

NeuralCCD: Integrating Multiple Features for Neural Coincidental Correctness Detection

Zhou Tao¹, Yan Lei^{1,2*}, Huan Xie¹, Jian Hu¹

¹ School of Big Data & Software Engineering, Chongqing University, Chongqing, China

² Peng Cheng Laboratory, Shenzhen, China

zhoutao@stu.cqu.edu.cn, yanlei@cqu.edu.cn, huanxie@cqu.edu.cn, jianhu@cqu.edu.cn

Abstract—Fault localization seeks to locate the suspicious statements possible for causing a program failure. Experimental evidence shows that fault localization effectiveness is affected adversely by the existence of coincidental correctness (CC) test cases, where a CC test case denotes the test case which executes a fault but no failure occurs. Even worse, CC test cases are prevailing in realistic testing and debugging, leading to a severe issue on fault localization effectiveness. Thus, it is indispensable to accurately detect CC test cases and alleviate their harmful effect on fault localization effectiveness.

To address this problem, we propose NeuralCCD: a neural coincidental correctness detection approach by integrating multiple features. Specifically, NeuralCCD first leverages suspiciousness score, coverage ratio and similarity to define three CC detection features. Based on these CC detection features and CC labels, NeuralCCD utilizes multi-layer perceptron to learn a different feature-based model for a program, and finally combine the trained models of different programs as an ensemble system to detect CC test cases. To evaluate the effectiveness of NeuralCCD, we conduct large-scale experiments on 247 faulty version of five representative benchmarks and compare NeuralCCD with four state-of-the-art CC detection approaches. The experimental results show that NeuralCCD significantly improves the effectiveness of CC detection, *e.g.*, NeuralCCD yields by at most 109.5%, 93% and 81.3% improvement of Top-1, Top-3 and Top-5 over *Tech-I* when utilized in Dstar formular.

Index Terms—fault localization, coincidental correctness, deep learning, multiple features

I. INTRODUCTION

With the developing and progressing of software, the software size and complexity sharply increases. Consequently, debugging a fault in a program is a painstaking and time-consuming task in software development and maintenance. To reduce the debugging cost, researches have proposed many fault localization (FL) techniques (*e.g.*, [1]–[8]) to provide automatic assistance in seeking the faults in recent years. These approaches try to locate the buggy statements through various speciousness evaluation algorithms and have shown promising results, *e.g.*, the widely-used Spectrum-based Fault Localization (SFL) [11]–[14].

However, there is a type of test cases referred to Coincidental Correctness (CC) test cases prevailing in real-life fault location work [18], [19], posing a severe adverse effect on FL. CC is a situation that within the execution process of a test case, the buggy entity is executed but the program outputs a correct result as expected. Since a CC test case executes

a fault whereas it produces a correct program output, the presence of CC can introduce some noises into the statistics of program behavior which is captured and utilized for FL. Consequently, it will lead to the inaccuracy of FL models, and finally decreases the effectiveness of fault localization. Prior studies [18]–[20] have empirically shown the harmful impact of CC on fault localization effectiveness.

Thus, there is an urgent demand to solve the CC test case detection problem. Some CC detection approaches are proposed to identify CC test cases [18], [22]–[28] based on heuristic algorithms [22] or traditional machine learning classification algorithms [18], [23], [25], [28]. These approaches just utilize one feature (*i.e.*, the similarity of CC test cases and failing test cases on coverage information) to construct their different CC detection models. Nevertheless, only one feature may miss some other inner features of CC test cases, restricting the CC detection effectiveness of these approaches.

To address this issue, our insight is that if we can extract different inner features, rather than one feature, of CC test cases, integrating these features into CC detection models should be beneficial for improving CC detection effectiveness. With the rapid development of deep learning, we observe that deep learning models, which are prevalent today, have a strong ability for feature learning [29]. Thus, we consider if we use different inner CC features as the training data to learn an integrated CC detection model, CC detection effectiveness may be improved.

