18], [19], [20], to identify CI Skip 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).

Algorithm 1. Pseudo Code of SPEA-2 [15]

```
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: $A = \bar{P}_{t+1}$
```

3.2.2 Adaptation

To adapt SPEA-2 to our problem (i.e.,CI skip commit detection), we need to define (i) the initial population (line 1 in Algorithm 1), (ii) the fitness assignment (line 3), (iii) the variation operators (line 6) and (iv) the stopping criteria (line 2). The parameters setting, such as the population size, is described later in Section 4.4.

i. Initialization. Before defining the initial population generation, we need first to define the population individuals or *candidate solutions*. In MOGP, a candidate solution, i.e., a detection rule, is represented as an IF – THEN rules with the following structure [10], [21], [22], [23]:

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

The antecedent of the IF statement describes the conditions, i.e.,pairs of features 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). Fig. 3 provides an example of a solution. This rule, represented by a binary tree, detects a CI skip 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`).

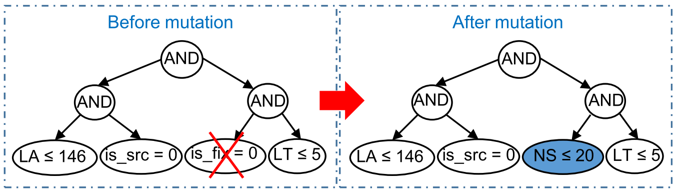


Fig. 4. An example of mutation operation.

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 features 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 features to use within the detection rule. More precisely, for each solution, we assign:

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

ii. Fitness Assignment. 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).

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

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 [11], [22] as follows:

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

where n represents the number of solutions generated by SPEA-2 TPR[i] and FPR[i] compute the TPR and FPR of solution $i \in \{1..n\}$.

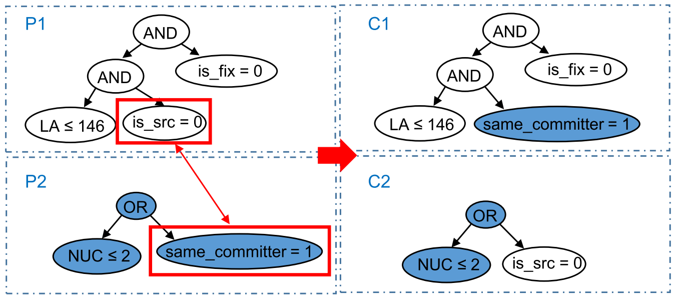


Fig. 5. An example of crossover operator.

iii. Variation.

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 feature 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 sub-tree. Fig. 4 illustrates an example of a mutation process, in which the terminal is replaced 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. Fig. 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 IS_SRC =0 **THEN** Skip Commit.

iv. Stopping Criteria. In our experiments, the algorithm stops when reaching a maximum number of generations.

3.3 Features for CI Skip Commits Detection

Table 1 lists the features, used to learn and generate our detection rules. Besides the features used by Abdalkareem *et al.* [6], we also generated other features related to the history of commits to better capture the salient characteristics of CI skip commits. Note that we wrote a Java script to compute the features based on their definition from the original paper (i.e.,column "inspired from). Please refer to our replication package for more details about the implemented script.

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 importance of terms in the commit message (i.e., CM) that has proved its efficiency to predict CI Skip commits in the recent work of Abdalkareem *et al.* [6] and other general statistics (*e.g.*, the day of the week).

TABLE 1
Features Used Form CI Skip Detection Extracted From Literature

Cat.	Feature	Description (Inspired from)	Rational
Statistics about current commit	Number of Sub-systems (NS)	The number of changed sub-systems. [6, 28, 29]	Commits that affect many subsystems are not usually likely to be CI skipped.
	Number of Directories (ND)	The number of changed directories. [6, 28, 29, 30, 31]	Commits that affect are based on many directories are not usually likely to be CI skipped.
	Number of Files (NF)	The number of changed files [6, 28, 29, 30, 31]	Usually non-trivial changes affect many files.
	ENTROPY	Measures the distribution of the change across the different files. [6, 28, 29, 30, 31]	High entropy is an indication of complicated changes.
	Lines Added (LA)	The number of added lines. [6, 28, 29]	Adding new lines means that the functionality of code has changed and hence should be tested.
	Lines Deleted (LD)	Number of deleted lines [4, 6, 25, 26, 28, 29, 30, 31, 32]	Many deleted lines indicate that the change should not be CI skipped.
	Day of the Week (DAY_WEEK)	Day of the week when the commit was performed.[25]	Routines of developers and the battle rhythms of projects can manifest themselves in CI skip activity.
	Commit Message (CM)	Measures the importance of terms appearing in the commit message using TF-IDF [6, 33]	Commit message may contain useful information about the type of commit.
	Types of Files Changed (TFC)	The number of the changed files' types identified by their extension.[3, 6, 25]	The type of files indicate the type of changes.
Commit Purpose	Classification (CLASSIF)	Feature Addition (1), Corrective (2), Merge (3), Perfective (4), Preventative (5), Non-Functional (6), None (7) [34]	The classification of the commit can be useful *e.g.* commits of category (1) cannot generally be skipped).
	Fixing Commit (IS_FIX)	Whether the commit is a bug fixing commit[6, 28, 29, 34]	Fixing bugs means that the code need to be tested.
	Documentation Commit (IS_DOC)	Whether the commit is a documentation commit, *i.e.*, contains only doc files [5, 6]	Commits that change documentation files are likely to be skipped.
	Building Commit (IS_BUILD)	Tests whether the commits contains only build files [5, 6]	if the changes in a commit are mainly related to release version,the latter is likely to be skipped.
	Meta-files Commit (IS_META)	Whether the commit a meta commit (contains only meta files) [5, 6, 35]	If a commit changes mainly meta files, it is likely to be CI skipped.
	Merging Commit (IS_MERGE)	The commit is a merge commit [6]	If the commit is a merging one, it is likely to be CI skipped.
	Media Commit (IS_MEDIA)	Whether the commit a media commit, *i.e.*, contains only media files like images	Commits that contain only changed media files are likely to be CI skipped.
	Source Commit (IS_SRC)	Whether the commit a source commit *i.e.* contains only source code files	Commits that contain only source files may need to be tested.
	Formatting Commit (FRM)	If the commit changes the formatting of the source code [5, 6]	In this case, the commit is likely to be a CI skip commit
	Comments (COM)	If the commit modifies only source code comments [5, 6]	This type of changes indicates that the commits is likely to be skipped.
	Maintenance Commit (MC)	If the commit is a maintenance activity [6]	The type of maintenance activity may indicate the need to test the changes.
Link to Last Commit(s)	Project Recent Skip (PRS)	Number of recently commits that were skipped in the 5 past commits [21, 24]	All previous skipped commits can impact the likelihood of the current commit to be skipped.
	Committer Recent Skip (CRS)	Number of recently commits that were skipped in the 5 past commits by the current committer[21, 24]	
	Previous Commit Result (PCR)	Whether previous commit was skipped or not.[21, 24]	
	Same Committer (SC)	Whether it is the same committer of the last commit [21, 24]	
	Number of Unique Changes (NUC)	The number of unique last commits of the modified files [6, 28, 29]	Larger NUC is an indication of the commit's complexity which requires testing the commit.
	AGE	The average time interval between the current and the last time these files were modified [6, 21]	Faster changes can introduce more bugs and thus need to be tested.
	Number of Developers (NDEV)	The number of developers that previously changed the touched file(s) in the past [6, 34]	The larger is NDEV, the more risky are the changes
	Lines Transformed (LT)	Size of changed files before the commit ([6, 34])	Higher LT indicates the complexity of the changed files and thus the need to be tested.
	Developer Experience (EXP)	The number of commits made by the developer before the current commit [6, 24]	Indication of how much is the developer familiar with the changed files and thus his/her knowledge about the need to skip the changes.
	Sub-system Experience (SEXP)	Subsystem experience measures the number of commits the developer made in the past to the subsystems that are modified by the current commit[6]	
	Recent Experience (REXP)	Recent experience is measured as the total experience of the developer in terms of commits, weighted by their age[6]	

