

Feature-FL: Feature-Based Fault Localization

Yan Lei , *Member, IEEE*, Huan Xie , Tao Zhang , *Senior Member, IEEE*, Meng Yan, Zhou Xu , and Chengnian Sun 

I. INTRODUCTION

Abstract—Fault localization aims at developing an effective methodology identifying suspicious statements potentially responsible for program failures. The spectrum-based fault localization is the widely used methodology by analyzing the statistical coincidences viewed from the spectrum to evaluate the suspiciousness of each statement of being faulty. However, just analyzing statistical coincidences in the coverage information perspective and without combining diverse amount of information may restrict fault localization effectiveness. Thus, this article proposes feature-based fault localization (**Feature-FL**): A family fault localization methodology of feature-based metrics by combining the feature diversity from the view of program features into suspiciousness evaluation. Specifically, **Feature-FL** defines a concept of branching execution probability to abstract program behaviors as the values of features. Then, **Feature-FL** uses feature selection (i.e., a family of feature-based metrics) to evaluate the relevance of each feature with program failures. Finally, **Feature-FL** associates each feature with its corresponding statement, and uses the relevance as the suspiciousness to locate suspicious statements. We present six feature-based metrics for **Feature-FL**, and conduct an extensive study to evaluate the effectiveness of **Feature-FL** and its potential over the state-of-the-art spectrum-based formulas. Our results provide insight into the potential among different feature-based metrics and also show **Feature-FL** significantly outperforms the state-of-the-art spectrum-based formulas, e.g., an average *saving* of at least 30% over spectrum-based formulas in case of real faults.

Index Terms—Execution probability, fault localization, feature selection, statistical debugging, suspiciousness.

Manuscript received August 2, 2021; revised October 28, 2021 and December 19, 2021; accepted December 28, 2021. Date of publication February 2, 2022; date of current version March 2, 2022. This work was supported in part by National Defense Basic Scientific Research Project under Grant WZC20205500308, in part by the Fundamental Research Funds for the Central Universities under Grant 2021CDJQY-018, in part by Chongqing Science and Technology Plan Projects under Grant cstc2018jszx-cyztzxX0037, in part by the Key Project of Technology Innovation and Application Development of Chongqing under Grant cstc2019jcsx-mbdxX0020, in part by the National Natural Science Foundation of China under Grant 62102054, and in part by the Open Foundation of Key Laboratory of Dependable Service Computing in Cyber Physical Society, Ministry of Education of China under Grant CPSDSC202004. Associate Editor: R. Gao. (*Corresponding author: Yan Lei.*)

Yan Lei, Huan Xie, Meng Yan, and Zhou Xu are with the Key Laboratory of Dependable Service Computing in Cyber Physical Society, Ministry of Education, Chongqing University, Chongqing 400044, China, and also with the School of Big Data and Software Engineering, Chongqing University, Chongqing 400044, China (e-mail: yanlei@cqu.edu.cn; huanxie@cqu.edu.cn; mengy@cqu.edu.cn; zhouxullx@cqu.edu.cn).

Tao Zhang is with the Faculty of Information Technology, Macau University of Science and Technology, Macau 999078, China (e-mail: tazhang@must.edu.mo).

Chengnian Sun is with the Cheriton School of Computer Science, University of Waterloo, Waterloo, ON N2L 3G1, Canada (e-mail: cnsun@uwaterloo.ca).

Color versions of one or more figures in this article are available at <https://doi.org/10.1109/TR.2022.3140453>.

Digital Object Identifier 10.1109/TR.2022.3140453

BEING an essential activity in software development and maintenance, debugging is a process of locating and fixing bugs. For debugging engineers, debugging is a painstaking task because it usually requires them to consume a significant amount of time and resource in pinpointing the location of a bug and understanding its cause of a failure [1], [2]. In order to improve debugging performance, researchers try to develop effective fault localization approaches, such as [3]–[14], to assist debugging engineers by presenting some advice on suspicious locations potentially responsible for the failures.

Among existing fault localization approaches, spectrum-based fault localization (SFL) is a widely used and studied methodology, showing its promising results over other fault localization approaches [15], [16]. SFL [17] generally uses code coverage information and test results to evaluate the suspiciousness of each statement of being faulty, and outputs a ranking list of all statements in descending order of their suspiciousness. The basic idea of SFL is that if a statement is executed by more failing test cases and less passing test cases, the statement should have a higher suspiciousness value of being faulty. Based on this idea, researchers develop many types of SFL techniques, such as Ochiai [18], Jaccard [19], Tarantula [20], and Wong [21], advancing the state-of-the-art SFL techniques rapidly. Furthermore, both theoretical and experimental studies [15], [16], [22], [23] have found the maximal SFL techniques, showing the maximal effectiveness that SFL techniques can reach.

Although the ability in fault localization of SFL techniques has been intensively studied by many researchers, existing statistics- or probability-based SFL methods have clear limitations. For instance, the statements in the same block (e.g., “if” statement, “while” statement, and “for” statement) without nested blocks will be assigned with the same suspiciousness value because those statements will be executed sequentially and then have the same coverage information accordingly. Apart from that, many SFL methods use code coverage information and test results for statistical analysis without considering various information from the program (e.g., structure information, error message, and the textual information of the program). Those limitations could heavily restrict the effectiveness of SFL techniques. Thus, there is an urgent need to utilize various information besides coverage information and reformulate fault localization problems from a different perspective. Recently, various deep learning models are used for more precise fault localization [7], [8], [10], [24]–[28]. Among those models, many of them also take code coverage information and the corresponding test results as inputs, aiming to learn the nonlinear

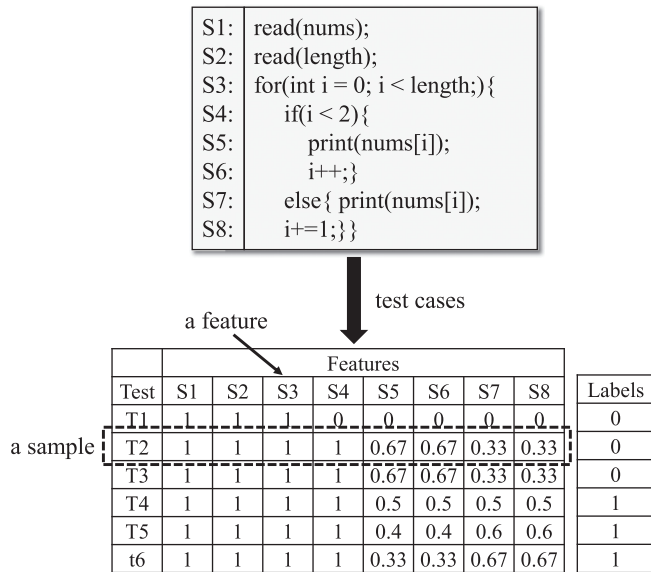


Fig. 1. An example to show the features, samples, and labels from the feature perspective.

relationship between statements and failures. Their empirical studies indicate that deep learning models can locate bugs more effectively than traditional SFL techniques, which means that there are distinguishing features (i.e., specific statements) that can make a distinction more effectively between passing test cases and failing ones. Motivated by these existing works, we take the code coverage information as samples that are composed of features. Thus, we can reformulate the fault localization problem by finding the failure-relevant features. In our article, we focus on fault localization techniques at the statement level, so each executable statement is a feature. If the granularity is at the method or class level, the feature should be a method or a class.

Fig. 1 shows an example that demonstrates the features, samples, and labels from the feature perspective. We can observe that there are eight executable statements with the line number at the top of Fig. 1, which means there are corresponding eight features (i.e., from “S1” to “S8”). When the program executes the test cases in the test suite, we will obtain all samples by collecting coverage information. And we can get the corresponding label by comparing the actual output and the expected one.

Thus, we can handle fault localization from a different perspective. We notice that a recent popular method named feature selection in machine learning domain may be potential for effective fault localization. Feature selection [29], [30] (also known as variable selection or attribute selection), in machine learning and statistics, is a process of selecting a subset of relevant features (e.g., variables, predictors) from data for improving the data use in analysis, model construction, abnormal feature detection, etc. To be more precise, feature selection methods are usually implemented to identify and remove irrelevant features from data that do not contribute to the accuracy of a predictive model or may in fact decrease the accuracy of the model [31]. Various studies show that feature selection is effective to remove irrelevant features without performance deterioration [32], [33].

Fault localization targets at finding the faulty statements that are the cause of program failures, and feature selection aims at removing the irrelevant features that are not contributed with a specific objective. Thus, there exists a natural connection between fault localization and feature selection. Since we take the code coverage information as samples, we can apply the process of feature selection into fault localization. Specifically, feature selection removes the nonfaulty statements (i.e., irrelevant features) and reserves the faulty statements (i.e., key features) that are contributed to program failures.

Therefore, we try to explore using feature selection for establishing an effective fault localization methodology in this article. We face a new challenge when bridging the gap between fault localization and feature selection: How to abstract program behaviors as the values of features (i.e., statements). Suppose we use binary coverage information (i.e., the value is 0 or 1 for each feature) as the program behavior like many traditional SFL techniques, which may drop a lot of essential information related to the program itself. Without delicate program behavior related to the program itself, the feature selection algorithm may be unable to capture the important features that lead to program failures. Thus, a more accurate description of program behavior is needed by the feature selection.

Based on the above idea, we propose feature-based fault localization (Feature-FL) to identify the suspicious statements potentially responsible for program failures from the feature perspective. Specifically, Feature-FL introduces a concept of branching execution probability as the values of features to depict each statement’s behavior with branching or nonbranching features. In other words, each statement has its own value denoted by branching execution probability. Branching execution probability is based on the branching module, and the branching module is a set of code that consists of the decision-making expression and the code that needs to be executed based on the evaluation result of the decision-making expression. The calculation of branching execution probability is defined in Section III-D.

Then, Feature-FL uses the test results (*passing* or *failing*) of each test case as labels. Next, Feature-FL uses feature selection to evaluate the relevance between each statement (i.e., feature) and the labels. A high relevance means the feature has a large effect on the labels, which means the behavior of the statement has a large impact on test results.

Thus, Feature-FL uses the relevance evaluated by feature selection as the suspiciousness to distinguish the suspicious statements.

Feature-FL takes the branch information of the program into consideration. Many existing studies also combine various dimensional information. SOBER [34] is a fault localization technique that also takes advantage of branch information. The main idea of SOBER is that the results of predicates (i.e., “if” statements) by failing test cases are different from that by passing test cases, which means SOBER can detect the abnormal behavior of the branches and locate the bug in predicates (i.e., “if” statements). Different from SOBER, Feature-FL mainly focuses on bugs of the statements in the branching module, not on that in “if” statements. The two kinds of fault localization

techniques may be complementary with each other and further studies should be conducted to validate. Apart from SOBER, there are some recent studies that utilize various information of the subject programs. DEEPRL4FL [28] devises an enhanced coverage matrix and adopts a test case ordering technique to fully explore the learning ability of the convolutional neural network. DeepFL [10] collects the spectrum-based suspiciousness, mutation-based suspiciousness, complexity-based fault proneness, and textual similarity information for more precise localization. ABLFL [35] takes more comprehensive features, i.e., statistical information (e.g., number of strings, number of integers, and number of operators), semantic information (e.g., code complexity and textual similarity), and dynamic information (e.g., coverage matrix, stack trace, and dynamic program slice), into account for localization. Generally, one method with more information tends to be more effective and less efficient than that with less information. However, the effectiveness also relies on the algorithms and the quality of input data.

Since feature selection consists of different feature selection algorithms, it means that Feature-FL is a family methodology of different criteria. Based on the six basic and typical feature selection criteria (i.e., Pearson correlation coefficient [30], Spearman's rank correlation coefficient [36], Kendall rank correlation coefficient [37], mutual information [30], Fisher score [29], [38], and Chi-squared test [39]), we present six feature selection criteria for Feature-FL.

We conduct a large-scale experimental study on real-life programs to study the fault localization potential among these feature-based methods, and also compare Feature-FL to four state-of-the-art SFL formulas [16]. The experimental results provide insight into the effectiveness distribution among different feature-based methods, and show that Feature-FL significantly outperforms the four state-of-the-art spectrum-based formulas (i.e., Dstar [15], Ochiai [40], Barinel [41], and Op2 [17]), e.g., the average saving is at least 40%.

The contribution of this article can be summarized as follows.

- We propose Feature-FL: A family fault localization methodology by abstracting program behaviors as the values of features and linking the features with failures to evaluate suspicious statements potentially responsible for failures.
- We show the promising prospect of combining the feature view of feature selection into fault localization for improving its effectiveness.
- We present six basic and typical feature-based methods of Feature-FL, and study their fault localization effectiveness distribution.
- We conduct an extensive study in two scenarios (i.e., artificial faults and real faults) to evaluate Feature-FL over the state-of-the-art spectrum-based formulas, showing its better effectiveness in improving fault localization.

The remainder of this article is organized as follows. Section II introduces background information. Section III presents our approach Feature-FL. Section IV shows the experimental results and analysis. Section V discusses related work, and Section VI concludes this article.

II. BACKGROUND

This section will introduce the methodology of feature selection and the four state-of-the-art fault localization techniques.

A. Feature Selection

Feature selection is a process of selecting only a subset of measured features (variables, predictors) to construct a model [29], [30], in an attempt to reduce the dimensionality of the training problem. The usual implementation of feature selection methods is to recognize and eliminate unneeded, irrelevant, and redundant attributes from data that do not result in the accuracy of a predicative model or possibly degrade the accuracy of the model [31]. The objective of feature selection has three different levels: Enhancing the prediction success rate of the predictors, presenting more efficient and cost-effective predictors, and producing a deeper understanding of the underlying process which produced the data. Consequently, feature selection is widely used for reducing the amount of data to deal with and the effect of the noise produced in the process, leading to a performance enhancement of the whole system.

In general, feature selection algorithms designed with different strategies are classified into three categories: Filter, wrapper, and embedded methods. The filter methods depend on the general characteristics of data and assess features without involving any learning algorithm. In contrast, the wrapper methods need a predetermined learning algorithm and utilizing its performance as evaluation criterion to select features. Embedded methods treat variable selection as a part of the training process, and feature relevance is obtained analytically from the objective of the customized learning model. Feature selection with filter and embedded methods may either produce the scores (measuring feature relevance) of all features or a subset of selected features. According to the type of the output, they can be divided into feature scoring and subset selection algorithms. Feature selection with wrapper methods usually output a subset of chosen features.

Since filter methods do not specify any learning algorithm in comparison with wrapper and embedded methods, this article considers filter methods. There are many types of filtering feature selection criteria, among which Pearson correlation coefficient [30], Spearman's rank correlation coefficient [36], Kendall rank correlation coefficient [37], mutual information [30], Fisher score [29], and Chi-squared test [39] are the basic and typical ones.