Following the above analysis, we propose NeuralCCD: a neural CC detection approach via integrating multiple features along with a neural network. NeuralCCD first identifies candidate program elements potentially related to CC, and use them to extract CC detection features. Specifically, we define three CC detection features, *i.e.*, suspiciousness score feature, coverage ratio feature and similarity feature, to make CC test cases being differentiated from passing test cases. To integrate the multiple features, NeuralCCD then utilizes these features as the input to a multi-layer perceptron and train an CC detection feature based model.

After that, NeuralCCD combines different CC detection models of different programs as an ensemble system to detect CC test cases. Finally, NeuralCCD changes the labels of the detected CC test cases in the original test cases to improve fault localization effectiveness.

To evaluate our approach NeuralCCD, we design and con-

*Yan Lei is the corresponding author.

duct large-scale experiments on 247 faulty version of five representative benchmarks. We compare NeuralCCD with four state-of-the-art CC detection approaches. The experimental results show that our approach is effective at detecting CC test cases and improves the effectiveness of fault localization. Specifically, the experimental results indicate that our approach improves the effectiveness of fault localization by at least 204.01%, 308.83%, and 199.38% on Top-1, Top-3, and Top-5 metrics on average compared with baselines, respectively.

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

- We propose a neural coincidental correctness approach NeuralCCD, which integrates multiple CC features into CC test case detection via a neural network (*i.e.*, a multi-layer perceptron).
- We use the candidate program elements potentially associated with CC to define three CC detection features (*i.e.*, suspiciousness score feature, coverage ratio feature and similarity feature) for differentiating CC test cases from passing test cases.
- We conduct large-scale experiments and compare our approach with four state-of-the-art CC detection approaches, showing that NeuralCCD significantly improves fault localization effectiveness.

The remainder of this paper is organized as follows. Section II introduces background information. Section III presents our approach NeuralCCD. Section IV and Section V show the experimental results and discussion. Section VI discusses related work and Section VII draws the conclusion.

II. BACKGROUND

A. Fault Localization

This section will introduce the amongst most popular approach used by our experiments, *i.e.*, Spectrum-based Fault Localization (SFL) [11]–[14].

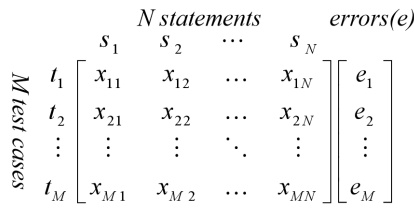


Fig. 1. Structure of information model

SFL [11]–[14] evaluates the suspiciousness of a program entity being faulty by exploiting the correlation between coverage information and test results to locate the buggy entity. SFL defines and utilizes an information model (*i.e.*, program spectrum and test results) as an input, and evaluates the suspiciousness score of each program element.

Fig. 1 shows the structure of the information model. To be specific, suppose that we have a faulty program P with N statements running against a test suite T with M test cases,

where $P = \{s_1, s_2, \dots, s_N\}$ and $T = \{t_1, t_2, \dots, t_M\}$. After executing all test cases of T on P , we can acquire the coverage matrix and result vector of the information model. As shown in Fig. 1, the $M \times N$ matrix is the coverage matrix recording the coverage information of each test case for all statements. An element x_{ij} in the coverage matrix denotes whether the statement s_j is covered by the test case t_i . $x_{ij}=1$ means t_i covers s_j and otherwise $x_{ij}=0$. An element e_i in the result vector e represents whether the test case is failed or not. $e_i=1$ means t_i is failed and otherwise $e_i=0$.

Based on the coverage matrix and result vector, SFL uses statistical analysis to evaluate the likelihood of each program entity being faulty. Researchers have proposed various SFL approaches and Table I depicts the typical SFL formulas, among which the prior study [39] has empirically shown that Dstar, Ochiai, Barinel are the most effective ones in locating real faults. Thus, our experimental study will include the three effective FL approaches.

TABLE I
Typical SFL approaches.