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 features that are linked to last commits [6], such as the committer experience, we add and deduce other detailed features including the number of recently skipped commits (in the project and by the current committer), the feature 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 features are inspired from existing works of CI build failure detection [24], [25], [26] 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 [27]).

4 Experimental Study Design

In this section, we describe the design of our empirical study to evaluate our SkipCI approach.

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 is to evaluate the SPEA-2 formulation from an SBSE perspective as recommended by Harman and Jones [8]. First, we compare SPEA-2 against Random Search (RS) [36], [37], 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 [9], [9], [38], [39] namely Non-dominated Sorting Genetic Algorithm (NSGA-II) [40], NSGA-III [41] and Indicator-Based Evolutionary Algorithm (IBEA) [42] in terms of the quality of non-dominated solutions known as *Pareto front* in the objective space [43].

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

$$F1\text{-}score = 2 * \frac{Precision * Recall}{Precision + Recall} \in [0,1] \quad (1)$$

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 features are computed as

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

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

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 [44]

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

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

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

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 [45]. 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 [46], [47], [48].

- *Hyper-volume (HV):* is the most employed measure according to a previous study by Riquelme *et al.* [47] 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 measures [47]. 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 [49]. 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 [43], [50].

RQ2. (Within-project validation) 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 of Abdalkareem *et al.* [6]. This comparison is important to motivate the need for 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 [4], [25], [32], [35], [51], [52], [52], 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, we preprocess the raw data to scale the features using the Standard Scaler module [5].

In addition, to mitigate the issue related to the imbalanced nature of the dataset, we rely on Synthetic Minority Oversampling Technique (SMOTE) method [53], 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-score and Accuracy* that were defined previously.

RQ3.(Cross-project validation) 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 model [6] (*e.g.,* small or new projects), which may prevent the project team from using a prediction tool. Cross-project validation is the-state-of-art technique to solve the lack of training data in software engineering [51].

Approach. To evaluate our approach on the cross-project validation, we train the studied approaches based on data from 14 projects and use the remaining one project to test the trained model/rule. This experiment is repeated for all the studied project. To gain better insights into the performance of our approach, we compare it with the ML techniques used in RQ2 based on F1-score, 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 features to detect commits to be skipped. The perspective is for researchers interested in understanding how CI commits features can be related to building activities and for

4. http://moeaframework.org/
5. https://scikit-learn.org/stable/modules/generated/sklearn.preprocessing.StandardScaler.html

TABLE 2
Statistics of the Studied Projects

Project	Language	Study period	the number of commits (filtered)	CI SKIP percentage(%)	Noise level(%)
contextlogger	Java	2014-12-12 - 2017-03-14	211	31	2
SAX	Java	2015-06-20 - 2018-02-20	403	22	5
future	Java	2017-02-05 - 2018-10-02	241	25	14
solr-iso639-filter	Java	2013-08-20 - 2015-06-16	384	45	9
GI	Java	2015-06-20 - 2018-05-27	338	12	3
grammarviz2_src	Java	2014-08-22 - 2018-03-06	403	14	5
parallec	Java	2015-10-28 - 2018-01-13	124	60	14
candybar-library	Java	2017-02-22 - 2018-02-05	250	80	13
steve	Java	2015-10-07 - 2020-04-01	770	10	5
mtsar	Java	2015-05-06 - 2019-07-17	338	40	13
searchkick	Ruby	2013-08-12 - 2020-03-31	1,509	30	18
groupdate	Ruby	2013-04-25 - 2020-03-18	535	25	17
SemanticMediaWiki	PHP	2013-06-17 - 2020-04-19	6,861	19	14
ransack	Ruby	2011-07-17 - 2020-04-04	1,371	20	8
pghero	Ruby	2016-02-19 - 2020-04-06	556	25	8

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 for each project. 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, 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.* [6] which comprises ten open-source Java projects using Travis CI. Moreover, we extended this dataset based on the same filter by Abdalkareem *et al.* [6] while considering different programming languages, i.e., projects that (*i*) use Travis CI system, a popular CI service on GitHub [54], 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[6] to query on the GitHub data [55]. Our filters result in a total of 15 projects including the 10 projects studied in Abdalkareem *et al.* [6]. Note that we only consider commits from the date a given project starts using Travis CI. After this filtering process, we end up with 16,334 commits in total.

Noise Detection and Removal Since the CI skip option is under-used in practice and that developers may skip commits that cause build failure if not skipped, it is inevitable that the collected data have noise. However, the noise can cause the deterioration of the classifier performance [56], [57] and difficulty in identifying the relevant features [58]. Therefore, we must proceed with a data cleaning process to find and remove the mislabeled instances. To deal with the noise in skipped commits class, we search and find the unit tests for the skipped commits to check whether those changes can cause test failure. To identify the linked tests, we verify, for each commit, whether the names of the changed source files appear in any of the test files (e.g.,in Java, we use import or new to call the class). Fortunately, the tested files have passed so we can consider that the problem is handled for class "skipped" (i.e.,= 1). Then, we employ the Python package *cleanlab*[7] which supports PU learning (Learning from Positive and Unlabeled examples). PU learning is special case when one of the classes has no error. In fact, P stands for the positive class and is assumed to have zero label errors (the skipped commits in our case) and U stands for unlabeled data which we assume contains some positive examples (the non skipped commits). The goal of PU learning is to (1) estimate the proportion of positives in the negative class, (2) find the errors, and (3) train on clean data. Using this approach, we removed around 12% of the data which left us with 14,294 commits. Table 2 provides some statistics about the studied projects and provides information about the level of noise in each project.

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 [59]. Additionally, in order to provide support for the conclusions derived from the obtained results, we use Wilcoxon signed rank test [60], a non-parametric statistical test method to detect significant performance differences with a 95% confidence level (α is set at 0.05). Vargha-Delaney A (VDA) [61] is also used to measure the effect size. This non-parametric method is widely recommended in SBSE context [62]. It indicates the probability that one technique will achieve better

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

7. https://github.com/cgnorthcutt/cleanlab

TABLE 3
Parameters' Settings for ML Techniques Under Comparison

Algorithm	Parameters
RF	Max depth of the tree =10 the number of estimators = 400
DT	Max depth of the tree=10
NB	Used NB model= Bernoulli NB
LR	Max iterations= 200 penalty = 'l2'
SVC	C=1 kernel='rbf' Max iterations= 2000

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 [63]. Vargha-Delaney statistic also classifies the magnitude of the obtained effect size value into four different levels: *negligible*, *small*, *medium*, and *large* [21], [64].