Table I lists the six feature selection criteria, where each one shows how to evaluate the relevance between an arbitrary attribute or variable X_i of the data X and the target labels of Y . Pearson correlation coefficient [30] is one of the simplest method, where $cov()$ is the covariance and σ is the standard deviation. Spearman's rank correlation coefficient [36] between two variables is equal to the Pearson correlation between the rank values of those two variables. Kendall rank correlation coefficient [37] is a statistic used to measure the ordinal association between two measured quantities. The mutual information [30] of two random variables is a measure of the mutual dependence between the two variables. Fisher score [29] refers to a vector of

TABLE I
FORMULAS OF TYPICAL FILTERING FEATURE SELECTION CRITERIA

Criterion	Formula	Criterion	Formula
Pearson	$\rho_{X_i, Y} = \frac{\text{cov}(X_i, Y)}{\sigma_{X_i} \sigma_Y}$	Mutual information	$I(X_i Y) = \sum_{x \in X_i} \sum_{y \in Y} p(x, y) \log \frac{p(x, y)}{p(x)}$
Spearman	$r_s = \rho_{rg_{X_i}, rg_Y} = \frac{\text{cov}(rg_{X_i}, rg_Y)}{\sigma_{rg_{X_i}} \sigma_{rg_Y}}$	Fisher score	$J_{\text{fisher}(i)} = \frac{S_B^{(i)}}{S_W^{(i)}}$
Kendall	$\tau_B = \frac{n_c - n_d}{\sqrt{(n_0 - n_1)(n_0 - n_2)}}$	Chi-squared test	$\chi^2 = \sum \frac{(\text{observed} - \text{expected})^2}{\text{expected}}$

The rg_* in formula of Spearman correlation coefficient means the ranking list of *. For more detailed information in formula of Kendall rank correlation coefficient and Chi-squared, refer to [42] and [43].

parameter derivatives of loglikelihood in a complicated model. In the formula of Fisher score, $S_B^{(i)}$ is named between-class variance of the i th feature, and $S_W^{(i)}$ is named within-class variance of the i th feature. Chi-squared test [39] is a statistical hypothesis test that is valid to perform when the test statistic is chi-squared distributed under the null hypothesis.

B. Fault Localization Techniques

Pearson *et al.* [16] recently have evaluated and summarized the existing fault localization techniques, identifying several optimal fault localization techniques, where we select the four best techniques. Here we describe the principle of the four localization techniques: Dstar [15], Ochiai [40], Barinel [41], and Op2 [17]. The four localization techniques are generally categorized into SFL, i.e., the four are all SFL methods. SFL utilizes the execution coverage information and test results to construct program spectrum, specifying an execution profile of program behavior. Based on program spectrum, SFL aims to establish an effective suspiciousness evaluation formula for evaluating the suspiciousness of a statement being faulty. Designing an evaluation formula generally shares an idea that if a statement is executed more often by failing test cases and less often by passing test cases, this statement should have a high suspiciousness value of being faulty. Following the idea, researchers have proposed many types of SFL techniques, among which this article considers Dstar [15], Ochiai [40], Barinel [41], and Op2 [17]. The recent studies have shown that Dstar, Ochiai, and Barinel are the most effective SFL techniques [16] and Op2 is a type of maximal formula [22]. The following equations show the suspiciousness evaluation structure of the four localization techniques.

$$Dstar : SP(s) = \frac{failing(s)^*}{passing(s) + (totalfailing - failing(s))} \quad (1)$$

$$Ochiai : SP(s) = \frac{failing(s)}{\sqrt{totalfailing \times (failing(s) + passing(s))}} \quad (2)$$

$$Barinel : SP(s) = 1 - \frac{passing(s)}{passing(s) + failing(s)} \quad (3)$$

$$Op2 : SP(s) = failing(s) - \frac{passing(s)}{1 + totalpassing} \quad (4)$$

where variable $*$ $> 0^1$. s represents a statement and $SP(s)$ denotes the suspiciousness of the statement s . $failing(s)$ is the number of those failing test cases executing the statement s . Similarly, $passing(s)$ is the number of those passing test cases executing the statement s . $totalfailing$ is the number of failing test cases. $totalpassing$ is the number of passing test cases.

III. FEATURE-BASED FAULT LOCALIZATION

A. Overview of Feature-FL

This section will present the methodology of Feature-FL, showing how Feature-FL takes advantage of feature selection to construct a fault localization methodology using the feature view.

Fig. 2 shows the overall pipeline of Feature-FL. Feature-FL runs the test cases in the test suite of a faulty program and collects the coverage information just like the traditional SFL techniques do. Different from SFL, Feature-FL identifies the branches of the program and saves the branch information. Then, Feature-FL uses the location of the branching module and the coverage information to calculate the branching execution probability. Thus, we obtain the coverage matrix with branching execution probability, which includes the abstraction of the program behavior. Finally, Feature-FL utilizes the power of feature selection algorithms to calculate the relevance between each statement and program failure, and uses the relevance as the suspiciousness value of each statement.

The contents of this section are organized as follows.

- We first explain the preliminary definition of the branching execution probability in Section III-B, i.e., the calculation of branching execution probability is not practical from the theoretical perspective.
- Then, we give the formal definition about the data we used for Feature-FL by constructing the FSet in Section III-C.
- We depict the calculation of branching execution probability in detail in Section III-D based on the formal definitions in two former subsections.
- As we obtained the data, we can evaluate the suspiciousness by using feature selection. We describe the process in detail in Section III-E.
- From the contents above, we demonstrate the details of Feature-FL. To be more illustrative about Feature-FL, we give a motivation example in Section III-F.

¹We used the most thoroughly explored value [16], i.e., $*$ = 2.

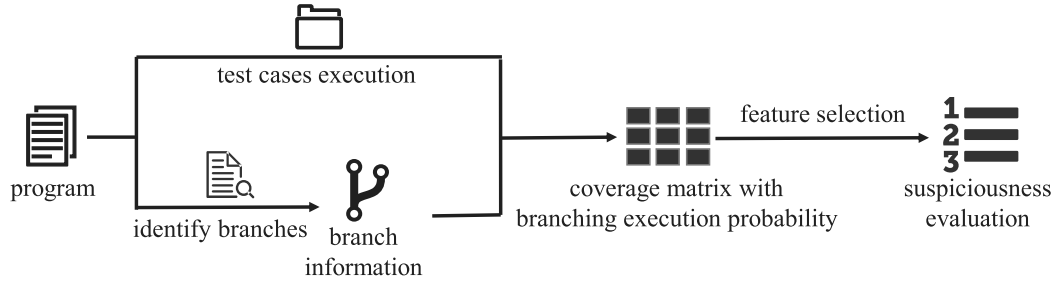


Fig. 2. Overview architecture of Feature-FL.

B. Define Branching Execution Probability

Branching structure is the basic structure that is widely used in program design and implementation [38], [44], which indicates that branching is the basic program behavior. The execution of a specific branch depends directly on the required condition satisfaction of this branch. For describing the branching behavior more accurately, we introduce a concept of branching execution probability. Informally, for a statement s that is inside a branching module b , the branch execution probability of s is defined as the ratio of the execution times of s to the execution times of b . For instance, suppose we have a branching module, which contains two mutually exclusive branches: one true branch and one false branch. And the branching module is wrapped with a *for* loop. For a concrete test case, the times of the *for* loop is set to 10, so the whole branching module is executed for 10 times. We further suppose the number of executing true branch is 7 (three times for false branch). In other words, a statement located in the true branch is executed seven times during the whole 10 times executions. Consequently, the branching execution probability of this statement can be calculated easily (i.e., $7/10=0.7$). On the contrary, the calculation of branching probability defined by theory tends to be tough in real scenarios. Compared to theoretical defined branching execution probability, ours mentioned above is easier to be calculated by the executions of the test suite. In addition, since the branching execution probability is usually bounded with a specific test case or a specific set of test cases, it helps to discover the particular or abnormal behaviors of each statement in a test case or a set of test cases. Hence, we can utilize the branching execution probability as the values of features to describe each statement's behaviors more precisely, and more importantly, it offers a manner of initializing the idea of using the feature selection methodology for fault localization.

C. Construct Feature Set

Next, we use branching execution probability to abstract program behavior as the values of features, that is, we should prepare the elaborate data for evaluating suspiciousness using feature selection. We will give the definition of the data formally. First, suppose that there is a program \mathcal{P} , consisting of a set of statements $\mathcal{S} = \{s_1, s_2, \dots, s_M\}$ and running against a test suite $\mathcal{T} = \{t_1, t_2, \dots, t_N\}$. Based on the program \mathcal{P} , our method defines a data set as the feature set $\text{FSet} = \{(f_i, l_i)\}_{i=1}^N$, where $f_i = [f_i^1, f_i^2, \dots, f_i^M]^T \in \mathbb{R}^{M \times 1}$ and l_i specifies the label of f_i

belonging to which class. f_i^j denotes the branching execution probability of the statement s_j in the execution of test case t_i . Based on FSet , we further define $F = [f_1, f_2, \dots, f_N] \in \mathbb{R}^{M \times N}$ to denote the input data matrix.

Thus, FSet constructs M features, among which each feature is a row of the input data matrix F , denoted as $f^j = [f_1^j, f_2^j, \dots, f_N^j] \in \mathbb{R}^{1 \times N}$, where $j \in \{1, 2, \dots, M\}$. f^j is the j th row of the input data matrix F , denoting the branching execution probabilities of the statement s_j in the executions of all test cases, i.e., it represents the behavioral feature of the statement s_j . In contrast to f^j , f_i is the i th column of the input data matrix, representing the overview branching features of M statements in the execution of test case t_i .

For an element f_i^j , three cases are used.

- 1) $f_i^j = 0$, if the statement s_j is not covered by the execution of the test case t_i .
- 2) $f_i^j = 1$, if the statement s_j which exists outside any branching module is covered by the execution of the test case t_i .
- 3) $f_i^j = bep$, if the statement s_j belongs to a certain branching module in the program. bep is the branching execution probability of s_j in the execution of the test case t_i .

Since f_i is based on the execution information of the test case t_i , we utilize the test results (*passing* or *failing*) of t_i to fix the label of f_i , showing that f_i is for which class. Therefore, two cases are used for l_i .

- 1) $l_i = 0$, if t_i is a *passing* test case.
- 2) $l_i = 1$, if t_i is a *failing* test case.

By this way, we successfully construct a feature set to depict branching behavior of a program from the feature view.

D. Calculate Branching Execution Probability

To facilitate the understanding and use of the feature set FSet , this step shows how to calculate the branching execution probability of the statements located in branching modules with the feature set FSet . Specifically, we demonstrate the calculation of branching execution probability along with the elements which are already defined in the above feature set.

Suppose that the statement s_j belongs to the module of a certain branching statement s_j^{if} , and is executed num_i times by a test case t_i . Furthermore, let $num_i^{if(j)}$ be the execution times of s_j^{if} in the test case t_i . Finally, we define the branching

```

1: /* Return nonzero when at end of file on input. */
2: local int input_eof()
3: {
4:     if (!decompress || last_member)
5:         return 1;
6:     if (inptr == insize)
7:     {
8:         if (insize != INBUFSIZ || fill_inbuf(1) == EOF)
9:             return 1;
10:        /* Unget the char that fill_inbuf got. */
11:        inptr = 0;
12:    }
13:    return 0;
14: }

```

Fig. 3. A program segment from `gzip`.

execution probability of the statement s_j in the test case t_i , i.e., we calculate the value of f_i^j in the feature set $FSet$ as follows:

$$f_i^j = \frac{num_i}{num_i^{if(j)}}. \quad (5)$$

In reality, it is common for a program to include nested branching structures, e.g., the program segment from `gzip`² as shown in Fig. 3. However, (5) does not take *nested* branching structures into consideration. For instance, suppose there is a sub-branch in the true branch of a branching module, the branching execution probabilities of the statements in the true branch will be the same value by only using (5). In other words, the behaviors of true sub-branch and the false sub-branch are not well depicted, and consequently, the feature selection algorithms could not capture the difference in the nested branch. Thus, we need an extension equation to include the information of nested branching structures. To this end, we extend our definition of execution branching probability based on (5). Since the statements of a nested branching module already owns an execution probability bep^{outer} from its outer branch before the execution jumps into those statements, the branching execution probability of each statement in this nested branching module is supposed to be multiplied by bep^{outer} . In this case, the branching execution probability of the statement s_j in t_i is defined as follows:

$$x_i^j = \frac{num_i \times bep^{outer}}{num_i^{if(j)}}. \quad (6)$$

Equation (6) is followed by the nature of the branch structure of the program. Although the branching execution probabilities will be low if there are too many nested branches, our extensive equation could also reflect the different behaviors of different branches of a specific branching module. As a reminder, in order to identify the branching modules belonging to which branch statement, we search for key words (e.g., `if` and `switch`) which stand for branching cases in the compiling phase. We implement the calculation of branching execution probability using recursive algorithm.

² is a free data compression utility, and we used it for the experimental study at Section IV.

E. Evaluate Suspiciousness Using Feature Selection

Feature selection is an effective approach using feature view to evaluate the relevance between features and the accuracy of the model. If we can abstract each statement's behavior as a feature, it means that we may use a feature perspective for fault localization to locate the root causes of failures. Following this intuition, we define the feature set $FSet$, which successfully depicts each statement's behaviors as features and associates those features with the test results using the labeling function. Thus, we construct a fault localization methodology based on feature selection, using the relevance of each feature with test results as the metric to measure the suspiciousness of each statement being faulty. Specifically, in the feature set $FSet$, the vector f^j represents the feature of s_j , showing the branching behavior of the statement s_j in the executions of all test cases. The labels $\{l_1, l_2, \dots, l_N\}$ denote the test results of all test cases. Thus, we use the feature vector of each statement as a feature representing their behaviors, and the labels of all test results as a reference feature referring to the relevance of each statement's feature. Feature-FL uses a feature selection criterion to evaluate the relevance of each statement's feature with the reference feature.

Therefore, Feature-FL is a family methodology of different criteria. Table I shows the six well-known effective feature selection criteria. Considering the popularity and effectiveness, we use the six feature selection criteria for our study. Since different feature selection methods share the same methodology structure (i.e., they have the same input and output), we take Pearson correlation coefficient [30] (i.e., the simplest but most effective one of the six feature selection criteria by our experimental study) as the representative to depict the methodology of Feature-FL.