Name	formula
Dstar	$\frac{a_{ef}^*}{a_{nf} + a_{ep}}$
Ochiai	$\frac{a_{ef}}{\sqrt{(a_{ef} + a_{nf})(a_{ef} + a_{ep})}}$
Barinel	$2 * (a_{ef} + \sqrt{a_{np}} + \sqrt{a_{ep}})$
Optimal_P	$\frac{a_{ef}}{a_{ep} + a_{np} + 1}$
Russel & Rao	$\frac{a_{ef}}{a_{ep} + a_{np} + a_{ef} + a_{nf}}$
GP02	$2(a_{ef} + \sqrt{a_{np}}) + \sqrt{a_{ep}}$
GP19	$a_{ef} \sqrt{ a_{ep} - a_{ef} + a_{nf} - a_{np} }$
GP03	$\sqrt{ a_{ef}^2 - \sqrt{a_{ep}} }$
Op2	$\frac{a_{ef}}{a_{ep} + a_{np} + 1}$
Jaccard	$\frac{a_{ef}}{a_{nf} + a_{ep} + a_{np}}$

a_{ef} is the number of one statement that executed in failing test cases while a_{ep} is the number of one statement that executed in passing test cases. a_{nf} and a_{np} means the number of one statement that not executed in failing test cases and passing test cases respectively.

B. Coincidental Correctness

Although existing FL approaches (*e.g.*, SFL) deliver promising fault localization results, their effectiveness in real-life scenarios will be adversely influenced by coincidental correctness (CC) [18], [19]. A CC test case denotes the test case executes a fault but produces a correct program output. The studies [18]–[20] have shown that the existence of CC is prevalent and harmful for fault localization.

Fig. 2 illustrates an example program with a fault at the 7th statement running against ten test cases. We use the two FL approaches (*i.e.*, *Ochiai* and *Op²* in Table I) as the representative to compute the suspiciousness of each statement being faulty. Based on the example program and its execution information, Table II show the results of *Ochiai* and *Op²* with and without CC for illustrating how CC decreases FL effectiveness.

t_1	t_2	t_3	t_4	t_5
$a = 2$ $b = 1$ $c = 2$	$a = 5$ $b = 3$ $c = 2$	$a = 8$ $b = 2$ $c = 3$	$a = 4$ $b = 1$ $c = 5$	$a = 7$ $b = 4$ $c = 4$
t_6	t_7	t_8	t_9	t_{10}
$a = 1$ $b = -2$ $c = 1$	$a = -4$ $b = -9$ $c = 1$	$a = 0$ $b = 3$ $c = 5$	$a = 7$ $b = 2$ $c = 0$	$a = 2$ $b = 9$ $c = 0$

S#		t_1	t_2	t_3	t_4	t_5	t_6	t_7	t_8	t_9	t_{10}
1	double Test(double a, double b, int c){	•	•	•	•	•	•	•	•	•	•
2	double sum = a + b;	•	•	•	•	•	•	•	•	•	•
3	double div = 1;	•	•	•	•	•	•	•	•	•	•
4	double max = b;	•	•	•	•	•	•	•	•	•	•
5	if ((a > 0) && (b > 0))	•	•	•	•	•	•	•	•	•	•
6	div = a / b;	•	•	•	•	•	•	•	•	•	•
7	if (a > b)	•	•	•	•	•	•	•	•	•	•
8	max = b; //Correct: max = a	•	•	•	•	•	•	•	•	•	•
9	if (c == 0)	•	•	•	•	•	•	•	•	•	•
10	return sum;	•	•	•	•	•	•	•	•	•	•
11	else if (c == 1)	•	•	•	•	•	•	•	•	•	•
12	return div;	•	•	•	•	•	•	•	•	•	•
12	else return max;	•	•	•	•	•	•	•	•	•	•
Verified test results		F	F	F	F	F	P	P	P	P	P

Fig. 2. An example program with one fault and a set of ten test cases.

TABLE II

Suspiciousness scores and ranking list of statements for the example program given in Fig. 2 obtained using Ochiai, Op2 formulas in two situations: with and without CC test cases.