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 [65]. This is also compulsory when using ML techniques [14]. There is no optimal parameters' setting to solve all problems, therefore, we used a trial-and-error method to select the hyper-parameters [9] to handle parameter tuning for search-based algorithms which is a common practice in SBSE [9]. These parameters are fixed as follows: population size = 100; maximum the number of generations = 500; crossover probability =0.7; and mutation probability = 0.1.

As for ML techniques, we employed *Grid Search* [66], an exhaustive search-based tuning method widely used in practice. We report in Table 3, the parameters' setting of the ML techniques used in this study.

5 Experimental Study Results

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

5.1 Results of SBSE Validation (RQ1)

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

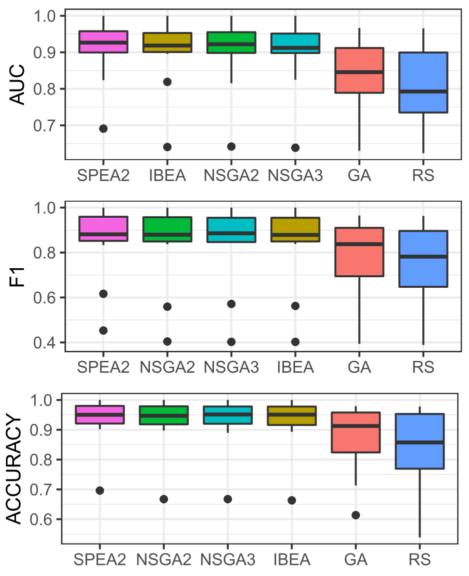


Fig. 6. Results of the search algorithms for the 4,650 experiment instances (31 runs, 10 validation iterations, 15 projects).

As shown in Fig. 6, GA achieved in median 85%, 84% and 91% in terms of AUC, F1-score and accuracy, respectively, while RS has shown a lower predictive performance of 80%, 78% and 86% in terms of AUC, F1-score 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, at least, a difference of 6%, 5% and 4% in terms of AUC, F1-score 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 SBSE validation.

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 0.97, compared to 0.95

TABLE 4
Statistical Tests Results of SPEA-2 Compared to Other Search-Based Techniques Under Cross-Validation

		versus IBEA	versus NSGA-II	versus NSGA-III	versus GA	versus RS
AUC	*p-value*	1	1	1	$< 10^{-16}$	$< 10^{-16}$
	A estimate	0.52	0.52	0.53	0.76	0.79
	Magnitude	N	N	N	L	L
F1	*p-value*	1	1	1	$< 10^{-16}$	$< 10^{-16}$
	A estimate	0.52	0.51	0.51	0.68	0.71
	Magnitude	N	N	N	L	L
Accuracy	*p-value*	1	1	1	$< 10^{-16}$	$< 10^{-16}$
	A estimate	0.53	0.52	0.52	0.71	0.75
	Magnitude	N	N	N	L	L

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

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	**0.97**	0.95	0.94	0.94
GD	**0.08**	0.10	0.14	0.13
SP	**0.07**	0.05	0.05	0.05

achieved by NSGA-II and 0.94 achieved by NSGA-III and IBEA respectively. In terms of GD, SPEA-2 achieved a better distance between its generated solutions and the optimal one (i.e.,Pareto front) by reaching 0.08 compared to 0.10, 0.14 and 0.13 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 93% of AUC, 88% of F1-score and 95% of accuracy in median. SPEA-2 achieved also higher optimization performance among the studied MOGP algorithms, which motivates our choice to use it a search-based approach.

5.2 Results of Within Project Validation (RQ2)

Table 6 (Appendix A, which can be found on the Computer Society Digital Library at http://doi.ieeecomputersociety.org/10.1109/TSE.2021.3129165) reports the average (of 10 cross-validation iterations) AUC, F1-score 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 all the studied projects, the best scores were obtained by SPEA-2

TABLE 6
Performance of SPEA-2 Versus ML Techniques Under Cross-Validation

Project	AUC						F1						Accuracy					
	SPEA-2	*SVC*	*RF*	*LR*	*NB*	*DT*	*SPEA-2*	*SVC*	*RF*	*LR*	*NB*	*DT*	*SPEA-2*	*SVC*	*RF*	*LR*	*NB*	*DT*
candybar-library	**99**	91	87	87	85	79	**99**	94	93	93	90	91	**98**	91	90	89	85	86
contextlogger	**100**	97	100	97	99	99	**100**	97	100	96	98	98	**100**	98	**100**	98	99	99
future	**100**	96	98	95	93	82	**100**	88	98	91	87	71	**100**	94	99	95	92	79
GI	**95**	90	91	93	86	89	**92**	85	88	85	77	71	**98**	97	97	96	95	93
grammarviz2	**93**	89	91	**93**	86	85	**92**	85	87	87	73	62	**98**	97	97	97	93	85
groupdate	**93**	85	83	85	83	73	**86**	77	72	76	73	59	**94**	88	85	87	85	71
mtsar	**91**	79	78	75	73	67	**88**	73	72	68	67	62	**92**	79	79	75	74	66
parallec	**96**	95	93	95	**96**	94	**97**	94	91	94	95	93	**96**	94	90	93	94	92
pghero	**92**	85	86	84	83	72	**85**	75	75	75	72	57	**93**	88	86	88	86	68
ransack	**89**	88	85	88	86	67	**85**	83	72	77	71	47	**95**	94	88	91	86	56
SAX	**99**	93	94	92	92	93	**99**	91	93	88	87	86	**99**	97	97	95	94	94
searchkick	**90**	**90**	**90**	89	87	79	83	**85**	83	83	81	68	90	**91**	89	89	89	77
SemanticMediaWiki	**69**	52	56	66	65	53	**45**	33	35	43	42	32	**70**	25	46	**70**	67	38
solr-iso639-filter	**90**	76	84	78	72	73	**88**	69	80	75	68	71	**90**	77	85	78	72	73
steve	**82**	80	75	77	79	61	**62**	46	50	39	47	25	**94**	86	91	82	87	73
Median	**93**	89	87	88	86	79	**88**	85	83	83	73	68	**95**	91	90	89	87	77
Average	**92**	86	86	86	84	78	**87**	78	79	78	75	66	**94**	86	88	88	87	77

TABLE 7
Statistical Tests' Results of SPEA-2 Compared to ML Techniques Under Cross-Validation

		versus DT	versus RF	versus LR	versus NB	versus SVC
AUC	*p-value*	$< 10^{-16}$	$< 10^{-16}$	$< 10^{-16}$	$< 10^{-16}$	$< 10^{-16}$
	A estimate	0.77	0.62	0.65	0.71	0.64
	Magnitude	L	S	S	M	S
F1	*p-value*	$< 10^{-16}$	$< 10^{-16}$	$< 10^{-16}$	$< 10^{-16}$	$< 10^{-16}$
	A estimate	0.77	0.60	0.66	0.70	0.64
	Magnitude	L	S	S	M	S
Accuracy	*p-value*	$< 10^{-16}$	$< 10^{-16}$	$< 10^{-16}$	$< 10^{-16}$	$< 10^{-16}$
	A estimate	0.82	0.63	0.70	0.74	0.67
	Magnitude	L	S	M	L	M