As we have $\binom{d}{m}$ candidate Z 's out of X , it means that feature selection faces a very challenging problem: A combinatorial optimization problem. Since the main purpose of our feature selection criteria is to find highly irrelevant feature and does not concern the number of reduced features, we adopt the widely used heuristic strategy to alleviate this problem. Specifically, we compute a score for each feature independently, i.e., there are only $\binom{d}{1} = d$ candidates. Then, according to the common formulas of filtering feature selection criteria and feature set $FSet$, the score of the j th feature can be defined as follows:

$$score(f^j, l) = criterion(f^j, l) \quad (7)$$


where $criterion()$ refers to any formula in Table I. For example, if we use Pearson correlation coefficient as our criterion and compute a score of the j th feature, the equation is defined as follows:

$$score(f^j, l) = \rho_{f^j, l} = \frac{cov(f^j, l)}{\sigma f^j \sigma l} \quad (8)$$

where $cov()$ is the covariance and σ is the standard deviation. If we use Spearman's rank correlation coefficient, the equation is defined as follows:

$$score(f^j, l) = \rho_{rg_{f^j}, rg_l} = \frac{cov(rg_{f^j}, rg_l)}{\sigma rg_{f^j} \sigma rg_l} \quad (9)$$

Program									Bug information		
S1: read(nums); S2: read(length); S3: for(int i = 0; i < length;){ S4: if(i < 2){ S5: print(nums[i]); S6: i++;} S7: else{ print(nums[i]); S8: i+=1;}}									S8 is faulty. Correct form: i+=2;		
input	S1	S2	S3	S4	S5	S6	S7	S8	output	oracle	failure
[], 0	1	1	1	0	0	0	0	0	-	-	0
[1,2,3], 3	1	1	1(4)	1(3)	1(2)	1(2)	1	1	123	123	0
[2,3,4], 3	1	1	1(4)	1(3)	1(2)	1(2)	1	1	234	234	0
[1,2,3,4], 4	1	1	1(5)	1(4)	1(2)	1(2)	1(2)	1(2)	1234	123	1
[1,2,3,4,5], 5	1	1	1(6)	1(5)	1(2)	1(2)	1(3)	1(3)	12345	1235	1
[1,2,3,4,5,6], 6	1	1	1(7)	1(6)	1(2)	1(2)	1(4)	1(4)	123456	1235	1
suspiciousness	nan	nan	nan	0.45	0.45	0.45	0.45	0.45	Using Pearson		
rank	6	7	8	1	2	3	4	5	correlation coefficient		


 Using branching execution probabilities
 as the values of the features

input	S1	S2	S3	S4	S5	S6	S7	S8	output	oracle	failure
[], 0	1	1	1	0	0	0	0	0	-	-	0
[1,2,3], 3	1	1	1	1	0.67	0.67	0.33	0.33	123	123	0
[2,3,4], 3	1	1	1	1	0.67	0.67	0.33	0.33	234	234	0
[1,2,3,4], 4	1	1	1	1	0.5	0.5	0.5	0.5	1234	123	1
[1,2,3,4,5], 5	1	1	1	1	0.4	0.4	0.6	0.6	12345	1235	1
[1,2,3,4,5,6], 6	1	1	1	1	0.33	0.33	0.67	0.67	123456	1235	1
suspiciousness	nan	nan	nan	0.45	-0.08	-0.08	0.84	0.84	Using Pearson		
rank	6	7	8	3	4	5	1	2	correlation coefficient		

Fig. 4. An illustrative example. “1(*)” indicates the statement is executed * times by a test case. “-” means the output is null. “nan” means the suspiciousness value is not a valid number due to the 0/0 case.

where ρ denotes the Pearson correlation coefficient, rg_{fj} and rg_l are the rank variables that converted from variable f_j and l , respectively. In this way, all criteria could be applied to compute the score of each feature. The score of f_j shows the relevance of the j th feature (i.e., the behavioral feature of the statement s_j in all test cases) with the corresponding labels (i.e., the labels of test results). In other words, the score of f_j indicates the relevance of the statement s_j with the test results.

Thus, each statement finally obtains a score and a higher score indicates a stronger correlation between the statement and the failure. We use the score of each statement as their suspiciousness values of being faulty, and rank all the statements in descending order. Consequently, we construct a fault localization methodology using the feature view to locate faults, and name our approach as Feature-FL.

F. Motivation Example

From the above subsections, we fully depict the process of Feature-FL, and in this subsection, we intend to show an illustrative example about Feature-FL.

Fig. 4 shows a small but complete program. The fault of the program is located in “S8,” and the correct form is “i+=2; }”. The program has six test cases, and each test case has an input (the “input” column) and a predicted output (the “oracle”

column). The “output” column means the actual output of the program and the “failure” column means the results of each test case. The columns “S1” to “S8” record the coverage information. For example, S4 is executed three times by the second test case, so the value of the cell is “1(3),” in which 1 means “S4” is covered by the second test case and 3 in the brackets means the statement is executed three times. The values in brackets are used for the calculation of branching execution probability. The last two rows of the upper table in Fig. 4 show the suspiciousness values and the corresponding ranks of each statement by using the Pearson correlation coefficient as the criterion. Note that we use the information of covered (the value of 1) or not covered (the value of 0). We can observe that “S4” to “S8” have the same coverage information, so their suspiciousness values are the same. Furthermore, because the rank strategy we adopt is the statements are sorted in ascending order according to the line number when the statements have the same suspiciousness value, the faulty statement is ranked at last among the five statements.

From the upper table, we can observe that the effectiveness of the feature selection algorithm is limited by the coverage matrix that contains only binary information. Feature-FL uses branching execution probabilities as the values of the features as shown at the bottom of Fig. 4. For example, “S4” (e.g., “if” statement) is executed three times by the second test case, and the results of the expression are *True*, *True*, and *False*,

respectively. Thus, the “True” branch (e.g., “S5” and “S6”) is executed two times and the “False” branch (e.g., “S7” and “S8”) is executed for one time. Consequently, the value of “S5” and “S6” is 0.67 (2/3) and that of “S7” and “S8” is 0.33 (1/3). When using the Pearson correlation coefficient as our criterion, the rank of the faulty statement is 2, showing the improvement of Feature-FL.

In this example, we demonstrate the usefulness of the branching execution probability and feature selection algorithms from the practical view. In the next section, we conduct large-scale experiments to validate the effectiveness of Feature-FL.

IV. EXPERIMENTAL STUDY

A. Experimental Setup

To evaluate the localization effectiveness, the experiments should achieve two goals. The first one is to evaluate the effectiveness distribution of different feature selection criteria of Feature-FL, and recommend the most effective criterion of Feature-FL for debugging engineers. We select the six well-known effective filtering feature selection criteria to evaluate their localization effectiveness. As shown in Table I, the six feature selection criteria are Pearson correlation coefficient [30], Spearman’s rank correlation coefficient [36], Kendall rank correlation coefficient [37], mutual information [30], Fisher score [29], and Chi-squared test [39]. Another one is to evaluate the effectiveness of Feature-FL over the state-of-the-art techniques. We compare Feature-FL with four state-of-the-art localization techniques summarized by recent studies [16], [22]. As shown in (1), (2), (3), and (4), the four localization techniques are Dstar [15], Ochiai [40], Barinel [41], and Op2 [17].

Furthermore, this study [16] has shown that artificial faults and real faults have different effect on fault localization effectiveness, and recommended using real faults for fault localization. Even so, artificial faults including seeded faults simulate the real scenarios, and may still happen in practice. Thus, if a fault localization approach can perform well in both artificial faults and real ones, we can more safely conclude that this localization approach is effective in comparison to the case of using only real faults, because we verify this approach in more cases which have already happened (i.e., real faults) or are potential to happen (i.e., artificial faults).

Consequently, we choose the widely used real-life subject programs as the benchmarks to evaluate Feature-FL over the four localization techniques in two different contexts, i.e., artificial faults and real ones. In the context of artificial faults, we chose `Siemens`, `flex`, and `grep`, each of which has a number of faulty versions with seeded faults. `Siemens` developed by Siemens Corporate Research has seven programs performing a variety of tasks: `print_tokens` and `print_tokens2` perform lexical analysis, `replace` is a pattern recognizer, `schedule` and `schedule2` perform priority scheduling, `tcas` is an aircraft collision avoidance system, and `tot_info` computes statistics given input data. `flex` and `grep` are two typical UNIX utility programs, performing lexical analysis and pattern recognition, respectively. In the context of real faults, we used `space`, `sed`, `libtiff`, `python`, and `gzip`, each

of which contains a number of faulty versions with real faults. `space` was first written by the European Space Agency and function as an interpreter for an array definition language (ADL). `sed` and `gzip` are also the typical UNIX utility programs, conducting stream editing and data compression, respectively. `libtiff` is a free and open-source library for reading and writing TIFF (Tagged Image File Format) graphics files. `python` is a widely used general-purpose, high-level programming language. We also use the widely used programs `Chart`, `Closure`, `Lang`, `Math`, `Mockito`, and `Time`.

Table II summarizes the information of the subject programs, including subject names (column “Program”), numbers of faulty versions used (column “Versions”), lines of code (column “Loc”), numbers of test cases (column “Test”), fault types (column “Type”), as well as the functional description (column “Description”). We merge the results of `print_tokens` and `print_tokens2`, `schedule` and `schedule2` as each two have similar structures and functionalities. All artificial faults and `space` are from SIR³; `libtiff`, `python`, and `gzip` are collected from ManyBugs⁴; the others are from Defects4J⁵.

Our study implements all the experiments on a 64-bit Linux server with 16 Intel(R) Xeon CPUs and 128 G RAM. The operating system is Ubuntu 16.04.3.

B. Evaluation Metrics

We adopt the following metrics for evaluating the performance of a fault localization.

- **EXAM**: The *EXAM* metric [16], which is the most popular metric, is the percentage of statements to be examined before finding the *first* actual faulty statement. Specifically, *EXAM* is n/N , where n is the rank of the *first* faulty statement in the ranking list of all statements, and N is the total number of all statements. A lower value of *EXAM* means examining less code for locating the faulty statement, showing better localization performance.
- **Number of Top-K**: It is the number of buggy versions with at least one faulty statement that is within the first K position of rank list by a fault localization technique.
- **Relative Improvement**: Given two fault localization techniques, namely, FL1 and FL2, relative improvement (referred as *RImp*) [45] is to compare the total number of statements that need to be examined to find *all faults* using FL1 versus the number that need to be examined by using FL2. A lower value of *RImp* means that FL1 examines less code to locate the fault in comparison to FL2, showing higher localization effectiveness over FL2.
- **Wilcoxon-signed-rank test**: It is a metric that could evaluate the effectiveness of a fault localization method over others from the statistical perspective.

Specifically, when several statements have the same suspiciousness value, we adopt the statement order-based strategy [46] that we sort the statements in ascending order according to the line number.

³[Online]. Available: <http://sir.unl.edu/portal/index.php>

⁴[Online]. Available: <http://repairbenchmarks.cs.umass.edu/ManyBugs/>

⁵[Online]. Available: <https://github.com/rjust/defects4j>

TABLE II
SUMMARY OF SUBJECT PROGRAMS

Program	Versions	Loc	Test	Type	Description
print_tokens (2 ver.)	15	570/726	316/393	Seeded	Lexical analyzer
replace	27	564	395	Seeded	Pattern recognition
schedule (2 ver.)	16	374/412	228/235	Seeded	Priority scheduler
tcas	29	173	83	Seeded	Altitude separation
tot_info	18	565	194	Seeded	Information measure
flex	53	10,459	567	Seeded	Lexical analyzer
grep	21	10,068	809	Seeded	Pattern recognition
Average	26	3,243	376		
space	35	6,199	4,333	Real	ADL interpreter
sed	29	14,427	370	Real	Stream editor
gzip	5	491,000	12	Real	Data compression
libtiff	12	77,000	78	Real	Image processing
python	8	407,000	355	Real	General-purpose language
Chart	26	96,000	2,205	Real	Java chart library
Closure	133	90,000	7,927	Real	Closure compiler
Math	106	85,000	3,602	Real	Apache commons-math
Lang	65	22,000	2,245	Real	Apache commons-lang
Time	27	28,000	4,130	Real	Standard date and time library
Mokito	38	67,000	1,075	Real	Mocking framework for Java
Average	44	125,784	2394		

C. Comparison Among Feature Selection Criteria for Feature-FL

EXAM Distribution: Fig. 5 displays the *EXAM* distribution by using six feature selection for fault localization. The legend of this figure is shown at the last of this figure. A point in Fig. 5 means that when a *EXAM* value is reached, the percentage of faulty versions has located their faults. From Fig. 5, we can notice that the curves of Pearson correlation coefficient, Spearman's rank correlation coefficient, and Kendall rank correlation coefficient are always above the other three feature selection criteria (i.e., mutual information, Fisher score, and Chi-squared test), and mutual information seems to be less effective on fault localization than other criteria in most programs. It means that Pearson correlation coefficient, Spearman's rank correlation coefficient, and Kendall rank correlation coefficient outperform the other three feature selection criteria in each program. Therefore, under the *EXAM* distribution, we take Pearson correlation coefficient, Spearman's rank correlation coefficient, and Kendall rank correlation coefficient as the candidate effective criteria for *Feature-FL*. In addition, Fig. 6 presents the *EXAM* distribution of each feature selection criterion in the context of artificial faults and real faults, respectively. As we can see, the curves of Pearson correlation coefficient, Spearman's rank correlation coefficient, and Kendall rank correlation coefficient are also always above the other three criteria in both cases of artificial faults and real faults, which indicates the conclusion is consistent with that of the single subject program comparison.

RImp Distribution: In order to find the most effective feature selection criterion in more detail, we use *RImp* to evaluate Pearson correlation coefficient [30] over Spearman's rank correlation coefficient, Kendall rank correlation coefficient [37], mutual information [30], Fisher score [29], and Chi-squared test. Fig. 7 displays *RImp* of Pearson correlation coefficient [30] over the other five feature selection criteria in 18 faulty programs. The cells below the columns show the specific *RImp* values of Pearson correlation coefficient [30] acquired. Take *gzip* in mutual

information [30] as an example. The *RImp* is 67.32%, meaning that Pearson correlation coefficient [30] takes up 67.32% of examined code of mutual information [30] to locate all faults in *gzip*. In other words, Pearson correlation coefficient [30] obtains a *saving* of 32.68% ($100\% - 67.32\% = 32.68\%$) over mutual information [30] in *gzip*.

As we can see from the cells below the columns, almost all the specific *RImp* values are less than or equal to 100%. Notice that the *RImp* values over Spearman's rank correlation coefficient and Kendall rank correlation coefficient [37] are around 100%, which means these three feature selection criteria (i.e., Pearson correlation coefficient [30], Spearman's rank correlation coefficient, and Kendall rank correlation coefficient [37]) have similar effectiveness in all programs, i.e., in different contexts of real faults and artificial faults. Thus, according to the *RImp* distribution, we can also take Pearson correlation coefficient [30], Spearman's rank correlation coefficient, and Kendall rank correlation coefficient [37] as candidate effective feature selection criteria of *Feature-FL*.