S#	Ochiai				Op ²			
	With CC		Without CC		With CC		Without CC	
	Susp(s_i)	Rank	Susp(s_i)	Rank	Susp(s_i)	Rank	Susp(s_i)	Rank
1	0.707	5	0.845	5	4.167	5	4.333	5
2	0.707	5	0.845	5	4.167	5	4.333	5
3	0.707	5	0.845	5	4.167	5	4.333	5
4	0.707	5	0.845	5	4.167	5	4.333	5
5	0.845	2	0.913	2	4.667	2	4.667	2
6	0.707	5	0.845	5	4.167	5	4.333	5
7	0.791	3	1	1	4.5	3	5	1
8	0.707	5	0.845	5	4.167	5	4.333	5
9	0	11	0	11	-0.333	11	-0.333	11
10	0.791	3	0.913	2	4.5	3	4.667	2
11	0	11	0	11	-0.333	11	-0.333	11
12	0.913	1	0.913	2	4.833	1	4.667	2

As shown in Fig. 2, the three passing test cases (i.e., t_6 , t_7 and t_9) are CC test cases since all of them execute the 7th statement that is faulty but produce correct program outputs. As shown in Table II, after removing CC test cases, the rank of the fault statement outputted by *Ochiai* and *Op²* becomes the first from the third. Thus, this example illustrates that the existence of CC test cases have an adversely influence on FL effectiveness.

III. APPROACH

A. Overview

This section presents the methodology of how our approach NeuralCCD detects CC test cases. Fig. 3 shows the overview architecture of our approach, which contains three main parts, namely source data collection, CC detection features calculation, and CC test cases detection. In the first part, NeuralCCD collects the raw data to construct the information model in Fig. 1; In second part, NeuralCCD will extract three CC features. Finally in the third part, the three CC features are utilized as the input to train an integrated model which is used to evaluate the possibility of each passing test case being a CC test case.

B. Source data collection

The first part of NeuralCCD is source data collection. Let us use the faulty program P and the test suite T defined in Section II-A, i.e., P with N statements $\{s_1, s_2, \dots, s_N\}$ running against the test suite T with M test cases $\{t_1, t_2, \dots, t_M\}$. We collect the execution information of all test cases and obtain the information model of FL, which is comprised of the $M \times N$ coverage matrix (denoted as X) and the result vector e that we have referred in Fig. 1. We utilize the $M \times N$ matrix X and the error vector e corresponding to each test case as the input of our approach.

C. CC detection features calculation

Before describing the second part of our approach, we formally define the data used in this section as follows:

Passing test case set T_p : The set of passing test cases in the test suite T , where a test case $t_i \in T_p$ and t_p satisfies that it is executed by the program P and behaves as expected (i.e., its test result $e_p = 0$).

Failing test case set T_f : The set of failing test cases in the test suite T , where a test case $t_f \in T_f$ and t_f satisfies that it is executed by the program P and behaves as unexpected (i.e., its test result $e_f = 1$).

Coincidental correctness element set cc_e : The set of program elements (e.g., statements used in our study) in the program P , where $e_i \in cc_e$ and e_i satisfies that it is executed by all failing test cases and a non-zero but not too large percentage (referred as $\theta \in [0.6, 0.9]$, following the prior work [22]) of passing test cases.

When the data flows to the second part of our approach, we firstly perform test case classification by acquiring T_p and T_f . Based on T_p and T_f , we obtain the set cc_e of the program P , where the program elements of cc_e are statements in our study. After acquiring the above three types of information, we evaluate the suspiciousness score of each statement of cc_e using the ten optimal formulas that we have depicted in Table I since it will reduce the bias effect by using ten formulas rather than using one. With the suspiciousness scores of each statement of cc_e , we define and compute the three CC detection features of all passing test cases, which will be used as the input of our neural network. The three CC detection features are as follows:

suspiciousness score feature (ssf). Based on the definition of cc_e , each statement of cc_e is executed by all failing test cases, meaning that each statement of cc_e has a high suspiciousness score of being faulty statement [32], [33]. Specifically, the passing test cases executing the statements of cc_e are more likely to be CC test cases since these passing test cases are more likely to execute the faulty statement. Thus, we include this feature in CC detection by computing the suspiciousness score feature of each passing test case t_p in T_p as follows:

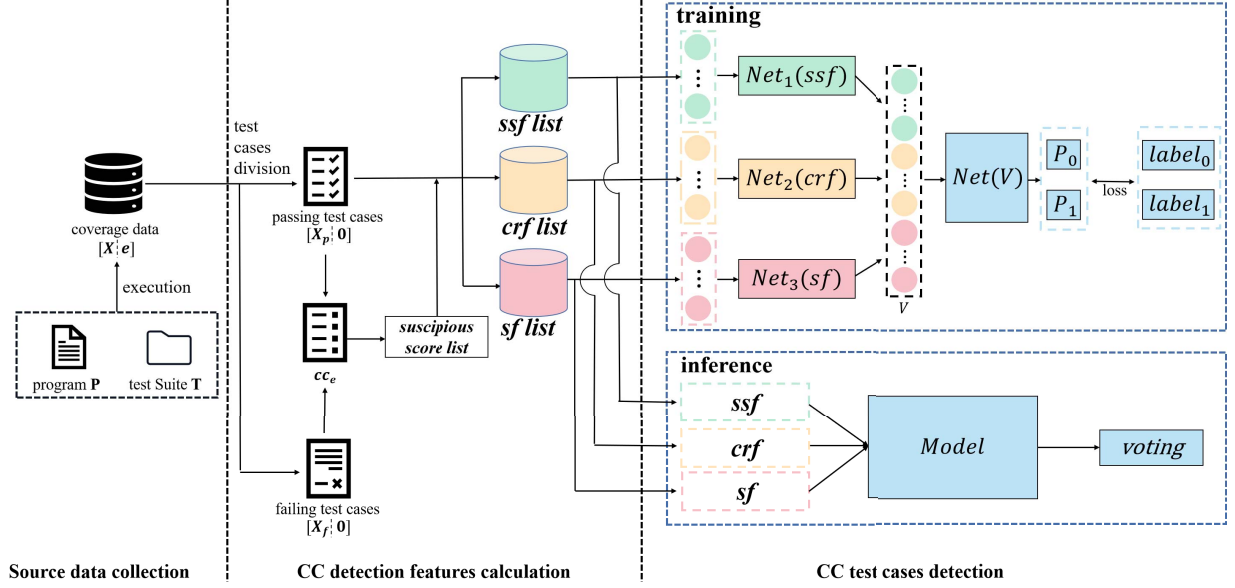


Fig. 3. Overview architecture of NeuralCCD

$$ssf(t_p) = \begin{cases} \frac{\sum_{s_b \in C_b} Sus(s_b)}{|C_b|} & |C_b| > 0 \\ \frac{\sum_{s_s \in C_s} Sus(s_s)}{|C_s|} & |C_b| = 0 \end{cases} \quad (1)$$

Where, C_b is a subset of cc_e , and a statement $s_b \in C_b$ denotes that s_b is executed by t_p and its suspiciousness score is in the range of $[0.5, 1]$; C_s is also a subset of cc_e , and a statement $s_s \in C_s$ denotes that s_s is executed by t_p and its suspiciousness score is in the range of $[0, 0.5]$; $Sus()$ performs the suspiciousness evaluation by using the ten optimal SFL formulas in Table I.

Coverage ratio feature (crf). It is intuitive that a passing test case executing more statements of cc_e whose suspiciousness score are high are more suspicious to be a CC test cases for that it is more likely to cover a buggy element [24]. Thus, we include and define the coverage ratio feature and compute the crf of a passing test case t_p in T_p as follows:

$$crf(t_p) = \frac{|C_{t_p}|}{|C|} \quad (2)$$

Where, C is a subset of cc_e , and a statement in C represents that the the suspiciousness score of t statement is in the range of $[0.5, 1]$; C_{t_p} is a subset of C and a statement in C_{t_p} represents that the statement is executed by t_p .