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

TABLE 8
Performance of SPEA-2 Versus ML Techniques Under Cross-Project Validation

Project	AUC						F1						Accuracy					
	SPEA2	*NB*	*LR*	*RF*	*SVC*	*DT*	*SPEA2*	*NB*	*LR*	*RF*	*SVC*	*DT*	*SPEA2*	*NB*	*LR*	*RF*	*SVC*	*DT*
candybar-library	**78**	**78**	**78**	71	60	59	**92**	86	72	84	85	81	**86**	79	65	75	76	70
contextlogger	**98**	**98**	**98**	53	60	57	**97**	96	96	49	53	51	**98**	97	98	36	45	41
future	**91**	89	71	38	48	45	**86**	78	55	32	38	37	**93**	87	70	21	28	23
GI	**89**	88	86	54	53	53	**87**	62	56	24	23	23	**97**	86	82	20	18	17
grammarviz2	**93**	85	87	55	54	52	**92**	57	65	27	26	26	**98**	80	87	25	22	20
groupdate	**84**	**84**	83	68	69	48	72	**74**	**74**	51	52	37	85	**86**	**86**	58	60	32
mtsar	**72**	68	68	66	56	53	**68**	64	60	63	59	54	**71**	67	**71**	63	49	47
parallec	95	94	**97**	85	58	76	96	96	**97**	91	78	86	95	95	**97**	88	66	81
pghero	80	**85**	82	57	55	50	68	**75**	72	43	43	40	82	**86**	85	46	37	29
ransack	**86**	83	84	59	54	54	**84**	61	67	37	35	34	**95**	78	84	37	26	27
SAX	**95**	86	88	55	53	51	**95**	69	74	38	38	36	**98**	82	86	32	27	24
searchkick	86	**88**	**88**	70	50	48	79	**82**	**82**	58	45	44	87	**89**	**89**	64	31	31
SemanticMediaWiki	**64**	61	62	63	63	62	**41**	37	39	39	40	38	69	73	**76**	68	76	68
solr-iso639-filter	75	65	**77**	71	61	45	**75**	65	72	73	67	44	73	64	**79**	68	57	45
steve	75	68	**77**	41	43	45	**52**	38	47	13	14	14	**91**	88	87	12	14	14
Median	**86**	85	83	59	55	52	**84**	69	72	43	43	38	**91**	86	85	46	37	31
Average	**84**	81	82	60	56	53	**79**	69	69	48	46	43	**87**	82	83	47	42	38

achieving on average 93% with an improvement of at least 4% over the best ML algorithm (SVC). Additionally, the statistical analysis underlines the significant difference with *small* to *large* VDA effect sizes (cf. Table 7). For instance, in the `mtsar` project, our approach obtained an AUC score of 91% while we recorded scores of 79%, 78%, 75%, 73% and 67% for SVC, RF, LR, NB and DT respectively. 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 [21], [67].

Looking at F1-score, we also observe that SPEA-2 achieved the best results for 14 out of 15 projects with an average score of 87% compared to 79% achieved by RF the best performer among ML techniques. The statistical tests' results reveal that SPEA-2 outperforms ML with small (compared to SVC, LR, RF), medium (compared to NB) and large (with DT) effect sizes. Hence, F1-score results demonstrate a compelling superiority of SPEA-2 to identify more CI skip 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.

However, in the `searchkick` project, SVC is slightly better by achieving a F1-score of 85% compared to 83% for SPEA2. But due to the black-box nature of SVC (it returns probabilities for belonging to a certain class), we cannot have an comprehensible explanation for this result.

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 92% and 87% in terms of AUC and F1-score, respectively, while reaching 94% of the overall classification accuracy.*

TABLE 9
Statistical Tests' Results of SPEA-2 Under Cross-Projects Compared to Its Achieved Within-Project Results, RF and DT (*)

		versus Cross-Validation	versus DT	versus RF
AUC	*p-value*	$< 10^{-16}$	$< 10^{-16}$	$< 10^{-16}$
	A estimate	0.27	0.98	0.94
	Magnitude	M	L	L
F1	*p-value*	$< 10^{-16}$	$< 10^{-16}$	$< 10^{-16}$
	A estimate	0.34	0.91	0.87
	Magnitude	S	L	L
Accuracy	*p-value*	$< 10^{-16}$	$< 10^{-16}$	$< 10^{-16}$
	A estimate	0.31	0.97	0.94
	Magnitude	M	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 as we only recorded 15 observation for each of them.

5.3 Results of Cross-Project Validation (RQ3)

In this RQ, we compare SkipCI with ML techniques under cross-project validation, using our evaluation measures, the Area Under the ROC Curve (AUC), F1-score, and accuracy, to measure the performance of our approach. Table 8 (Appendix A, available in the online supplemental material) 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 15 observations for each of them, under the cross-project validation. Recall that in cross-project validation, we train the studied approaches based on data from 14 projects and use the remaining project's data as a testing set and this experiment is repeated for the 15 projects.

First, the results show that SPEA-2 achieves high scores of AUC with a median of 86% while the values range from 63-97%. We observe that 10 out of 15 projects showed high performance results ($\geq$80%). In particular, `contextlogger` achieves a significant AUC score of 98%. Additionally, we observe an improvement of 2% on average over LR the best performer. 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 11 out of 15 projects by achieving 84% on median compared to ML techniques that achieved F1-scores of 72% for LR, 69% for NB, 43% for RF as well as SVC and 38% for DT. 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 9 projects out of 15 by achieving in median 91% (and 87% on average) and the accuracy values range from 61-98%. 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.

In some projects such as `parallec` and `pghero`, NB and LR showed a slightly better performance than SPEA2 but again due to the black-box nature of these algorithms, it is not possible to have an explanation for this result. This recalls the need for explainable classification in machine learning.

However, compared to the within-project validation (RQ2), the results indicate that our approach can achieve a less significant performance with small to medium effect sizes, which may be explained by the fact under cross-project, the target project may have a low collinearity with the source project features 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 validation, our SkipCI *still outperforms state-of-the-art ML techniques by achieving an average of 84%, 79% and 87%* in terms of AUC, F1-score 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 of Features Analysis (RQ4)

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

5.4.1 Within-Project Results

Tables 10 summarizes the ranking of the most occurring features among all the studied projects considering the cross-validation scenario.