Number of Top-K: From the *EXAM* distribution and *RImp* distribution, we can observe that the Pearson correlation coefficient, Spearman's rank correlation coefficient, and Kendall rank correlation coefficient are the most effective criteria among the six criteria. However, in this study [47], they reported that they have clear limitations to more experienced developers and simpler code when applying the *EXAM* and *RImp* as the metrics. An alternative metric named the number of Top-K is better studied in Ref. [48]. For a concrete example, when comparing approach A with approach B and A performs slightly better than B (e.g., ranking the faulty statement at position 2 instead of 5), but in one case A ranks another bug at position 500 and B ranks it at position 100, *RImp* metric favors B over A. But under the Top-3 values, approach A outperforms approach B. In a previous study, most respondents view fault localization as successful only if it can localize a bug in the Top-5 positions from a practical perspective [49]. So we should apply multidimensional

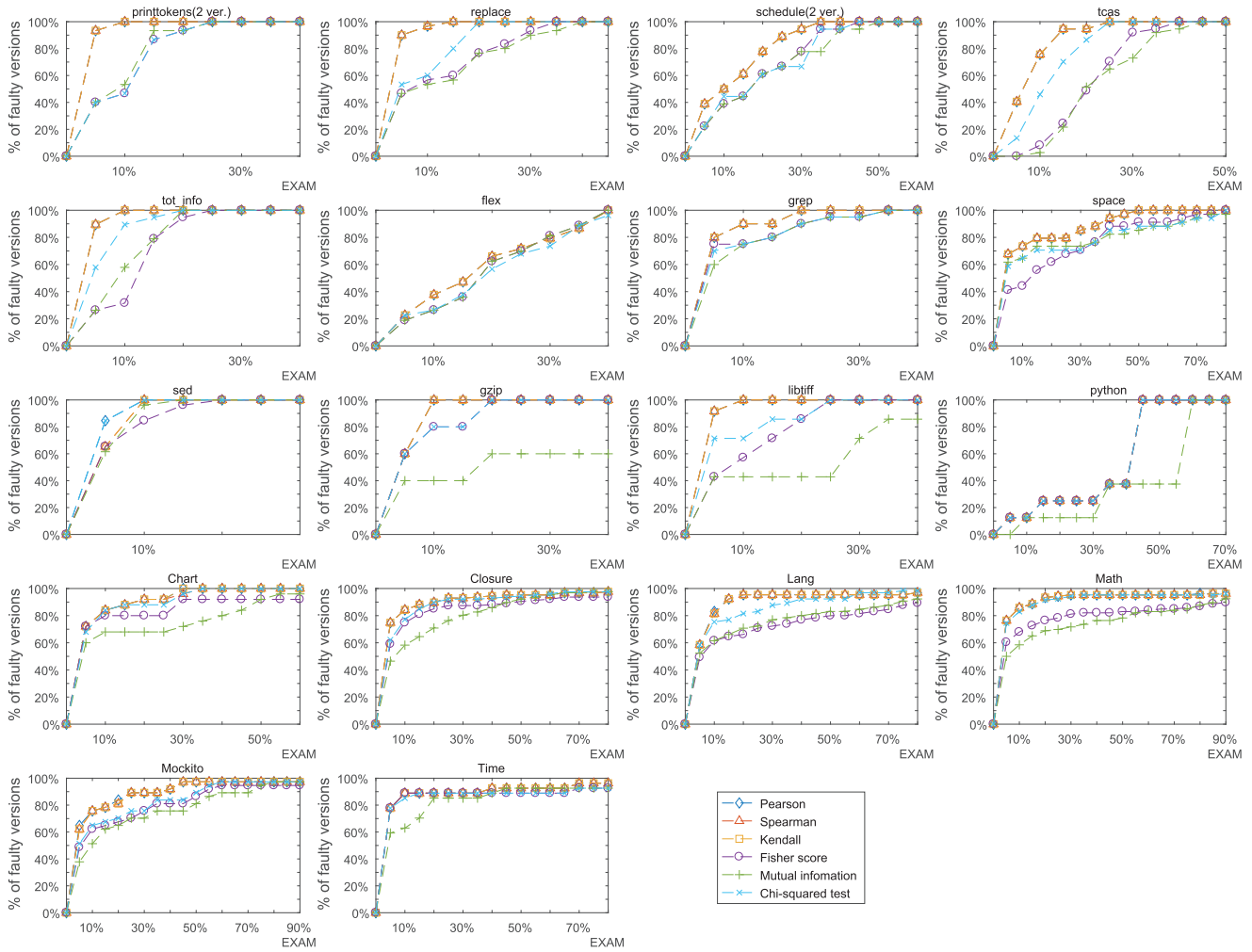


Fig. 5. EXAM distribution of six feature selection criteria in each program.

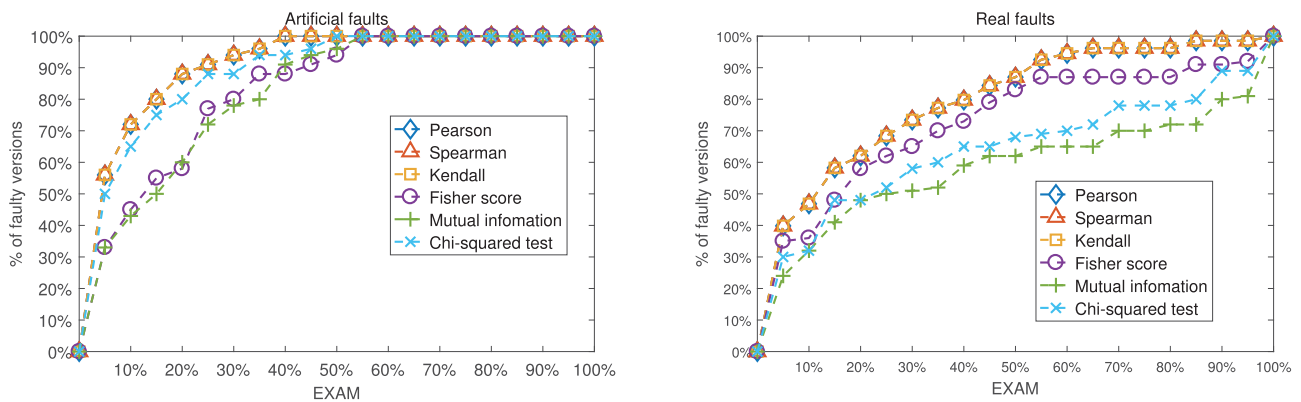


Fig. 6. EXAM distribution of six feature selection criteria in artificial faults and real faults.

evaluation metrics to demonstrate the effectiveness of a specific fault localization approach. Following the prior work [10], [28], [50], we assign K with the value of 1, 3, and 5 for our evaluation.

Tables III and IV show the results of Top-1, Top-3, and Top-5 for each program of artificial and real faults, respectively. The “summation” rows in Tables III and IV list the sum of Top- K

metrics of all listed program. From Tables III and IV, we can observe that the Pearson correlation coefficient, Spearman’s rank correlation coefficient, and Kendall rank correlation coefficient can locate the most bugs under Top-1, Top-3, and Top-5 metrics. Notice that the numbers of Top- K in terms of `flex` in Table III and that of Top- K in terms of `space` and `python` in Table IV

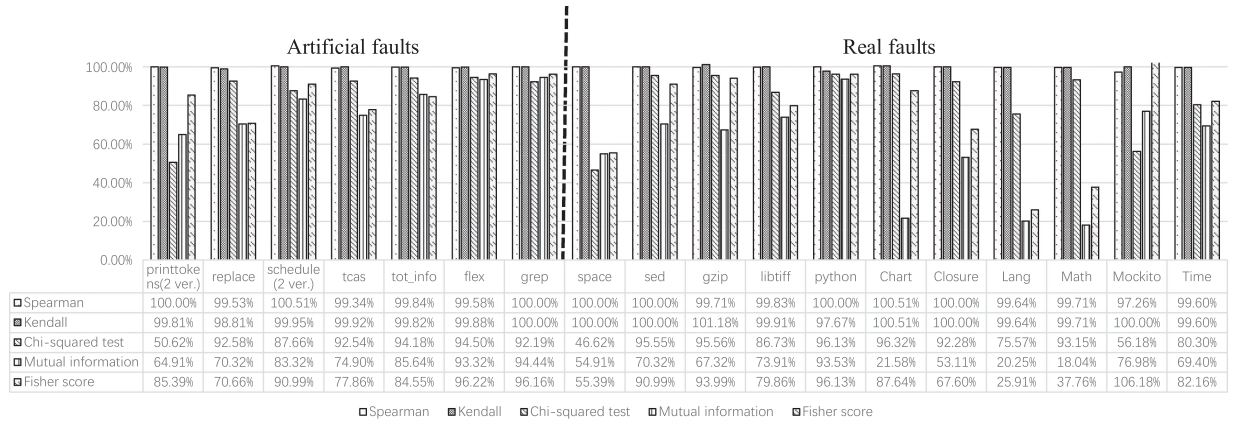


Fig. 7. *RImp* distribution in each program (pearson vs. other five feature selection criteria).

TABLE III
COMPARISON AMONG FEATURE SELECTION CRITERIA UNDER THE NUMBER OF TOP-1, TOP-3, AND TOP-5 METRICS OF EACH PROGRAM WITH ARTIFICIAL FAULTS

		Pearson	Spearman	Kendall	chisquared test	mutual information	fisher score
printtokens	Top-1	3	3	3	2	2	3
	Top-3	7	7	7	7	6	7
	Top-5	10	10	10	9	9	9
replace	Top-1	3	3	3	2	3	3
	Top-3	7	7	7	4	6	7
	Top-5	11	11	11	8	10	10
schedule	Top-1	1	1	1	0	1	1
	Top-3	4	4	4	2	2	3
	Top-5	4	4	4	3	3	4
tcas	Top-1	2	2	2	0	1	1
	Top-3	5	5	5	3	3	4
	Top-5	9	9	9	5	6	8
tot_info	Top-1	2	2	2	0	1	2
	Top-3	3	3	3	2	1	2
	Top-5	5	5	5	3	3	4
flex	Top-1	0	0	0	0	0	0
	Top-3	0	0	0	0	0	0
	Top-5	0	0	0	0	0	0
grep	Top-1	3	3	3	1	0	1
	Top-3	3	3	3	1	1	1
	Top-5	3	3	3	2	1	2
summation	Top-1	14	14	14	5	8	11
	Top-3	29	29	29	20	19	24
	Top-5	42	42	42	30	32	37

The bold numbers indicate the corresponding techniques with the most number of bugs located.

is 0, which means the effectiveness of fault localization from the practical perspective is heavily limited and still fault localization has a long way to go.

Statistical Comparison: We compare the six feature selection criteria using the three metrics (i.e., *EXAM* distribution, number of Top-K, and *RImp* distribution) in detail. For a further rigorous evaluation, we compare the six feature selection criteria by using the paired Wilcoxon-signed-rank test [51] with a Bonferroni correction [52], which is a nonparametric statistical hypothesis test for testing the differences between pairs of measurements $F(x)$ and $G(y)$, which do not follow a normal distribution. It makes use of the sign and the magnitude of the rank of the differences between $F(x)$ and $G(y)$. At the given significant level σ , we can use both two-tailed and one-tailed p -value to obtain a conclusion.

For the two-tailed p -value, if $p \geq \sigma$, the null hypothesis H_0 that $F(x)$ and $G(y)$ are not significantly different is accepted; otherwise, the alternative hypothesis H_1 that $F(x)$ and $G(y)$ are significantly different is accepted. For one-tailed p -value, there are two cases: the right-tailed case and the left-tailed case. In the

TABLE IV
COMPARISON AMONG FEATURE SELECTION CRITERIA UNDER THE NUMBER OF TOP-1, TOP-3, AND TOP-5 METRICS OF EACH PROGRAM WITH REAL FAULTS

		Pearson	Spearman	Kendall	chisquared test	mutual information	fisher score
space	Top-1	0	0	0	0	0	0
	Top-3	0	0	0	0	0	0
	Top-5	0	0	0	0	0	0
sed	Top-1	0	0	0	0	0	0
	Top-3	4	4	4	2	2	3
	Top-5	5	5	5	3	3	4
gzip	Top-1	0	0	0	0	0	0
	Top-3	0	0	0	0	0	0
	Top-5	1	1	1	0	0	1
libtiff	Top-1	2	2	2	1	1	1
	Top-3	2	2	2	1	1	2
	Top-5	4	4	4	3	2	3
python	Top-1	0	0	0	0	0	0
	Top-3	0	0	0	0	0	0
	Top-5	0	0	0	0	0	0
Chart	Top-1	3	3	3	3	2	3
	Top-3	6	6	6	5	4	5
	Top-5	9	9	9	7	6	7
Closure	Top-1	5	5	5	3	1	5
	Top-3	14	13	13	7	6	16
	Top-5	18	18	18	9	10	18
Lang	Top-1	5	5	5	5	4	2
	Top-3	17	17	17	16	15	13
	Top-5	24	24	24	22	21	19
Math	Top-1	16	16	16	18	11	10
	Top-3	32	32	32	28	19	20
	Top-5	39	39	39	35	23	28
Mockito	Top-1	6	6	6	4	5	6
	Top-3	11	11	11	5	7	8
	Top-5	12	12	12	7	8	10
Time	Top-1	2	2	2	2	2	3
	Top-3	10	10	10	5	9	10
	Top-5	10	10	10	8	9	11
summation	Top-1	39	39	39	36	26	30
	Top-3	96	95	95	69	63	77
	Top-5	122	122	122	94	84	101

The bold numbers indicate the corresponding techniques with the most number of bugs located.

right-tailed case, if $p \geq \sigma$, H_0 that $F(x)$ does not significantly tend to be greater than the $G(y)$ is accepted; otherwise, H_1 that $F(x)$ significantly tends to be greater than the $G(y)$ is accepted. And in the left-tailed case, if $p \geq \sigma$, H_0 that $F(x)$ does not significantly tend to be less than the $G(y)$ is accepted; otherwise, H_1 that $F(x)$ significantly tends to be less than the $G(y)$ is accepted [53].

The experiments performed five paired Wilcoxon-signed-rank test: The localization effectiveness of Pearson correlation coefficient [30] vs. that of each of the other five feature selection criteria. Each test uses both the two-tailed and one-tailed checking at the σ level of 0.05. Given a program, we use the list of the *EXAM* in all faulty versions of the program for using