Similarity feature (sf). The similarity of coverage information between CC test cases and failing test cases is another feature should be involved to describe CC test cases. A passing test case is more likely to be a CC test case if it has similar coverage information on cc_e with failing test cases. We utilize

the following formula to compute the similarity feature of a passing test case t_p :

$$sf(t_p) = \frac{1}{\min \left(\sqrt{\sum_{j=1}^n (s_{pj} \cdot c_p(cc_{e_j}) - s_{fj} \cdot c_f(cc_{e_j}))^2} \right)} \quad (3)$$

Where, n represents the number of the statements of cc_e ; s_{pj} and s_{fj} denotes the suspiciousness scores of the j -th statement in cc_e (denoted as cc_{e_j}) that executed by the passing test case t_p in T_p and the failing test case t_f in T_f respectively; the $\min()$ function returns the square root of the minimum distance (i.e., Euclidean distance [34]) between the passing test case t_p and the failing test case t_f by traversing t_f in T_f ; c_i is a binary function defined as follows:

$$c_i(cc_{e_j}) = \begin{cases} 1 & \text{if } cc_{e_j} \text{ is executed by the test case } t_i, \\ 0 & \text{otherwise.} \end{cases} \quad (4)$$

For each passing test case t_p , NeuralCCD acquires ten values of each feature by adopting each of the then SFL optimal formulas in Table I respectively. Then, NeuralCCD combines them as three 1×10 feature lists: **ssf list**, **crf list** and **sf list**. Finally, we utilize the three feature list of all passing test cases as the input of the third part of our approach NeuralCCD.

D. CC test case detection

This section introduces how to train a multi-layer perceptron based on integrating multiple features, which is denoted as MFMLP and utilize the trained model to detect the CC test cases. It is worth noting that we choose multi-layer perceptron as the neural network in our model just in order to explore

a way of integrating multiple CC features and we will try to design a more clever network in the future. Fig. 3 depicts the general architecture of our CC detection model (*i.e.*, MFMLP) trained by multi-layer perceptron. The neural network (*i.e.*, multi-layer perceptron) consists of one input layer, four network components Net_1 , Net_2 , Net_3 , Net and one output layer.

In the input layer, the three feature lists (*i.e.*, *ssf list*, *crf list* and *sf list*), which we acquire from the second part of our approach in Section III-C for each passing test case t_p in T_p , are used as the input.

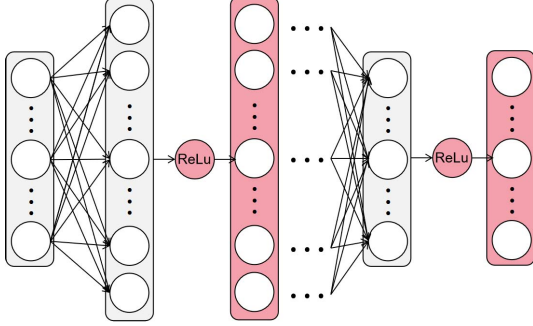


Fig. 4. Architecture of Net_1, Net_2 and Net_3

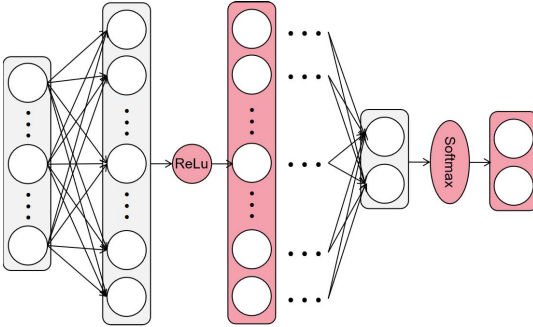


Fig. 5. Architecture of Net

Net_1 , Net_2 , Net_3 , which are depicted in the Fig. 4, respectively extract the information of the three feature list; and the Net depicted in Fig. 5 utilizes the combination of the outputs from them to give the result vector that indicates the probability of a test case being CC test case. The four network components are comprised of several fully connected layers and activation functions. And Net_1 , Net_2 , and Net_3 have the same structure, which are different from that of Net . Two activation functions, which are often used in deep learning [35], are utilized in the four $Nets$: $ReLU$ function and $Softmax$ function. As shown in Eq 5, x is a variable, $ReLU$ function returns the value of x itself if x is greater than 0; otherwise it returns 0. $ReLU$ function has a advantage that it does not suffer from the gradient vanishing problem. The $Softmax$ function returns the probabilities of each value in the input vector being the largest one. $Softmax$ function calculates