TABLE 10
Top-Features Analysis for All Projects (Within-Project Validation)

TOP-1	# of rules
IS_DOC	34
CM	27
EXP	23
CRS	21
PRS	17
ENTROPY	8
LA	7
SEXP	6
NDEV	3
REXP	2
IS_FIX	1
ND	1
Total	150

Link to Last Commits category has shown to be a strong indicator of the commit likelihood to be CI skip as six out of the 12 top-features belong to this category. Additionally, theses features appear the most in 72 out of 150 rules. For example, the committer experience i.e.,*EXP*, the top three feature, appears the most in 23 rules out of 150. Similarly to *EXP*, *SEXP* and *REXP* appear also in the top-10 features which indicates that the experienced developers are more familiar with CI features. We observe also that the feature *CRS*, i.e., number of recently skipped commits by the committer, is the most appearing feature in 21 out of 150. A closer examination reveals that this feature has a clear indication of whether a commit should be CI skip or not. For example, in `candybar-library` 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 85% of the commits in this project. A similar observation can be applied in `searchkick` project, in which we also observed that having at least 2 recently commits alone would detect the CI skip commits in this project with a F1-score of 78%. Similarly to *CRS* feature, *PRS* also seems to be relevant and appears the most in 17 out of 150 rules. Thus, it is clear that, similar to build failures [24], 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. *NDEV*, i.e.,the number of developers that previously touched the changed files, is also a prominent feature and appears the most in three out of 150 rules. For example, in `candybar-library` project, the condition of having $NDEV \leq 3$ would detect alone the CI skip commits in this project with a F1-score of 89%. This indicates that less is *NDEV*, the less risky are the changes and thus the commit can be safely skipped.

Statistics about the current commit have a strong probability to indicate whether this commit should be skipped and four of these features appear in the top-1 list and the most in 43 out of 150 rules. First, terms appearing in the commit message can be useful as *CM* is the most occurring feature in 27 out of 150 rules. For example, in `contextlogger` 90% of skipped commit messages contain "Update Readme. md". This finding was previously stated by Abdalkareem *et al.* [6]. In addition, features providing statistics about the current commit change size can also be useful to detect CI skip commits. For example, in the project `mtsar` for which the number of added lines LA is the most important feature, 60% of skipped commits have $LD \leq 23$ which indicates that small changes are more likely to be skipped. The same observation is applied to *ENTROPY* and *ND*.

Commit Purpose are also important in detecting CI Skip commits as they appear the most in 35 out of 150 rules. Additionally, the top-1 feature, i.e.,*IS_DOC* belongs to this category. For instance, in `parallec` project, the condition $IS\_DOC = 1$ can detect alone the CI skip commits with 95% of F1-score, and in `candybar-library` project 90% of the non-fixing commits, i.e.,$IS\_FIX = 0$, are skipped, which is consistent with the real world situation as fixing bugs/problems should be tested. Abdalkareem *et al.* [6] also pointed out that these features (i.e.,CI skip rules as mentioned in the paper) can be strong indicators for CI skip detection.

5.4.2 Cross-Project Results

The top-features analysis under cross-project validation is presented in Table 11. This table clearly indicates that, similarly to within-project results, *IS_DOC* is the most important feature across the studied projects. This result is in line with the observations of Abdalkareem *et al.* [5] as they found that 52% of skipped commits are those that touch documentation or non-source code files. Additionally, statistics linked to last commits are prominent features in cross-project context. Specifically, recent skipped commits from all developers in the project *PRS* are dominant in 4 out of 15 rules while the number of recently skipped commits from the current committer *CRS* is dominant in 3 out of the 15 cross-project rules. This strengthens our previous findings claiming that if the commit is skipped, the next commit is more likely to be skipped as well.

Another important observation to consider is that the statistics of the current commit is not present in the top-features list, which indicates that these features 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).

Summary for RQ4. Within-project validation, the feature analysis reveals that (1) whether the commit changes only documentation files i.e.,IS_DOC, (2) terms appearing in the commit message and (3) the committer experience are the most prominent features in CI skip detection. When it comes to the cross-project rules, the results confirm that IS_DOC is a dominant feature and 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 Skip 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:

TABLE 11
Top-Features Analysis for All Projects (Cross-Project Validation)

TOP-1	the number of rules
IS_DOC	8
PRS	4
CRS	3
Total	15

RQ5.Are the CI skip commit recommendations provided by SkipCI *useful for developers who use CI 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. The CI system provides the CI skip option, similar to Travis CI. Both projects had 24% and 31% of commits are skipped in project-1 and project-2, respectively, during the last 3 years. Usually, developers working on both projects, integrate their code changes more than once per day. As for case study participants, we have reached out to 36 developers working full time on both projects and invited them to participate in our experiments. A total number of 14 developers volunteered to participate in the experiments, 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 tool 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 for both

TABLE 12
Participants Partition Statistics

Project	Tool	Group	the number of developers	the number of commits
Project-1	SkipCI	A1	4	61
	Random	B1	4	52
Project-2	SkipCI	A2	3	43
	Random	B2	3	48

studied projects. Thereafter, we deployed SkipCI and integrated it into the CI service pipeline for both projects. The SkipCI tool 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 tool 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.

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 declined 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 *declined* 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

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, 🟥Declined.

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 highly appreciated 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 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).

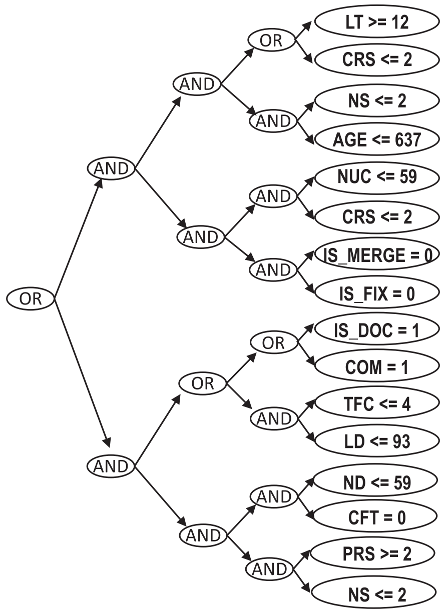


Fig. 7. An illustrative example of CI skip detection rule for the `Semantic MediaWiki` project.

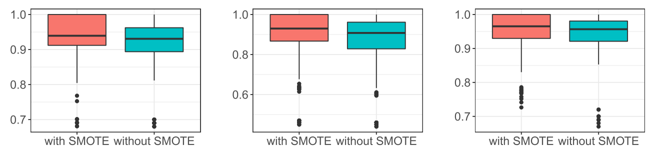
From this user study, we learn that to improve SkipCI, we have to include three aspects. First, we need to deploy along with the tool, a set of interpretable and configurable rules so that developers can understand and customize the tool based on their preferences/experience on what should or should 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 Discussion and Implications

7.1 For CI Developers

Developers can efficiently identify the CI Skip 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, Fig. 7 illustrates an example of a detection rule generated by SkipCI to detect CI Skip commits in the `Semantic MediaWiki`[8] project

8. https://github.com/SemanticMediaWiki/SemanticMediaWiki



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

Fig. 8. Comparison of SkipCI results with (red) and without (blue) using SMOTE.

with a high AUC score of 80%. Using this rule, we can detect the CI commit described early in Section 2 as CI skip since its characteristics satisfy the conditions of the rule, *e.g.*, having a number of changed files $NS \leq 2$, $LD \leq 93$ and $IS\_FIX = 0$. 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 [21], [68].

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 Skip commits of SkipCI, even though the latter has shown less sensitivity to the class imbalance problem as pointed out in prior research [12], [13], [17], [67]. 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. Fig. 8 shows the obtained results with (in red) and without (in blue) using SMOTE.