TABLE V
STATISTICAL COMPARISON OF PEARSON AND THE OTHER FIVE FEATURE SELECTION CRITERIA

Program	Comparison	2-tailed	1-tailed(right)	1-tailed(left)	Conclusion
printtokens(2 ver.)	Pearson v.s. Spearman	1.00E+00	1.00E+00	1.00E+00	SIMILAR
	Pearson v.s. Kendall	1.00E+00	1.00E+00	1.00E+00	SIMILAR
	Pearson v.s. Chi-squared test	6.32E-01	5.02E-01	7.87E-01	SIMILAR
	Pearson v.s. Mutual information	1.67E-02	8.99E-01	9.22E-03	BETTER
	Pearson v.s. Fisher score	5.77E-01	4.33E-01	5.02E-01	SIMILAR
replace	Pearson v.s. Spearman	1.00E+00	1.00E+00	1.00E+00	SIMILAR
	Pearson v.s. Kendall	1.00E+00	1.00E+00	1.00E+00	SIMILAR
	Pearson v.s. Chi-squared test	8.98E-01	8.52E-01	7.42E-01	SIMILAR
	Pearson v.s. Mutual information	1.98E-01	5.19E-01	2.85E-01	SIMILAR
	Pearson v.s. Fisher score	5.54E-01	4.17E-01	5.49E-01	SIMILAR
schedule(2 ver.)	Pearson v.s. Spearman	1.00E+00	1.00E+00	1.00E+00	SIMILAR
	Pearson v.s. Kendall	1.00E+00	1.00E+00	1.00E+00	SIMILAR
	Pearson v.s. Chi-squared test	9.88E-01	8.98E-01	9.57E-01	SIMILAR
	Pearson v.s. Mutual information	8.97E-01	7.51E-01	1.17E-01	SIMILAR
	Pearson v.s. Fisher score	7.54E-01	4.18E-01	8.58E-01	SIMILAR
tcas	Pearson v.s. Spearman	1.00E+00	1.00E+00	1.00E+00	SIMILAR
	Pearson v.s. Kendall	1.00E+00	1.00E+00	1.00E+00	SIMILAR
	Pearson v.s. Chi-squared test	8.89E-01	6.33E-01	5.48E-01	SIMILAR
	Pearson v.s. Mutual information	2.89E-01	8.19E-01	3.80E-01	SIMILAR
	Pearson v.s. Fisher score	1.87E-01	6.32E-01	1.84E-01	SIMILAR
tot_info	Pearson v.s. Spearman	1.00E+00	1.00E+00	1.00E+00	SIMILAR
	Pearson v.s. Kendall	1.00E+00	1.00E+00	1.00E+00	SIMILAR
	Pearson v.s. Chi-squared test	9.89E-01	9.84E-01	9.89E-01	SIMILAR
	Pearson v.s. Mutual information	9.10E-01	3.91E-01	9.80E-01	SIMILAR
	Pearson v.s. Fisher score	4.57E-01	2.18E-01	3.64E-01	SIMILAR
flex	Pearson v.s. Spearman	1.00E+00	1.00E+00	1.00E+00	SIMILAR
	Pearson v.s. Kendall	1.00E+00	1.00E+00	1.00E+00	SIMILAR
	Pearson v.s. Chi-squared test	9.65E-01	9.22E-01	9.44E-01	SIMILAR
	Pearson v.s. Mutual information	6.05E-01	3.04E-01	6.00E-01	SIMILAR
	Pearson v.s. Fisher score	8.83E-01	1.32E-01	9.64E-01	SIMILAR
grep	Pearson v.s. Spearman	1.00E+00	1.00E+00	1.00E+00	SIMILAR
	Pearson v.s. Kendall	1.00E+00	1.00E+00	1.00E+00	SIMILAR
	Pearson v.s. Chi-squared test	4.79E-01	8.55E-01	1.78E-01	SIMILAR
	Pearson v.s. Mutual information	4.02E-01	7.12E-01	4.05E-01	SIMILAR
	Pearson v.s. Fisher score	9.83E-01	1.92E-01	8.44E-01	SIMILAR
gzip	Pearson v.s. Spearman	1.00E+00	1.00E+00	1.00E+00	SIMILAR
	Pearson v.s. Kendall	1.00E+00	1.00E+00	1.00E+00	SIMILAR
	Pearson v.s. Chi-squared test	1.00E+00	6.05E-01	6.05E-01	SIMILAR
	Pearson v.s. Mutual information	2.25E-01	9.11E-01	1.40E-01	SIMILAR
	Pearson v.s. Fisher score	6.55E-01	8.14E-01	5.00E-01	SIMILAR
libtiff	Pearson v.s. Spearman	1.00E+00	1.00E+00	1.00E+00	SIMILAR
	Pearson v.s. Kendall	1.00E+00	1.00E+00	1.00E+00	SIMILAR
	Pearson v.s. Chi-squared test	6.55E-01	5.00E-01	8.14E-01	SIMILAR
	Pearson v.s. Mutual information	1.80E-02	9.93E-01	1.12E-02	BETTER
	Pearson v.s. Fisher score	6.55E-01	5.00E-01	8.14E-01	SIMILAR
python	Pearson v.s. Spearman	1.00E+00	1.00E+00	1.00E+00	SIMILAR
	Pearson v.s. Kendall	1.00E+00	1.00E+00	1.00E+00	SIMILAR
	Pearson v.s. Chi-squared test	1.80E-01	9.63E-01	1.86E-01	SIMILAR
	Pearson v.s. Mutual information	6.79E-02	9.78E-01	5.02E-02	SIMILAR
	Pearson v.s. Fisher score	4.65E-01	8.19E-01	2.92E-01	SIMILAR
space	Pearson v.s. Spearman	1.00E+00	1.00E+00	1.00E+00	SIMILAR
	Pearson v.s. Kendall	1.00E+00	1.00E+00	1.00E+00	SIMILAR
	Pearson v.s. Chi-squared test	4.04E-04	1.00E+00	2.10E-04	BETTER
	Pearson v.s. Mutual information	7.96E-03	9.96E-01	4.10E-03	BETTER
	Pearson v.s. Fisher score	5.87E-03	9.71E-01	3.01E-02	BETTER
Defects4J	Pearson v.s. Spearman	8.82E-02	9.56E-01	4.49E-02	BETTER
	Pearson v.s. Kendall	8.82E-02	9.56E-01	4.49E-02	BETTER
	Pearson v.s. Chi-squared test	8.03E-20	1.00E+00	4.03E-20	BETTER
	Pearson v.s. Mutual information	1.24E-26	1.00E+00	6.23E-27	BETTER
	Pearson v.s. Fisher score	2.54E-12	1.00E+00	1.27E-12	BETTER
total	Pearson v.s. Spearman	1.00E+00	1.00E+00	1.00E+00	SIMILAR
	Pearson v.s. Kendall	9.99E-01	9.89E-01	9.95E-01	SIMILAR
	Pearson v.s. Chi-squared test	8.75E-01	8.65E-01	8.10E-01	SIMILAR
	Pearson v.s. Mutual information	7.06E-01	8.96E-01	8.11E-01	SIMILAR
	Pearson v.s. Fisher score	8.87E-01	7.78E-01	7.21E-01	SIMILAR

Pearson correlation coefficient [30] as the list of measurements of $F(x)$, while the list of measurements of $G(y)$ is the list of $EXAM$ for using one of the other five feature selection criteria. Hence, in the two-tailed test, Pearson correlation coefficient [30] has SIMILAR effectiveness as the compared feature selection criterion when H_0 is accepted at the significant level of 0.05. And in the one-tailed test (right), Pearson correlation coefficient [30] has WORSE effectiveness than the compared feature selection

criterion when H_1 is accepted at the significant level of 0.05. Finally, in the one-tailed test (left), Pearson correlation coefficient [30] has BETTER effectiveness than the compared feature selection criterion when H_1 is accepted at the significant level of 0.05.

Table V shows the statistical results of Pearson correlation coefficient over each of the other five feature selection criteria in all programs, i.e., in different contexts of real faults and artificial

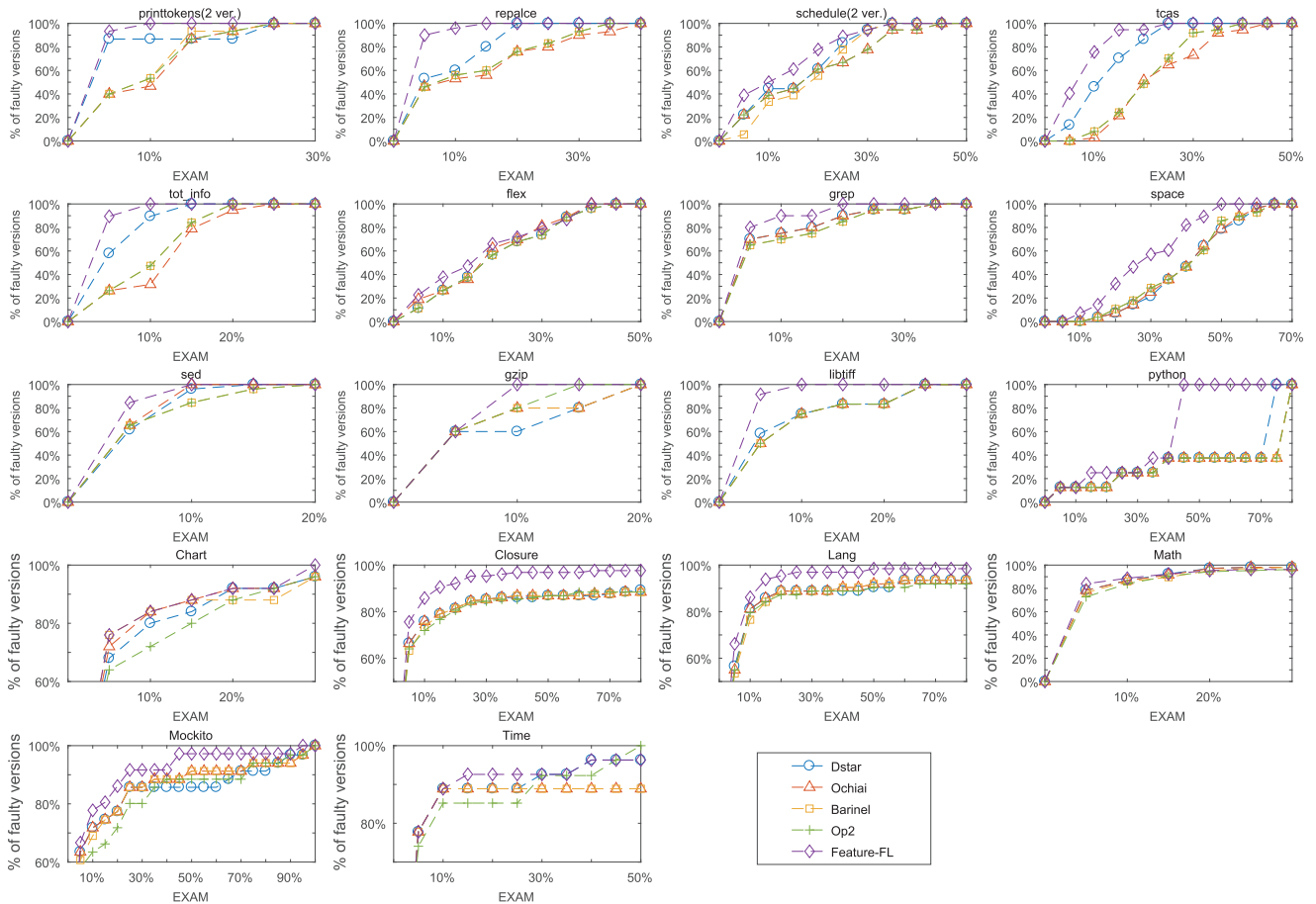


Fig. 8. *EXAM* distribution in each program.

faults. We view the programs in Defects4J as a whole for this statistical comparison for convenience of presentation. We can observe that Pearson correlation coefficient obtains SIMILAR effectiveness over Spearman's rank correlation coefficient and Kendall rank correlation coefficient in most cases. Furthermore, Pearson correlation coefficient obtains BETTER or SIMILAR results over mutual information, Fisher score and Chi-squared test. Based on the statistical results, we can also identify that Pearson correlation coefficient, Spearman's rank correlation coefficient and Kendall rank correlation coefficient are the candidate effective feature selection criteria for Feature-FL.

Summary: Based on the above results (i.e., *EXAM* distribution, number of Top-K, *RImp* distribution, and statistical comparison), we can safely conclude that Pearson correlation coefficient, Spearman's rank correlation coefficient and Kendall rank correlation coefficient are the most effective feature selection criteria in fault localization in the two contexts (i.e., artificial faults and real ones), recommending using the three feature selection criteria as the candidates of applying Feature-FL in practice.

D. Comparison Between Feature-FL and SFL

Since Pearson correlation coefficient is the simplest structure and one of the most effective feature selection criteria, we use

Pearson correlation coefficient as the representative to be compared with the four state-of-the-art techniques (i.e., Dstar [15], Ochiai [40], Barinel [41], and Op2 [17]) in the two contexts (i.e., artificial faults and real ones).

EXAM Distribution: Fig. 8 displays the *EXAM* distribution of each program by using each localization technique, respectively. The first seven subfigures are those programs with artificial faults while the subsequent 11 subfigures represent those programs with real ones. The legend of these subfigures is shown at the last of the figure. Apparently, if a curve of a fault localization technique reaches the 100% of faulty versions more quickly, this localization technique is more effective. As shown in Fig. 8, the curves of the other fault localization techniques are always beneath those of Feature-FL in each program. It means that Feature-FL outperforms the other four state-of-the-art techniques in each program. Furthermore, Fig. 9 summarizes the *EXAM* distribution of each fault localization technique in the context of artificial faults and real faults, respectively. We can observe that Feature-FL reaches 100% of faulty versions much faster than the other four localization techniques in both cases of artificial faults and real faults. Thus, Feature-FL significantly outperforms the four state-of-the-art fault localization techniques.

RImp Distribution: For detailed improvement in each program, we use *RImp* to evaluate Feature-FL over the four

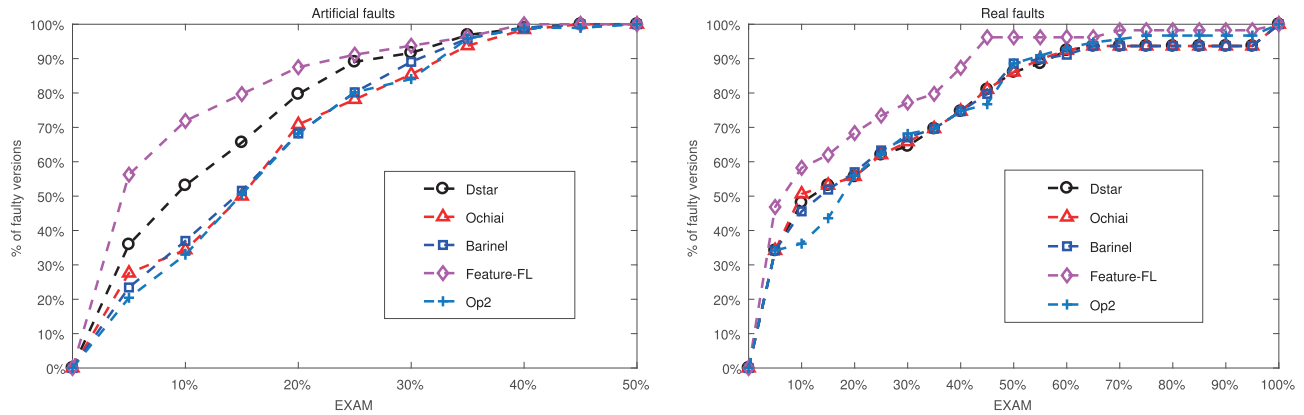


Fig. 9. EXAM distribution of Feature-FL and FL in artificial faults and real faults.

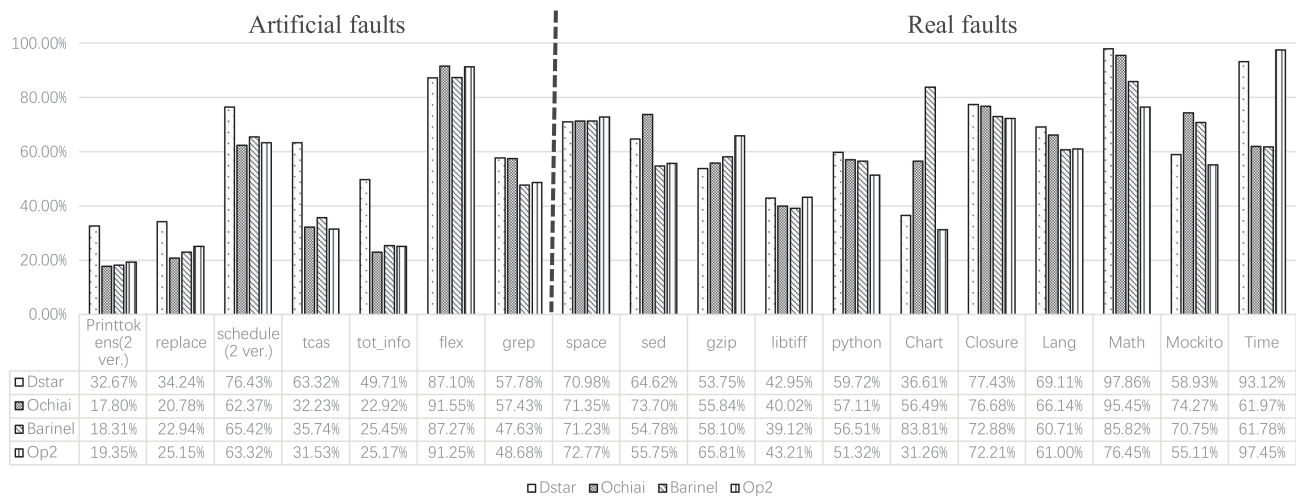


Fig. 10. RImp distribution in each program.

state-of-the-art localization techniques. Fig. 10 displays *RImp* of Feature-FL over Dstar, Ochiai, Barinel, and Op2 in all faulty versions of each program. The cells below the columns show the specific *RImp* values of Feature-FL acquired. Take `print_tokens(2 ver.)` in Dstar as an example. The *RImp* is 32.67%, meaning that Feature-FL takes up 32.67% of examined code of Dstar to locate all faults in `print_tokens(2 ver.)`. In other words, Feature-FL obtains a *saving* of 67.33% ($100\% - 32.67\% = 67.33\%$) over Dstar in `print_tokens(2 ver.)`.