the probabilities using the formula depicted in Eq. 6. The benefit of the $Softmax$ function is that it can magnify the gap between the elements in the vector through using exponential function. For Net_1, Net_2 and Net_3 , we choose $ReLU$ function as the activation function utilized in each layer since it does not suffer from gradient vanishing problem. In Net , we use $ReLU$ function in the ordinary layers but $Softmax$ function in the output layer to guarantee that the output vector matches our target vector. The target vector is constructed based on the CC labels of each passing test case t_f . If a t_f is CC test case, its CC label will be 1; otherwise 0. We utilize the mapping method shown in Table III to construct the target vector. In the output layer, both of the two elements in the result vector of the $Softmax$ function have differences with their corresponding elements in the target vector. The difference is the loss between the output and target.

$$ReLU(x) = \begin{cases} x & \text{if } x \geq 0, \\ 0 & \text{otherwise.} \end{cases} \quad (5)$$

$$Softmax(x_i) = \frac{e^{(x_i)}}{\sum_{k=1}^K e^{(x_k)}} \quad (6)$$

TABLE III
Mapping method

CC Label	Target Vector
0	[1,0]
1	[0,1]

We utilize back propagation algorithm [35] to fine-tune the parameters of the model. The algorithm is widely used to adjust the weights of the neurons through calculating the gradient of the loss function in deep learning area. The goal is to minimize the loss given by the loss function between the output vector and the target vector. It computes the gradient of the loss function for each neuron and updates the weights according to the gradient direction. We utilize the $MSELoss$ function in our model, which is shown in Eq. 7. The speed of convergence depends on the learning rate. Our model takes advantages of dynamic adjusting learning rate for its two benefits. On the one hand, it can make larger changes at the beginning of the training process when using larger learning rate values. On the other hand, it can dynamically lower the learning rate to increase the speed of resulting in accurate weights. The experimental results are consistent with the validity of dynamically adjusting the learning rate.

$$MSELoss(x, y) = \frac{1}{n} \sum_{i=1}^N (x_i - y_i)^2 \quad (7)$$

Our approach NeuralCCD seeks to train different models and combine them as an ensemble learning system. We use the voting strategy to increase the accuracy and stability when detecting CC test cases. The voting algorithm is depicted in the Algorithm 1. We first initialize the T_{CC} and ModelList as empty sets (lines 2-3), and then we apply cc_e to coverage

matrix to construct a new matrix (line 5). After that, we compute suspiciousness list with ten suspiciousness scores and use them to compute the three features of CC test cases (lines 6-8) by applying the mean variance normalization (lines 36-43). NeuralCCD changes and determines the CC labels (line 10). We utilize the three feature lists and label list to train a CC feature-based model (*i.e.*, MFMLP function) for each program (line 11). After that, we traverse the N program and each time utilize the N-1 models that trained by the other N-1 program data to predict if the passing test cases of the program being traversed are CC test cases (lines 13-33). If the number of supporting are larger than the opposite, the test case will be detected as CC test case.

Finally, we summarize the steps of NeuralCCD as follows:

- 1) NeuralCCD divides the coverage data and execution results of the test cases shown in Fig. 3 into passing test cases T_p and failing test cases T_f . Based on T_p and T_f , we construct cc_e for the program P .
- 2) NeuralCCD uses ten suspiciousness score formulas depicted in Table I and all the test cases in T_p and T_f to compute the suspiciousness scores for each statement of cc_e . Then it uses the suspiciousness scores of the statements of cc_e to compute the three features of CC test cases for all passing test cases. Finally, it combines them into three feature lists for each passing test case: $ssf list$, $crf list$, $sf list$.
- 3) NeuralCCD takes the three feature lists of each program used as training sample set and use a neural network to train a model for each program. NeuralCCD then traverses the programs and combines the models trained by other programs as an ensemble system with a vote strategy to detect the CC test cases in the program being traversed.

IV. EMPIRICAL EVALUATION

A. Datasets

To evaluate the effectiveness of NeuralCCD, we conduct the experiments on 247 faulty versions (*i.e.*, real faults) of five representative benchmark programs. They are Chart, Lang, Math, Time and Mockito from Defects4J¹. It is time-consuming to collect inputs since the programs of Defects4J are large-sized. Thus, we reuse the coverage matrix collected by Pearson *et al.* [39]

B. Experimental Settings

We ran the experiments on a Linux server with 40 cores of 2.4GHz CPU, 252GB RAM and two NVIDIA GeForce RTX 3090 GPUs with 24 GB memory. The operating system is Ubuntu 20.04.