We observe that by using SMOTE, SkipCI can achieve in median 94%, 93% and 96% in terms of AUC, F1-score and accuracy respectively, which represents an improvement of 1%, 5% and 1%, respectively with negligible (for AUC and accuracy) to small (for F1-score) effect sizes. 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 [14] as the original and the balanced training corpora have different characteristics. Furthermore, rebalancing can also be computationally expensive [17]. 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 skipped commits. Our features analysis results (RQ4) reveal that many CI skip commits occurred consecutively and features related to historical statistics about the commits are strong indicators of the current commit outcome. Hence, we 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 Skip Commits

Abdalkareem *et al.* [5] are the first to examine the CI commits that can be CI skip 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 five rules related mainly to non-source files (e.g.documentation) and cosmetic changes (e.g.. source code formatting). 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-score.

A related effort for improving CI skip detection, by Abdalkareem *et al.* [6], 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 [5], the used DT model achieved higher F1-score of 79% within project validation and 55% under cross-project validation. Additionally, they investigated the most prominent features 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.

In a similar context, Kawrykow *et al.* [69] proposed a rule-based approach to detect non-essential changes (and hence worth to be skipped). The authors defined six rules including documentation updates, trivial type modifications and rename-induced modifications. The proposed approach correctly classified non-essential updates with 92.8% of precision on average along 7 open source systems. However, we believe that this approach cannot be successfully adopted in a CI context, as the proposed rules miss the updates related to a CI environment such as those related to preparing releases (which represent 15% of CI skip changes according to [5]). This suggests that similarly to Abdalkareem *et al.* [5] rule-based approach, this work would induce many cases where the commit is worth to be skipped (i.e. false negatives).

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 [25] leveraged the history of Ant, Maven and Gradle build systems in order to predict CI build failure. Their approach achieved over 90% in terms of AUC on average. Xia and Li [32] employed and compared nine ML models to train CI build failure prediction models. Another work by Jin and Servant [70] 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.* [21], [71] proposed a MOGP technique to predict CI build failures by adapting the NSGA-II algorithm [40]. 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 features 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 %takes the advantage of MOGP which has already shown good performance in solving other CI-related problems [21], [71] and several other software engineering problems [9], [72]. In particular, SkipCI aims at taking one step forward to speed up the CI build process and reduce its costs by identifying CI skip commits. It is worth noting that SkipCI could be used jointly with existing CI build failure prediction techniques [21], [32], [70] 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 declining 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.

Threats to construct validity are mainly related to the rigor of the study design. First, one possible threat is related to the selected performance metrics as there exist many other metrics. We basically used standard performance metrics namely F1-score, AUC and accuracy that are widely employed in predictive models comparison [73] 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. Another threat to construct validity is related to our industrial validation since we considered the random search algorithm as baseline. We plan as a future work to extend this validation with other baseline approaches.

Third, a threat to construct validity could be related to the annotated set of skipped commits used in our dataset. To mitigate this issue, we ran unit tests that are linked to the skipped commits and checked that none of those tests would fail. Second, to deal with the noise of the negative class, we employed the python package *cleanlab* which helped us to detect about 12% of the data as noise. We also checked manually the precision of the detected labels by verifying that at least those non skipped commits are not correlated with a build failure. We found that the precision of this tool ranges from 80% to 100%. Nevertheless, it may misses other commits that should have been skipped (i.e., the recall). 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 [66] 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 15 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 under-used in practice and hence labeled training data is not always available. Additionally, while in our approach we focus on Travis CI, a popular CI system [54], 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 14,294 Travis CI commits of 15 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 validations 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 documentation changes,terms appearing in the commit message and the committer experience are the most prominent features in CI skip detection. When it comes to the cross-project scenario, the results reveal that besides the documentation changes, there is a strong link between current and previous commits results.

Our future research agenda includes performing a larger empirical study with other open-source projects while considering other features. For instance, one can measure the code similarity before and after the commit to identify whether the changes do probability preserve the code functionality and hence can be skipped. The tool can also detect whether the developer performed a code refactoring [27] such as renaming. This would allow the tool to detect other CI skip opportunities that may have not been detected yet using the current features.

Acknowledgments

A special thanks to the engineers who participated in this study for their time and feedback.

References

[1] B. Vasilescu, Y. Yu, H. Wang, P. Devanbu, and V. Filkov, "Quality and productivity outcomes relating to continuous integration in gitHub," in *Proc. 10th Joint Meeting Found. Softw. Eng.*, 2015, pp. 805–816.

[2] M. Fowler, "Continuous integration," 2006. [Online]. Available: https://www.martinfowler.com/articles/continuousIntegration.html

[3] T. A. Ghaleb, D. A. da Costa, and Y. Zou, "An empirical study of the long duration of continuous integration builds," *Empirical Softw. Eng.*, pp. 1–38, 2019.

[4] Y. Luo, Y. Zhao, W. Ma, and L. Chen, "What are the factors impacting build breakage?," in *Proc. Web Inf. Syst. Appl. Conf.*, 2017, pp. 139–142.

[5] R. Abdalkareem, S. Mujahid, E. Shihab, and J. Rilling, "Which commits can be CI skipped?," *IEEE Trans. Softw. Eng.*, vol. 47, no. 3, pp. 448–463, Mar. 2021.

[6] R. Abdalkareem, S. Mujahid, and E. Shihab, "A machine learning approach to improve the detection of CI skip commits," *IEEE Trans. Softw. Eng.*, to be published, doi: 10.1109/TSE.2020.2967380.

[7] U. Bhowan, M. Johnston, and M. Zhang, "Evolving ensembles in multi-objective genetic programming for classification with unbalanced data," in *Proc. Annu. Conf. Genetic Evol. Comput*, 2011, pp. 1331–1338.

[8] M. Harman and B. F. Jones, "Search-based software engineering," *Inf. Soft. Technol.*, vol. 43, no. 14, pp. 833–839, 2001.

[9] M. Harman, S. A. Mansouri, and Y. Zhang, "Search-based software engineering: Trends, techniques and applications," *ACM Comput. Surv.*, vol. 45, no. 1, pp. 1–16, 2012.

[10] M. Kessentini and A. Ouni, "Detecting android smells using multi-objective genetic programming," in *Proc. Int. Conf. Mobile Softw. Eng. Syst.*, 2017, pp. 122–132.

[11] A. Ouni, M. Kessentini, H. Sahraoui, K. Inoue, and K. Deb, "Multi-criteria code refactoring using search-based software engineering: An industrial case study," *ACM Trans. Softw. Eng. Methodol.*, vol. 25, no. 3, pp. 1–53, 2016.

[12] U. Bhowan, M. Johnston, M. Zhang, and X. Yao, "Evolving diverse ensembles using genetic programming for classification with unbalanced data," *IEEE Trans. Evol. Comput.*, vol. 17, no. 3, pp. 368–386, Jun. 2013.

[13] U. Bhowan, M. Johnston, M. Zhang, and X. Yao, "Reusing genetic programming for ensemble selection in classification of unbalanced data," *IEEE Trans. Evol. Comput.*, vol. 18, no. 6, pp. 893–908, Dec. 2014.