In the context of artificial faults, Feature-FL examines a range from 32.67% to 87.10% of the examined code of Dstar (12.90%–67.33% *saving*), 17.80%–91.55% of the examined code of Ochiai (8.45% to 82.20% *saving*), 18.31%–87.27% of the examined code of Barinel (12.73%–81.69% *saving*), and 19.35%–91.25% off the examined code of Op2 (8.75%–80.65% *saving*). In the context of real faults, Feature-FL examines a range from 36.61% to 97.86% of the examined code of Dstar (2.14%–63.39% *saving*), 40.02%–95.45% of the examined code of Ochiai (4.55%–59.98% *saving*), 39.12%–85.82% of the examined code of Barinel (14.18%–60.88% *saving*), and

31.26%–97.45% off the examined code of Op2 (2.55%–68.74% *saving*).

In summary, Feature-FL obtains an average *saving* of 42.68% in Dstar, 56.42% in Ochiai, 56.75% in Barinel, and 56.51% in Op2 in case of artificial faults while Feature-FL acquires an average *saving* of 34.08% in Dstar, 33.73% in Ochiai, 34.96% in Barinel, and 37.97% in Op2 in case of real faults. Thus, Feature-FL significantly improves fault localization effectiveness in different contexts, i.e., artificial faults and real faults.

Number of Top-K: Like the comparison among the feature selection criteria, we compare Feature-FL with the four state-of-the-art fault localization techniques under the metrics of number of Top-1, Top-3, and Top-5. From Tables VI and VII, we can observe that Feature-FL can locate more bugs than the other four SFL techniques in most programs under the Top-1, Top-3, and Top-5 metrics. Specifically, for artificial faults, Feature-FL can obtain the 15 optimal number of bugs totally out of 21 comparisons, i.e., 7 (programs with artificial faults) \times 3 (metrics for each program) = 21 comparisons, the values of Dstar, Ochiai, Barinel, and Op2 are only five, four, four,

TABLE VI
COMPARISON BETWEEN Feature-FL AND FOUR FL TECHNIQUES UNDER
THE NUMBER OF TOP-1, TOP-3, AND TOP-5 METRICS OF EACH PROGRAM
WITH ARTIFICIAL FAULTS

		Dstar	Ochiai	Barinel	Op2	Feature-FL
printtokens	Top-1	3	3	0	1	3
	Top-3	8	3	2	2	7
	Top-5	9	3	2	2	10
replace	Top-1	2	5	3	3	3
	Top-3	5	8	7	7	7
	Top-5	7	9	8	8	11
schedule	Top-1	0	0	0	0	1
	Top-3	0	0	0	0	4
	Top-5	0	0	0	0	4
tcas	Top-1	1	0	0	0	2
	Top-3	2	0	0	0	5
	Top-5	3	0	0	0	9
tot_info	Top-1	1	1	1	1	2
	Top-3	1	2	1	1	3
	Top-5	4	3	3	3	5
flex	Top-1	0	0	0	0	0
	Top-3	0	0	0	0	0
	Top-5	0	0	0	0	0
grep	Top-1	3	3	3	3	3
	Top-3	3	3	3	3	3
	Top-5	4	4	4	4	3
summation	Top-1	10	12	7	8	14
	Top-3	19	16	13	13	29
	Top-5	27	19	17	17	42

The bold numbers indicate the corresponding techniques with the most number of bugs located.

TABLE VII
COMPARISON BETWEEN Feature-FL AND FOUR FL TECHNIQUES UNDER
THE NUMBER OF TOP-1, TOP-3, AND TOP-5 METRICS OF EACH PROGRAM
WITH REAL FAULTS

		Dstar	Ochiai	Barinel	Op2	Feature-FL
space	Top-1	0	0	0	0	0
	Top-3	0	0	0	0	0
	Top-5	0	0	0	0	0
sed	Top-1	0	0	0	0	0
	Top-3	0	0	0	0	4
	Top-5	0	0	0	0	5
gzip	Top-1	0	0	0	1	0
	Top-3	0	0	0	1	0
	Top-5	1	1	1	2	1
libtiff	Top-1	0	0	0	0	2
	Top-3	2	2	2	2	2
	Top-5	2	2	2	2	4
python	Top-1	0	0	0	0	0
	Top-3	0	0	0	0	0
	Top-5	0	0	0	0	0
Chart	Top-1	3	3	2	2	3
	Top-3	6	6	5	5	6
	Top-5	9	9	9	8	9
Closure	Top-1	6	5	5	8	5
	Top-3	13	13	10	13	14
	Top-5	17	18	17	19	18
Lang	Top-1	5	5	5	5	5
	Top-3	17	17	17	16	17
	Top-5	24	24	24	23	24
Math	Top-1	9	9	9	7	16
	Top-3	19	19	19	16	32
	Top-5	21	22	22	18	39
Mockito	Top-1	5	6	4	4	6
	Top-3	12	12	9	9	11
	Top-5	13	13	11	11	12
Time	Top-1	2	2	2	3	2
	Top-3	10	10	9	10	10
	Top-5	10	10	10	10	10
summation	Top-1	30	30	27	30	39
	Top-3	79	79	71	72	96
	Top-5	97	99	96	93	122

The bold numbers indicate the corresponding techniques with the most number of bugs located.

and four, respectively. For real faults, Feature-FL can get the 18 optimal number of bugs totally out of 33 comparisons [i.e., 11 (programs) \times 3 (metrics)], the values of Dstar, Ochiai, Barinel, and Op2 are 11, 12, 6, and 10, respectively. For the sed program in Table VII, the four SFL techniques could not locate any bug in the five positions, but Feature-FL can locate four and five bugs in the three positions and five positions, respectively, showing better performance of Feature-FL than the four SFL techniques.

Statistical Comparison: Although *RImp* can show more detailed improvement than the *EXAM* scores, the analysis using *RImp* still evaluates Feature-FL from the overview of the results, and may miss other detailed view of the results. For example, suppose that Feature-FL just obtains very higher effectiveness than Dstar in several faulty versions of a program, and, however, Dstar has moderately higher effectiveness in most faulty versions of the programs. The sheer high effectiveness of Feature-FL in the several faulty versions may make its *RImp* score lower than Dstar, showing that Feature-FL performs better than Dstar. However, in such case, we cannot conclude that Feature-FL performs better than Dstar.

Therefore, we further conduct a more scientific and rigorous method, that is, the paired Wilcoxon-signed-rank test to evaluate the effectiveness of Feature-FL over that of the other four fault localization techniques. The experiments performed one paired Wilcoxon-signed-rank test: The localization effectiveness of Feature-FL vs. that of each of the other four fault localization techniques. Each test uses both the two-tailed and one-tailed checking at the σ level of 0.05. Given a program, we use the list of the *EXAM* in all faulty versions of the program for using our approach Feature-FL as the list of measurements of $F(x)$, while the list of measurements of $G(y)$ is the list of *EXAM* for using one of the other four fault localization techniques. Hence, in the two-tailed test, Feature-FL has SIMILAR effectiveness as the compared fault localization technique when H_0 is accepted at the significant level of 0.05. And in the one-tailed test (right), Feature-FL has WORSE effectiveness than the compared fault localization technique when H_1 is accepted at the significant level of 0.05. Finally, in the one-tailed test (left), Feature-FL has BETTER effectiveness than the compared fault localization technique when H_1 is accepted at the significant level of 0.05.

Table VIII shows the statistical results of Feature-FL over each of four state-of-the-art fault localization techniques in each program. The “total” row demonstrates the statistical comparison between Feature-FL and each of the four fault localization techniques, in all artificial faults and all real ones, respectively. For example, in comparison to Dstar on `print_tokens(2 ver.)`, the p-values of 2-tailed, 1-tailed(right), and 1-tailed(left) are 3.08E-03, 9.55E-01 and 1.63E-02 respectively. It means that the *EXAM* of Feature-FL significantly tends to be less than that of Dstar on `print_tokens(2 ver.)`, leading to a BETTER result. We can observe that Feature-FL obtains BETTER results over the four fault localization techniques on all the subject programs. Furthermore, Feature-FL also obtains BETTER results on

TABLE VIII
 STATISTICAL COMPARISON OF Feature-FL AND THE FOUR TECHNIQUES

Program	Comparison	2-tailed	1-tailed(right)	1-tailed(left)	Conclusion
print_tokens (2 ver.)	Feature-FL v.s. Dstar	3.08E-03	9.55E-01	1.63E-02	BETTER
	Feature-FL v.s. Ochiai	2.37E-03	9.99E-01	1.33E-0	BETTER
	Feature-FL v.s. Barinel	1.20E-03	9.99E-01	6.65E-04	BETTER
	Feature-FL v.s. Op2	2.40E-03	1.00E+00	2.82E-03	BETTER
replace	Feature-FL v.s. Dstar	3.94E-04	1.00E+00	2.07E-04	BETTER
	Feature-FL v.s. Ochiai	7.49E-04	1.00E+00	3.91E-04	BETTER
	Feature-FL v.s. Barinel	5.40E-04	1.00E+00	2.82E-04	BETTER
	Feature-FL v.s. Op2	5.20E-04	1.00E+00	4.57E-04	BETTER
schedule (2 ver.)	Feature-FL v.s. Dstar	1.77E-02	9.15E-01	9.25E-03	BETTER
	Feature-FL v.s. Ochiai	1.56E-02	9.93E-01	8.30E-03	BETTER
	Feature-FL v.s. Barinel	1.39E-02	9.93E-01	7.37E-03	BETTER
	Feature-FL v.s. Op2	1.24E-02	9.91E-01	7.32E-03	BETTER
tcas	Feature-FL v.s. Dstar	1.81E-03	9.99E-01	9.31E-04	BETTER
	Feature-FL v.s. Ochiai	8.06E-07	1.00E+00	4.19E-07	BETTER
	Feature-FL v.s. Barinel	1.16E-06	1.00E+00	6.02E-07	BETTER
	Feature-FL v.s. Op2	4.40E-06	1.00E+00	5.32E-06	BETTER
tot_info	Feature-FL v.s. Dstar	3.52E-03	9.98E-01	1.89E-03	BETTER
	Feature-FL v.s. Ochiai	2.54E-04	1.00E+00	1.38E-04	BETTER
	Feature-FL v.s. Barinel	3.84E-04	1.00E+00	2.10E-04	BETTER
	Feature-FL v.s. Op2	2.50E-04	1.00E+00	1.62E-04	BETTER
flex	Feature-FL v.s. Dstar	1.89E-03	9.99E-01	9.58E-04	BETTER
	Feature-FL v.s. Ochiai	1.52E-02	9.25E-01	7.64E-03	BETTER
	Feature-FL v.s. Barinel	1.95E-03	9.99E-01	9.87E-04	BETTER
	Feature-FL v.s. Op2	1.20E-03	9.95E-01	2.12E-03	BETTER
grep	Feature-FL v.s. Dstar	3.68E-02	8.22E-01	1.91E-02	BETTER
	Feature-FL v.s. Ochiai	2.87E-02	8.62E-01	1.49E-02	BETTER
	Feature-FL v.s. Barinel	3.84E-02	9.68E-01	3.60E-02	BETTER
	Feature-FL v.s. Op2	5.40E-02	9.21E-01	3.89E-02	BETTER
total (artificial faults)	Feature-FL v.s. Dstar	2.35E-10	1.00E+00	1.18E-10	BETTER
	Feature-FL v.s. Ochiai	2.67E-17	1.00E+00	1.34E-17	BETTER
	Feature-FL v.s. Barinel	7.58E-20	1.00E+00	3.81E-20	BETTER
	Feature-FL v.s. Op2	7.40E-18	1.00E+00	4.31E-20	BETTER
space	Feature-FL v.s. Dstar	8.16E-04	1.00E+00	4.25E-04	BETTER
	Feature-FL v.s. Ochiai	1.04E-03	9.99E-01	5.42E-04	BETTER
	Feature-FL v.s. Barinel	4.42E-03	9.98E-01	2.29E-03	BETTER
	Feature-FL v.s. Op2	4.31E-03	9.89E-01	2.22E-03	BETTER
sed	Feature-FL v.s. Dstar	2.42E-04	1.00E+00	1.27E-04	BETTER
	Feature-FL v.s. Ochiai	4.62E-03	9.98E-01	2.41E-03	BETTER
	Feature-FL v.s. Barinel	2.03E-03	9.99E-01	1.06E-03	BETTER
	Feature-FL v.s. Op2	8.40E-03	9.98E-01	2.91E-03	BETTER
gzip	Feature-FL v.s. Dstar	1.09E-02	9.69E-01	9.07E-03	BETTER
	Feature-FL v.s. Ochiai	1.09E-02	9.69E-01	9.07E-03	BETTER
	Feature-FL v.s. Barinel	1.09E-02	9.69E-01	9.07E-03	BETTER
	Feature-FL v.s. Op2	1.34E-02	9.53E-01	9.34E-03	BETTER
libtiff	Feature-FL v.s. Dstar	3.62E-02	9.84E-01	2.05E-02	BETTER
	Feature-FL v.s. Ochiai	3.62E-02	9.84E-01	2.05E-02	BETTER
	Feature-FL v.s. Barinel	3.62E-02	9.84E-01	2.05E-02	BETTER
	Feature-FL v.s. Op2	5.22E-03	9.86E-01	2.36E-04	BETTER
python	Feature-FL v.s. Dstar	9.75E-03	9.96E-01	5.99E-03	BETTER
	Feature-FL v.s. Ochiai	9.75E-03	9.96E-01	5.99E-03	BETTER
	Feature-FL v.s. Barinel	9.75E-03	9.96E-01	5.99E-03	BETTER
	Feature-FL v.s. Op2	9.40E-03	9.88E-01	7.32E-03	BETTER
Defects4J	Feature-FL v.s. Dstar	9.75E-03	9.96E-01	5.99E-03	BETTER
	Feature-FL v.s. Ochiai	9.75E-03	9.96E-01	5.99E-03	BETTER
	Feature-FL v.s. Barinel	9.75E-03	9.96E-01	5.99E-03	BETTER
	Feature-FL v.s. Op2	9.49E-03	9.93E-01	5.82E-03	BETTER
total (real faults)	Feature-FL v.s. Dstar	6.09E-09	1.00E+00	3.10E-09	BETTER
	Feature-FL v.s. Ochiai	9.45E-09	1.00E+00	4.80E-09	BETTER
	Feature-FL v.s. Barinel	7.40E-08	1.00E+00	3.76E-08	BETTER
	Feature-FL v.s. Op2	5.89E-07	1.00E+00	5.39E-07	BETTER

“total” comparison over the four fault localization techniques in both cases of artificial faults and real ones. The results show that Feature-FL significantly outperforms the four state-of-the-art fault localization techniques, despite whether it is the context of artificial faults or real faults.

Summary: We evaluate Feature-FL over the four state-of-the-art techniques in two different contexts, i.e., artificial faults and real faults. We first show the *EXAM* distribution, then use the *RImp* for obtaining detailed improvement. To overcome the limitations of *EXAM* and *RImp* metrics and obtain the overall

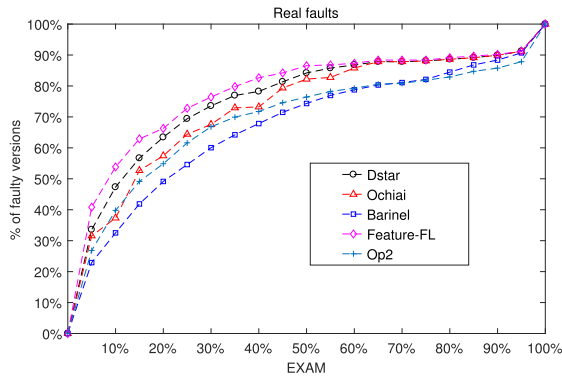


Fig. 11. *EXAM* Distribution of Feature-FL and FL in multiple faults programs.

performance of a fault localization, we further show the results under the Top-1, Top-3, and Top-5 metrics, and finally apply the quantitative analysis using Wilcoxon-signed-rank test to justify the advantages of Feature-FL. Based on all experimental results, we can conclude that Feature-FL significantly improves fault localization effectiveness in both artificial faults and real faults.

E. Threats to Validity

Threats When There are Multiple Faults: The effectiveness of Feature-FL on programs with multiple faults should be explicitly discussed. We use the faulty versions of programs with real faults in our subject programs to evaluate the effectiveness of Feature-FL; they are a subset of all faulty versions of the last 11 programs in Table II. Followed by Wong *et al.* [15], we could also apply *EXAM* to multiple faults through the definition of the percentage of the statements to be examined before finding *all* faulty statements. From Fig. 11, the curve of Feature-FL is above the curves of the other four SFL techniques, which means Feature-FL can also locate multiple bugs more effectively than other four SFL techniques. Note that the Feature-FL could not reach 100% faster than the other techniques, which means Feature-FL has some limitations on multiple faults programs. The reason may be that the feature selection algorithms we used do not consider the combination of features. Further studies about the combination features of feature selection algorithms should be conducted to validate the effectiveness on multiple faults.

Threats to Test Strategies: Different test strategies will lead to different branching execution probabilities, which may consequently influence the effectiveness of Feature-FL. Also, the effectiveness of other localization techniques will vary as the test strategy changes. The influence of test strategy on fault localization is an important and interesting research topic, and we will leave it to future work.

Threats to Internal Validity: This type of threats involves the relationship between independent and dependent variables in this study, which are beyond researchers' knowledge. There may be chances that some undetected implementation flaws existing in our experiment may have affected the results. To ensure the

accuracy of the experiments, we have carefully realized the relevant techniques and comprehensive functional testing.

Threats to External Validity: This type of threats corresponds to the generalization of the experimental results. A threat is about the subject programs. For obtaining reliable experimental results, we choose seven small programs with artificial faults and seven large real-life programs mostly with real faults because they are commonly used in debugging. Even so, there still exist many unknown and complicated factors in realistic debugging. Therefore, it would be worthwhile to use more programs (e.g., multiple-faults programs and large-sized programs) to further strengthen the experimental results.

Threats to Construct Validity: This type of threats concerns the appropriateness of the evaluation measurement. We use *EXAM* and *RImp* to evaluate fault localization effectiveness. The two metrics are widely used in fault localization community [16], [45], and thus the threat is acceptably mitigated.

V. RELATED WORK

There exists a large body of research on fault localization, e.g., information retrieval based fault localization [54]–[56] and deep-learning-based fault localization [10], [26], [57]. This section surveys closely related work on SFL techniques. More other work on fault localization can refer to a survey by Wong *et al.* [1].

Due to its straightforward idea and promising results, SFL has received considerable attention and motivated much research on fault localization. SFL usually utilizes the coverage information of program entities (e.g., statements [17], [20], [58]–[60], basic blocks [18], components [19], and paths [61]) to evaluate the suspiciousness of an entity of being faulty. SFL shares an intuition that a program entity, covered by more failing test cases and less passing test cases, should have high suspiciousness of being faulty. Following this intuition, researchers have proposed many suspiciousness evaluation formulas, e.g., Ochiai [18], Jaccard [19], Tarantula [20], GP-evolved formulas [62], Naish1 [17], and Wong1 [59]. Many of them (e.g., Tarantula [20], Ochiai [17]) have been widely integrated into automated program repair (APR) pipelines as a crucial start in [63]–[70], which indicates SFL is one of the important guidelines for the following steps of APR. However, the above approaches mainly discover statistical coincidence and lack the combination of diverse amount of information. Thus, Feature-FL combines feature diversity into fault localization to improve localization effectiveness.

In addition to the origin formulas, researchers have proposed some enhanced fault localization techniques based on spectrum. In order to alleviate the imbalance of test cases, the method of cloning failing test cases has been proposed [71], [72]. Gao *et al.* [71] first proposed a cloning failing test cases strategy to produce a balanced test suite. They adopted rigorous analysis from a theoretical perspective to prove that their idea would have positive effect on SFL techniques. Followed by Gao *et al.* [71], Zhang *et al.* [72] conducted a large-scale experiment of cloning failing test cases on SFL techniques, which the empirical results are in line with the former theoretical analysis. In this

work [73], they validate the effectiveness of cloning technique on deep-learning-based fault localization. Li *et al.* [28] devised the enhanced code coverage matrix (i.e., spectrum) by setting the value of the error-exhibiting lines to -1 . The enhanced code coverage matrix encodes more information than original one, which can be fully explored by convolutional neural networks. In contrast to optimizing test datasets, our work focuses on presenting effective fault localization methodology.

Furthermore, for supporting locating multiple faults, Jones *et al.* [74] adopted a clustering technique to divide failing test cases into different clusters, where each cluster represents only one fault. With the assistance of this clustering technique, SFL techniques are applicable to locate multiple faults by locating the one fault of each cluster in parallel. Abreu *et al.* [75] proposed a spectrum-based multiple fault localization approach called Zoltar-M by integrating the methodology of SFL with model-based diagnosis. Zheng *et al.* [76] proposed a fast software multifault localization framework via utilizing genetic algorithms with simulated annealing. Their solution successfully transforms a multifault localization problem into a search one. For alleviating the test oracle problem, Abreu *et al.* [77] designed invariants for fault localization and utilized error detection to judge failures. Xie *et al.* [78] leveraged metamorphic testing to define two key concepts (i.e., metamorphic slice and metamorphic result) for constructing a fault localization methodology, which successfully replaces a test result of being *failing* or *passing* with a metamorphic result of being *violated* or *nonviolated*. In order to reduce the inaccuracy caused by sensitive factors such as the characteristic of faults and the quality of the test suite, Gopinath *et al.* [79] incorporated the merits of specification-based analysis into fault localization to enable more precise fault localization. Le *et al.* [80], [81] used machine learning (e.g., feature extraction) to predict the effectiveness of fault localization techniques. Feature-FL also faces the above problems (e.g., multiple faults and oracle problem), and the above approaches can also serve potential solutions to Feature-FL.

Since the rapid progress on SFL techniques has made a lot of achievements in this research area. Some researchers try to summarize SFL techniques from both theoretical and experimental levels. Xie *et al.* [22], [23] theoretically summarized the maximal effectiveness that different SFL techniques can achieve. Recently, Pearson *et al.* [16] systematically evaluated and summarized the existing state-of-the-art techniques, showing that the promising results of SFL techniques from a comprehensive experimental study. From their studies, we can see the maximum effectiveness of SFL techniques that have reached. Thus, we leverage a recent popular method named feature selection to propose Feature-FL for locating faults via a feature view different from SFL. Our study has shown that Feature-FL is more effective than four state-of-the-art SFL techniques, and significantly improves localization effectiveness.

VI. CONCLUSION

In this article, we tried to propose a fault localization methodology different from the state-of-the-art view. To achieve this

goal, we proposed a Feature-FL methodology by locating faults from the feature view. Feature-FL defines branching execution probability to depict program behaviors as the values of features, then uses feature selection to quantify the relevance of each feature to failures, and, finally, associates each feature with its corresponding statement to rank all statements in terms of suspiciousness computed by the relevance. We presented six basic and typical feature selection criteria for Feature-FL. We conducted a large-scale experimental study to investigate the effectiveness distribution among different feature selection criteria and evaluate Feature-FL over four state-of-the-art techniques in two different contexts, namely artificial faults and real faults. The results recommend for using the three effective feature selection criteria (i.e., Pearson correlation coefficient, Spearman's rank correlation coefficient, and Kendall rank correlation coefficient) as the candidates of applying Feature-FL in practice. In addition, the results also showed that Feature-FL significantly improves fault localization effectiveness, despite whether it is the context of artificial faults or real faults.

As for future work, since feature selection is a family method, we plan to optimize Feature-FL by taking more scoring metrics into consideration. In addition, we plan to extend Feature-FL to the context of multiple faults.

REFERENCES

- [1] W. E. Wong, R. Gao, Y. Li, R. Abreu, and F. Wotawa, "A survey on software fault localization," *IEEE Trans. Softw. Eng.*, vol. 42, no. 8, pp. 707–740, Aug. 2016.
- [2] X. Mao, Y. Lei, Z. Dai, Y. Qi, and C. Wang, "Slice-based statistical fault localization," *J. Syst. Softw.*, vol. 89, pp. 51–62, 2014.
- [3] C. M. Tang, W. Chan, Y. T. Yu, and Z. Zhang, "Accuracy graphs of spectrum-based fault localization formulas," *IEEE Trans. Rel.*, vol. 66, no. 2, pp. 403–424, Jun. 2017.
- [4] B. Hofer, A. Höfler, and F. Wotawa, "Combining models for improved fault localization in spreadsheets," *IEEE Trans. Rel.*, vol. 66, no. 1, pp. 38–53, Mar. 2017.
- [5] J. Jiang, R. Wang, Y. Xiong, X. Chen, and L. Zhang, "Combining spectrum-based fault localization and statistical debugging: An empirical study," in *Proc. IEEE Int. Conf. Automated Softw. Eng.*, 2019, pp. 502–514.
- [6] Z. Zuo, S.-C. Khoo, and C. Sun, "Efficient predicated bug signature mining via hierarchical instrumentation," in *Proc. Int. Symp. Softw. Testing Anal.*, 2014, pp. 215–224.
- [7] W. E. Wong, Y. Shi, Y. Qi, and R. Golden, "Using an RBF neural network to locate program bugs," in *Proc. IEEE 19th Int. Symp. Softw. Rel. Eng.*, 2008, pp. 27–36.
- [8] W. E. Wong, V. Debroy, R. Golden, X. Xu, and B. Thuraisingham, "Effective software fault localization using an RBF neural network," *IEEE Trans. Rel.*, vol. 61, no. 1, pp. 149–169, Mar. 2012.
- [9] C. Sun and S.-C. Khoo, "Mining succinct predicated bug signatures," in *Proc. 9th Joint Meeting Found. Softw. Eng.*, 2013, pp. 576–586.
- [10] X. Li, W. Li, Y. Zhang, and L. Zhang, "DeepFL: Integrating multiple fault diagnosis dimensions for deep fault localization," in *Proc. 28th ACM Special Interest Group Softw. Eng. Int. Symp. Softw. Testing Anal.*, 2019, pp. 169–180.
- [11] J. Tu, X. Xie, T. Y. Chen, and B. Xu, "On the analysis of spectrum based fault localization using hitting sets," *J. Syst. Softw.*, vol. 147, pp. 106–123, 2019.
- [12] R. Gao and W. E. Wong, "MSeer—An advanced technique for locating multiple bugs in parallel," *IEEE Trans. Softw. Eng.*, vol. 45, no. 3, pp. 301–318, Mar. 2019.
- [13] Y. Lei, C. Sun, X. Mao, and Z. Su, "How test suites impact fault localisation starting from the size," *Institution Eng. Technol. Softw.*, vol. 12, no. 3, pp. 190–205, 2018.
- [14] D. Zou, J. Liang, Y. Xiong, M. D. Ernst, and L. Zhang, "An empirical study of fault localization families and their combinations," *IEEE Trans. Softw. Eng.*, vol. 47, no. 2, pp. 332–347, Feb. 2021.