[14] C. Tantithamthavorn, S. McIntosh, A. E. Hassan, and K. Matsumoto, "The impact of automated parameter optimization on defect prediction models," *IEEE Trans. Softw. Eng.*, vol. 45, no. 7, pp. 683–711, 2018.

[15] E. Zitzler, M. Laumanns, and L. Thiele, "SPEA2: Improving the strength pareto evolutionary algorithm," TIK-Rep., vol. 103, pp. 1–21, 2001.

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

[17] U. Bhowan, M. Johnston, and M. Zhang, "Differentiating between individual class performance in genetic programming fitness for classification with unbalanced data," in *Proc. IEEE Congr. Evol. Comput.*, 2009, pp. 2802–2809.

[18] F. Zhao, W. Lei, W. Ma, Y. Liu, and C. Zhang, "An improved SPEA2 algorithm with adaptive selection of evolutionary operators scheme for multiobjective optimization problems," *Math. Problems Eng.*, vol. 2016, pp. 1–20, 2016.

[19] S. Garcia and C. T. Trinh, "Comparison of multi-objective evolutionary algorithms to solve the modular cell design problem for novel biocatalysis," *Processes*, vol. 7, no. 6, 2019, Art. no. 361.

[20] S. Sofianopoulos and G. Tambouratzis, "Studying the SPEA2 algorithm for optimising a pattern-recognition based machine translation system," in *Proc. IEEE Symp. Comput. Intell. Multicriteria Decis.-Mak.*, 2011, pp. 97–104.

[21] I. Saidani, A. Ouni, M. Chouchen, and M. W. Mkaouer, "Predicting continuous integration build failures using evolutionary search," *Inf. Softw. Technol.*, vol. 128, 2020, Art. no. 106392.

[22] A. Ouni, M. Kessentini, H. Sahraoui, and M. Boukadoum, "Maintainability defects detection and correction: A multi-objective approach," *Automated Softw. Engi.*, vol. 20, no. 1, pp. 47–79, 2013.

[23] A. Ouni, M. Kessentini, K. Inoue, and M. O. Cinnéide, "Search-based web service antipatterns detection," *IEEE Trans. Services Comput.*, vol. 10, no. 4, pp. 603–617, Jul./Aug. 2017.

[24] A. Ni and M. Li, "Cost-effective build outcome prediction using cascaded classifiers," in *Proc. IEEE/ACM 14th Int. Conf. Mining Softw. Repositories*, 2017, pp. 455–458.

[25] F. Hassan and X. Wang, "Change-aware build prediction model for stall avoidance in continuous integration," in *Proc. ACM/IEEE Int. Symp. Empirical Softw. Eng. Meas.*, 2017, pp. 157–162.

[26] Z. Xie and M. Li, "Cutting the software building efforts in continuous integration by semi-supervised online AUC optimization." in *Proc. 27th Int. Joint Conf. Artif. Intell.*, 2018, pp. 2875–2881.

[27] M. Fowler, *Refactoring: Improving the Design of Existing Code*. Boston, MA, USA: Addison-Wesley Professional, 2018.

[28] W. Fu and T. Menzies, "Revisiting unsupervised learning for defect prediction," in *Proc. 11th Joint Meeting Found. Softw. Eng.*, 2017, pp. 72–83.

[29] Y. Kamei *et al.*, "A large-scale empirical study of just-in-time quality assurance," *IEEE Trans. Softw. Eng.*, vol. 39, no. 6, pp. 757–773, Jun. 2012.

[30] S. McIntosh and Y. Kamei, "Are fix-inducing changes a moving target? A longitudinal case study of just-in-time defect prediction," *IEEE Trans. Softw. Eng.*, vol. 44, no. 5, pp. 412–428, May 2018.

[31] Y. Yang *et al.*, "Effort-aware just-in-time defect prediction: Simple unsupervised models could be better than supervised models," in *Proc. 24th ACM SIGSOFT Int. Symp. Found. Softw. Eng.*, 2016, pp. 157–168.

[32] J. Xia and Y. Li, "Could we predict the result of a continuous integration build? An empirical study," in *Proc. IEEE Int. Conf. Softw. Qual., Rel. Secur. Companion*, 2017, pp. 311–315.

[33] E. A. Santos and A. Hindle, "Judging a commit by its cover," in *Proc. 13th Int. Workshop oMining Softw. Repositories-MSR*, 2016, vol. 16, pp. 504–507.

[34] C. Rosen, B. Grawi, and E. Shihab, "Commit guru: Analytics and risk prediction of software commits," in *Proc. 10th Joint Meeting Found. Softw. Eng*, 2015, pp. 966–969.

[35] M. Santolucito, J. Zhang, E. Zhai, and R. Piskac, "Statically verifying continuous integration configurations," Tech. Rep., 2018, *arXiv:1805.04473*.

[36] M. Harman, P. McMinn, J. T. De Souza , and S. Yoo, "Search based software engineering: Techniques, taxonomy, tutorial," in *Proc. Empirical Softw. Eng. Verification*, 2010, pp. 1–59.

[37] D. C. Karnopp, "Random search techniques for optimization problems," *Automatica*, vol. 1, no. 2–3, pp. 111–121, 1963.

[38] M. Harman, "The current state and future of search based software engineering," *Future Softw. Eng.*, pp. 342–357, 2007.

[39] W. Mkaouer *et al.*, "Many-objective software remodularization using NSGA-iii," *ACM Trans. Softw. Eng. Methodol.*, vol. 24, no. 3, pp. 1–45, 2015.

[40] K. Deb, A. Pratap, S. Agarwal, and T. Meyarivan, "A fast and elitist multiobjective genetic algorithm: NSGA-II," in *IEEE Trans. Evol. Comput.*, vol. 6, no. 2, pp. 182–197, Apr. 2002.

[41] K. Deb and H. Jain, "An evolutionary many-objective optimization algorithm using reference-point-based nondominated sorting approach, part I: Solving problems with box constraints," *IEEE Trans. Evol. Comput.*, vol. 18, no. 4, pp. 577–601, Aug. 2014.

[42] F. di Pierro, S.-T. Khu, and D. A. Savic, "An investigation on preference order ranking scheme for multiobjective evolutionary optimization," *IEEE Trans. Evol. Comput.*, vol. 11, no. 1, pp. 17–45, Feb. 2007.

[43] D. Hadka, "MOEA framework user guide: A free and open source java framework for multiobjective optimization," *Free Softw. Found.*, Version 2.4, pp. 1–192, 2014.

[44] J. Cervantes, X. Li, and W. Yu, "Using genetic algorithm to improve classification accuracy on imbalanced data," in *Proc. IEEE Int. Conf. Syst., Man, Cybern.*, 2013, pp. 2659–2664.

[45] T. Chen, M. Li, and X. Yao, "How to evaluate solutions in pareto-based search-based software engineering? A critical review and methodological guidance," 2020, *arXiv: 2002.09040*.

[46] S. Wang, S. Ali, T. Yue, Y. Li, and M. Liaaen, "A practical guide to select quality indicators for assessing pareto-based search algorithms in search-based software engineering," in *Proc. 38th Int. Conf. Softw. Eng.*, 2016, pp. 631–642.