- [15] W. E. Wong, V. Debroy, R. Gao, and Y. Li, "The DStar method for effective software fault localization," *IEEE Trans. Rel.*, vol. 63, no. 1, pp. 290–308, Mar. 2014.
- [16] S. Pearson *et al.*, "Evaluating and improving fault localization," in *Proc. Int. Conf. Softw. Eng.*, 2017, pp. 609–620.
- [17] L. Naish, H. Lee, and K. Ramamohanarao, "A model for spectra-based software diagnosis," *Assoc. Comput. Machinery Trans. Softw. Eng. Methodol.*, vol. 20, no. 3, pp. 1–32, 2011.
- [18] R. Abreu, P. Zoetewij, and A. Van Gemund, "On the accuracy of spectrum-based fault localization," in *Proc. IEEE Testing: Academic Ind. Conf. Pract. Res. Techn.-Mutation*, 2007, pp. 89–98.
- [19] M. Chen, E. Kiciman, E. Fratkin, A. Fox, and E. Brewer, "Pinpoint: Problem determination in large, dynamic internet services," in *Proc. IEEE Int. Conf. Dependable Syst. Netw.*, 2002, pp. 595–604.
- [20] J. Jones, M. Harrold, and J. Stasko, "Visualization of test information to assist fault localization," in *Proc. 24th Int. Conf. Softw. Eng.*, 2002, pp. 467–477.
- [21] W. E. Wong, V. Debroy, and B. Choi, "A family of code coverage-based heuristics for effective fault localization," *J. Syst. Softw.*, vol. 83, no. 2, pp. 188–208, 2010.
- [22] X. Xie, T. Y. Chen, F.-C. Kuo, and B. Xu, "A theoretical analysis of the risk evaluation formulas for spectrum-based fault localization," *Assoc. Comput. Machinery Trans. Softw. Eng. Methodol.*, vol. 22, pp. 1–40, 2013.
- [23] X. Xie, F.-C. Kuo, T. Y. Chen, S. Yoo, and M. Harman, "Provably optimal and human-competitive results in SBSE for spectrum based fault localisation," in *Proc. 5th Symp. Search-Based Softw. Eng.*, Springer, 2013, pp. 224–238.
- [24] W. E. Wong, L. Zhao, Y. Qi, K.-Y. Cai, and J. Dong, "Effective fault localization using BP neural networks," in *Proc. 4th Int. Conf. Softw. Eng. Knowl. Eng.*, 2007, pp. 374–379.
- [25] W. Zheng, D. Hu, and J. Wang, "Fault localization analysis based on deep neural network," *Math. Problems Eng.*, vol. 2016, pp. 1–11, 2016.
- [26] Z. Zhang, Y. Lei, X. Mao, and P. Li, "CNN-FL: An effective approach for localizing faults using convolutional neural networks," in *Proc. IEEE 26th Int. Conf. Softw. Anal. Evol. Reeng.*, 2019, pp. 445–455.
- [27] Z. Zhang, Y. Lei, Q. Tan, X. Mao, P. Zeng, and X. Chang, "Deep learning-based fault localization with contextual information," *Inst. Electron. Inf. Commun. Engineers Trans. Inf. Syst.*, vol. 100, no. 12, pp. 3027–3031, 2017.
- [28] Y. Li, S. Wang, and T. N. Nguyen, "Fault localization with code coverage representation learning," in *Proc. IEEE/ACM 43rd Int. Conf. Softw. Eng.*, 2021, pp. 661–673.
- [29] P. Duda and D. G. Stork, *Pattern Classification*. Hoboken, NJ, USA: Wiley-Interscience Publication, 2001.
- [30] I. Guyon and A. Elisseeff, "An introduction to variable and feature selection," *J. Mach. Learn. Res.*, vol. 3, pp. 1157–1182, 2003.
- [31] M. H. Liu H, "Feature Selection for Knowledge Discovery and Data Mining," Berlin, Germany: Springer, 2012.
- [32] D. L. Donoho, "For most large underdetermined systems of equations, the minimal 1-norm near-solution approximates the sparsest near-solution," *Commun. Pure Appl. Math.*, vol. 59, no. 7, pp. 907–934, 2006.
- [33] A. Y. Ng, "Feature selection, 1 vs. 1.2 regularization, and rotational invariance," in *Proc. 21st Int. Conf. Mach. Learn.*, 2004, p. 78.
- [34] C. Liu, X. Yan, L. Fei, J. Han, and S. P. Midkiff, "SOBER: Statistical model-based bug localization," *ACM ACM Special Interest Group Softw. Eng. Notes*, vol. 30, no. 5, pp. 286–295, 2005.
- [35] X. Xiao, Y. Pan, B. Zhang, G. Hu, Q. Li, and R. Lu, "ALBFL: A novel neural ranking model for software fault localization via combining static and dynamic features," *Inf. Softw. Technol.*, 2021, Art. no. 106653.
- [36] S. Kim, M. Ouyang, and X. Zhang, "Compute spearman correlation coefficient with matlab/CUDA," in *Proc. IEEE Int. Symp. Signal Process. Inf. Technol.*, 2012, pp. 000055–000060.
- [37] M. G. Kendall, "A new measure of rank correlation," *Biometrika*, vol. 30, no. 1–2, pp. 81–93, 1938. doi: [10.1093/biomet/30.1-2.81](https://doi.org/10.1093/biomet/30.1-2.81).
- [38] A. Li, Y. Lei, and X. Mao, "Towards more accurate fault localization: An approach based on feature selection using branching execution probability," in *Proc. IEEE Int. Conf. Softw. Qual. Rel. Secur.*, 2016, pp. 431–438.
- [39] R. L. Plackett, "Karl Pearson and the Chi-squared test," *Int. Stat. Rev./Revue Int. Statistique*, pp. 59–72, 1983.
- [40] A. Rui, P. Zoetewij, R. Golsteijn, and A. J. C. V. Gemund, "A practical evaluation of spectrum-based fault localization," *J. Syst. Softw.*, vol. 82, no. 11, pp. 1780–1792, 2009.
- [41] R. Abreu, P. Zoetewij, and A. J. C. Van Gemund, "Spectrum-based multiple fault localization," in *Proc. 24th IEEE/ACM Int. Conf. Automated Softw. Eng.*, 2009, pp. 88–99.
- [42] "Kendall rank correlation coefficient." Accessed: 2021. [Online]. Available: https://en.wikipedia.org/wiki/Kendall_rank_correlation_coefficient#Tau-b
- [43] "Chi-squared test." Accessed: 2021. [Online]. Available: https://en.wikipedia.org/wiki/Chi-squared_test
- [44] Y. Guédon, D. Barthélémy, Y. Caraglio, and E. Costes, "Pattern analysis in branching and axillary flowering sequences," *J. Theor. Biol.*, vol. 212, no. 4, pp. 481–520, 2001.
- [45] V. Debroy, W. Wong, X. Xu, and B. Choi, "A grouping-based strategy to improve the effectiveness of fault localization techniques," in *Proc. IEEE 10th Int. Conf. Qual. Softw.*, 2010, pp. 13–22.
- [46] X. Xu, V. Debroy, W. Eric Wong, and D. Guo, "Ties within fault localization rankings: Exposing and addressing the problem," *Int. J. Softw. Eng. Knowl. Eng.*, vol. 21, no. 06, pp. 803–827, 2011.
- [47] C. Parnin and A. Orso, "Are automated debugging techniques actually helping programmers?," in *Proc. Int. Symp. Softw. Testing Anal.*, 2011, pp. 199–209.
- [48] A. Ang, A. Perez, A. Van Deursen, and R. Abreu, "Revisiting the practical use of automated software fault localization techniques," in *Proc. IEEE Int. Symp. Softw. Reliab. Eng. Workshops*, 2017, pp. 175–182.
- [49] P. S. Kochhar, X. Xia, D. Lo, and S. Li, "Practitioners' expectations on automated fault localization," in *Proc. 25th Int. Symp. Softw. Testing Anal.*, 2016, pp. 165–176.
- [50] J. Sohn and S. Yoo, "FLUCCS: Using code and change metrics to improve fault localization," in *Proc. 26th ACM Int. Symp. Softw. Testing Anal.*, 2017, pp. 273–283.
- [51] G. W. Corder and D. I. Foreman, *Nonparametric Statistics for Non-Statisticians: A Step-By-Step Approach*. Hoboken, NJ, USA: Wiley, 2014.
- [52] H. Abdi, "The Bonferonni and Aïdák corrections for multiple comparisons," *Encyclopedia Meas. Statist.*, vol. 3, pp. 103–107, 2007.
- [53] J. D. Gibbons and S. Chakraborti, "Nonparametric statistical inference," in *International Encyclopedia of Statistical Science*. Berlin, Germany: Springer, 2011, pp. 977–979.
- [54] J. Zhou, H. Zhang, and D. Lo, "Where should the bugs be fixed? More accurate information retrieval-based bug localization based on bug reports," in *Proc. IEEE 34th Int. Conf. Softw. Eng.*, 2012, pp. 14–24.
- [55] Q. Wang, C. Parnin, and A. Orso, "Evaluating the usefulness of IR-based fault localization techniques," in *Proc. Int. Symp. Softw. Testing Anal.*, 2015, pp. 1–11.
- [56] A. R. Chen, T.-H. P. Chen, and S. Wang, "Pathidea: Improving information retrieval-based bug localization by re-constructing execution paths using logs," *IEEE Trans. Softw. Eng.*, to be published, doi: [10.1109/TSE.2021.3071473](https://doi.org/10.1109/TSE.2021.3071473).
- [57] Z. Zhang, Y. Lei, X. Mao, M. Yan, L. Xu, and X. Zhang, "A study of effectiveness of deep learning in locating real faults," *Inf. Softw. Technol.*, vol. 131, 2021, Art. no. 106486.
- [58] J. Jones and M. Harrold, "Empirical evaluation of the tarantula automatic fault-localization technique," in *Proc. 20th Int. Conf. Automated Softw. Eng.*, 2005, pp. 273–282.
- [59] W. Wong, Y. Qi, L. Zhao, and K. Cai, "Effective fault localization using code coverage," in *Proc. IEEE 31st Annu. Int. Comput. Softw. Appl. Conf.*, 2007, pp. 449–456.
- [60] R. Santelices, J. Jones, Y. Yu, and M. Harrold, "Lightweight fault-localization using multiple coverage types," in *Proc. IEEE 31st Int. Conf. Softw. Eng.*, 2009, pp. 56–66.
- [61] J. H. Mary, R. Gregg, W. Rui, and Y. Liu, "An empirical investigation of program spectra," in *Proc. ACM SIGPLAN-SIGSOFT Workshop Prog. Anal. Softw. Tools Eng.*, 1998, pp. 83–90.
- [62] S. Yoo, "Evolving human competitive spectra-based fault localisation techniques," in *Proc. 4th Int. Symp. Search-Based Softw. Eng.*, 2012, pp. 244–258.
- [63] J. Jiang, Y. Xiong, H. Zhang, Q. Gao, and X. Chen, "Shaping program repair space with existing patches and similar code," in *Proc. 27th ACM SIGSOFT Int. Symp. Softw. Testing Anal.*, 2018, pp. 298–309.
- [64] K. Liu, A. Koyuncu, D. Kim, and T. F. Bissyandé, "TBar: Revisiting template-based automated program repair," in *Proc. 28th ACM SIGSOFT Int. Symp. Softw. Testing Anal.*, 2019, pp. 31–42.
- [65] S. Mechtaev, J. Yi, and A. Roychoudhury, "Angelix: Scalable multiline program patch synthesis via symbolic analysis," in *Proc. 38th Int. Conf. Softw. Eng.*, 2016, pp. 691–701.
- [66] F. Long *et al.*, "Automatic patch generation via learning from successful human patches," Ph.D. dissertation, Massachusetts Institute of Technology, Cambridge, MA, USA, 2018.
- [67] F. Long and M. Rinard, "Staged program repair with condition synthesis," in *Proc. 10th Joint Meeting Found. Softw. Eng.*, 2015, pp. 166–178.

- [68] M. Martinez and M. Monperrus, "Astor: A program repair library for java," in *Proc. 25th Int. Symp. Softw. Testing Anal.*, 2016, pp. 441–444.
- [69] K. Liu, A. Koyuncu, K. Kim, D. Kim, and T. F. Bissyandé, "LSRepair: Live search of fix ingredients for automated program repair," in *Proc. IEEE 25th Asia-Pacific Softw. Eng. Conf.*, 2018, pp. 658–662.
- [70] Y. Li, S. Wang, and T. N. Nguyen, "DLfix: Context-based code transformation learning for automated program repair," in *Proc. ACM/IEEE 42nd Int. Conf. Softw. Eng.*, 2020, pp. 602–614.
- [71] Y. Gao, Z. Zhang, L. Zhang, C. Gong, and Z. Zheng, "A theoretical study: The impact of cloning failed test cases on the effectiveness of fault localization," in *Proc. IEEE 13th Int. Conf. Qual. Softw.*, 2013, pp. 288–291.
- [72] L. Zhang, L. Yan, Z. Zhang, J. Zhang, W. Chan, and Z. Zheng, "A theoretical analysis on cloning the failed test cases to improve spectrum-based fault localization," *J. Syst. Softw.*, vol. 129, pp. 35–57, 2017.
- [73] Z. Zhang, Y. Lei, X. Mao, M. Yan, L. Xu, and J. Wen, "Improving deep-learning-based fault localization with resampling," *J. Softw.: Evol. Process.*, vol. 33, no. 3, 2021, Art. no. 2312.
- [74] J. A. Jones, J. F. Bowring, and M. J. Harrold, "Debugging in parallel," in *Proc. Int. Symp. Softw. Testing Anal.*, 2007, pp. 16–26.
- [75] R. Abreu, P. Zoetewij, and A. J. Van Gemund, "Simultaneous debugging of software faults," *J. Syst. Softw.*, vol. 84, no. 4, pp. 573–586, 2011.
- [76] Y. Zheng, Z. Wang, X. Fan, X. Chen, and Z. Yang, "Localizing multiple software faults based on evolution algorithm," *J. Syst. Softw.*, vol. 139, pp. 107–123, 2018.
- [77] R. Abreu, A. González, P. Zoetewij, and A. J. van Gemund, "Automatic software fault localization using generic program invariants," in *Proc. ACM Symp. Appl. Comput.*, 2008, pp. 712–717.
- [78] X. Xie, W. E. Wong, T. Y. Chen, and B. Xu, "Metamorphic slice: An application in spectrum-based fault localization," *Inf. Softw. Technol.*, vol. 55, pp. 866–879, 2012.
- [79] D. Gopinath, R. N. Zaeem, and S. Khurshid, "Improving the effectiveness of spectra-based fault localization using specifications," in *Proc. IEEE 27th IEEE/ACM Int. Conf. Automated Softw. Eng.*, 2012, pp. 40–49.
- [80] T.-D. B. Le and D. Lo, "Will fault localization work for these failures? An automated approach to predict effectiveness of fault localization tools," in *Proc. IEEE Int. Conf. Softw. Maintenance*, 2013, pp. 310–319.
- [81] T.-D. B. Le, F. Thung, and D. Lo, "Predicting effectiveness of IR-based bug localization techniques," in *Proc. IEEE 25th Int. Symp. Softw. Rel. Eng.*, 2014, pp. 335–345.



Yan Lei received the BA, MA, and Ph.D. degrees in computer science and technology, all from the National University of Defense Technology, China. He is an Associate Professor at the School of Big Data and Software Engineering in Chongqing University, China. His research interests include fault localization, program repair, program slicing, etc.



Huan Xie received the B.S. degree in software engineering in 2020 from Chongqing University, Chongqing, China, where he is currently working toward the M.S. degree in Electronic Information with the School of Big Data and Software Engineering. His main research interests include fault localization, program repair, and API misuse.



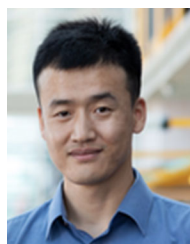
Tao Zhang received the Ph.D. degree from University of Seoul, South Korea in 2013. He is currently an associate professor at Faculty of Information Technology, Macau University of Science and Technology. His main research interests include mining software repositories, intelligent software engineering, mobile App security.



Meng Yan received Ph.D. degree in June 2017 under the supervision of Prof. Xiaohong Zhang from Chongqing University, China. He is now a Research Professor at the School of Big Data & Software Engineering, Chongqing University, China. His current research focuses on how to improve developers' productivity, how to improve software quality, and how to reduce the effort during software development by analyzing rich software repository data.



Zhou Xu received the dual Ph.D. degree from the School of Computer Science, Wuhan University, China, and the Department of Computing, The Hong Kong Polytechnic University, Hong Kong. He is currently an Assistant Professor at the School of Big Data and Software Engineering, Chongqing University, Chongqing, China. His research interests include feature engineering and data mining.



Chengnian Sun received a Ph.D. degree from the School of Computing, National University of Singapore. He is currently an assistant professor with Cheriton School of Computer Science, University of Waterloo, Canada. His research interests are in software engineering and programming languages, focusing on techniques, tools, and methodologies for improving software quality and developers' productivity. He has a Ph.D. in Computer Science from National University of Singapore.