[47] N. Riquelme, C. Von Lücken, and B. Baran, "Performance metrics in multi-objective optimization," in *Proc. Latin Amer. Comput. Conf.*, 2015, pp. 1–11.

[48] M. Li and X. Yao, "Quality evaluation of solution sets in multiobjective optimisation: A survey," *ACM Comput. Surv.*, vol. 52, no. 2, pp. 1–38, 2019.

[49] H.-y. Meng, X.-h. Zhang, and S.-y. Liu, "New quality measures for multiobjective programming," in *Proc. Int. Conf. Natural Comput.*, 2005, pp. 1044–1048.

[50] D. Hadka, "Moea framework," Accessed: Dec. 01, 2020, [Online]. Available: http://moeaframework.org/

[51] J. Xia, Y. Li, and C. Wang, "An empirical study on the cross-project predictability of continuous integration outcomes," in *Proc. Web Inf. Syst. Appl. Conf.*, 2017, pp. 234–239.

[52] A. Ni and M. Li, "Poster: ACONA: Active online model adaptation for predicting continuous integration build failures," in *Proc. IEEE/ACM 40th Int. Conf. Softw. Eng., Companion*, 2018, pp. 366–367.

[53] N. V. Chawla, K. W. Bowyer, L. O. Hall, and W. P. Kegelmeyer, "SMOTE: Synthetic minority over-sampling technique," *J. Artif. Intell. Res.*, vol. 16, pp. 321–357, 2002.

[54] M. Hilton, T. Tunnell, K. Huang, D. Marinov, and D. Dig, "Usage, costs, and benefits of continuous integration in open-source projects," in *Proc. 31st IEEE/ACM Int. Conf. Automated Softw. Eng.*, 2016, pp. 426–437.

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

[56] S. Gupta and A. Gupta, "Dealing with noise problem in machine learning data-sets: A systematic review," *Procedia Comput. Sci.*, vol. 161, pp. 466–474, 2019.

[57] B. Frénay and M. Verleysen, "Classification in the presence of label noise: A survey," *IEEE Trans. Neural Netw. Learn. Syst.*, vol. 25, no. 5, pp. 845–869, 2013.

[58] R. Ekambaram, D. B. Goldgof, and L. O. Hall, "Finding label noise examples in large scale datasets," in *Proc. IEEE Int. Conf. Syst., Man, Cybern.*, 2017, pp. 2420–2424.

[59] A. Arcuri and L. Briand, "A practical guide for using statistical tests to assess randomized algorithms in software engineering," in *Proc. Int. Conf. Softw. Eng.*, 2011, pp. 1–10.

[60] F. Wilcoxon, S. Katti, and R. A. Wilcox, "Critical values and probability levels for the wilcoxon rank sum test and the wilcoxon signed rank test," *Sel. Tables Math. Statist.*, vol. 1, pp. 171–259, 1970.

[61] A. Vargha and H. D. Delaney, "A critique and improvement of the CL common language effect size statistics of mcgraw and wong," *J. Educ. Behav. Statist.*, vol. 25, no. 2, pp. 101–132, 2000.

[62] A. Arcuri and L. Briand, "A hitchhiker's guide to statistical tests for assessing randomized algorithms in software engineering," *Softw. Testing Verification Rel.*, Wiley Online Library, vol. 24, no. 3, pp. 219–250, 2014.

[63] S. W. Thomas, H. Hemmati, A. E. Hassan, and D. Blostein, "Static test case prioritization using topic models," *Empirical Softw. Eng.*, vol. 19, no. 1, pp. 182–212, 2014.

[64] S. Scalabrino, G. Grano, D. Di Nucci , R. Oliveto, and A. De Lucia , "Search-based testing of procedural programs: Iterative single-target or multi-target approach?," in *Proc. Int. Symp. Search Based Softw. Eng.*, 2016, pp. 64–79.

[65] A. Arcuri and G. Fraser, "On parameter tuning in search based software engineering," in *Proc. Int. Symp. Search Based Softw. Eng.*, 2011, pp. 33–47.

[66] Scikit-learn.org, "Parameter estimation using grid search with scikit-learn. available online:," Accessed: Dec. 01, 2020: [Online]. Available: https://scikit-learn.org/stable/modules/grid_search.html

[67] U. Bhowan, M. Zhang, and M. Johnston, "Genetic programming for classification with unbalanced data," in *Proc. Eur. Conf. Genetic Program.*, 2010, pp. 1–13.

[68] I. Saidani, A. Ouni, M. Chouchen, and M. W. Mkaouer, "BF-detector: An automated tool for CI build failure detection," in *Proc. ACM Joint Meeting Eur. Softw. Eng. Conf. Symp. Found. Softw. Eng.*, 2021, pp. 1530–1534.

[69] D. Kawrykow and M. P. Robillard, "Non-essential changes in version histories," in *Proc. 33rd Int. Conf. Softw. Eng.*, 2011, pp. 351–360.

[70] X. Jin and F. Servant, "A cost-efficient approach to building in continuous integration," in *Proc. ACM/IEEE 42nd Int. Conf. Softw. Eng.*, 2020, pp. 13–25

[71] I. Saidani, A. Ouni, M. Chouchen, and M. W. Mkaouer, "On the prediction of continuous integration build failures using search-based software engineering," in *Proc. Annu. Genet. Evol. Comput. Conf.*, 2020, pp. 313–314.

[72] A. Ouni, "Search-based software engineering: Challenges, opportunities and recent applications," in *Proc. Genetic Evol. Comput. Conf.*, 2020, pp. 1114–1146.

[73] T. Hastie, R. Tibshirani, and J. Friedman, *The Elements of Statistical Learning: Data Mining, Inference, and Prediction*. Berlin, Germany: Springer Science & Business Media, 2009.

Islem Saidani (Student Member, IEEE) received the BSc degree and MSc degrees from the University of Carthage, Tunisia, in 2015 and 2017, respectively. She is currently working toward the PhD degree with ETS Montreal, Canada. She has authored or coauthored several papers in well-ranked journals and conferences. Her research interests include continuous integration, empirical software engineering, search-based software engineering, and machine learning.

Ali Ouni received the PhD degree in computer science from the University of Montreal in 2015. He is currently an associate professor with the Department of Software Engineering and IT, ETS Montreal, University of Quebec, where he leads the Software Technology and Intelligence (STI) Research Lab, since 2017. His research interests include software engineering including software maintenance and evolution, refactoring of software systems, software quality, service-oriented computing, and the application of artificial intelligence techniques to software engineering. He was a program committee member and reviewer in several journals and conferences. He is a member of the ACM and IEEE Computer Society.

Mohamed Wiem Mkaouer received the PhD degree from the University of Michigan-Dearborn in 2016. He is currently an assistant professor with the Software Engineering Department, in the B. Thomas Golisano College of Computing and Information Sciences, Rochester Institute of Technology. His research interests include software quality, refactoring, and software testing. He is the lead of the Software Maintenance and Intelligent Evolution (SMILE) Research Group. He is a member of the Association for Computing Machinery and the IEEE Computer Society.