C. Evaluation Metrics

Before introducing the evaluation metrics, we give the definition of the data used in this section formally as follows:

CC test cases set T_{CC} : The set of total CC test cases in a program.

¹<https://github.com/rjust/defects4j>

Algorithm 1: The voting algorithm in NeuralCCD

Input: Coverage information of N programs: $X_1, \dots, X_N$,
Result vector y of N programs: $y_1, \dots, y_N$,
CC Labels of N programs: $\{y_{c_1}, \dots, y_{c_N}\}$

Output: CC test cases : T_{CC}

```

1: function MAIN
2:  $T_{CC} \leftarrow \emptyset$ 
3:  $ModelList \leftarrow \emptyset$ 
4: for  $i = 1$  to  $N$  do
5:    $X' \leftarrow$  applying  $cc_e$  to coverage matrix  $X$ 
6:    $ssList \leftarrow$  calculate ten suspiciousness scores of  $X'$ 
     by ten formulas
7:    $ssf list_i = \text{standardize}(\text{suspiciousScore}(X', ssList))$ 
8:    $crf list_i = \text{standardize}(\text{coverRatio}(X', ssList))$ 
9:    $sf list_i =$ 
      $\text{standardize}(\text{similarityFeature}(X', ssList))$ 
10:   $y_c list_i \leftarrow$  the mapping vector of  $y_{c_i}$  following the
     method in Table III
11:   $ModelList \leftarrow$ 
      $MFMLP(ssf list_i, crf list_i, sf list_i, y_c list_i)$ 
12: end for
13: for  $i$  in  $\text{range}(\text{len}(ModelList))$  do
14:    $NewModelList \leftarrow ModelList - ModelList[i]$ 
15:    $resultList \leftarrow \emptyset$ 
16:   for  $model$  in  $NewModelList$  do
17:      $resultList \cup \text{model}(ssf list_i, crf list_i, sf list_i)$ 
18:   end for
19:   for  $j$  in  $\text{range}(\text{len}(ssf list_i))$  do
20:      $cc_{num} = 0$ 
21:      $ncc_{num} = 0$ 
22:     for  $k$  in  $\text{range}(\text{len}(resultList))$  do
23:       if  $resultList[k][j][0] = 1$  then
24:          $cc_{num} ++$ 
25:       else
26:          $ncc_{num} ++$ 
27:       end if
28:     end for
29:     if  $cc_{num} \geq ncc_{num}$  then
30:        $T_{CC} \cup j$ 
31:     end if
32:   end for
33: end for
34: return  $T_{CC}$ 
35: end function
36: function STANDARDIZE( $featureList$ )
37: for  $i$  in  $\text{range}(\text{len}(featureList))$  do
38:    $\mu \leftarrow$  the mean value of  $featureList[i]$ 
39:    $s \leftarrow$  the standard deviation of  $featureList[i]$ 
40:    $featureList[i] = (featureList[i] - \mu) / s$ 
41: end for
42: return  $featureList$ 
43: end function

```

TABLE IV
Characteristics of subject programs

Program	Description	Versions	LoC (k)	Test
Chart	Java chart library	26	96	2205
Lang	Apache commons-lang	65	22	2245
Math	Apache commons-math	106	85	3602
Time	Standard date and time library	27	28	4130
Mockito	Mocking framework for Java	23	67	1075
total	-	247	298	13257

Detected CC test cases set T_{dCC} : The set of CC test cases that detected by our approach.

We utilize several widely used evaluation metrics in our experiments to examine the performance of NeuralCCD on CC test cases detection and improvement in the validity of FL approaches. And we will show the details of these metrics as following:

1) *Evaluation Metrics for CC detection:* In order to evaluate the effectiveness of our model on CC detection, we use three measurements as follows.

- **Precision:** Precision reflects the ratio of the number of the correctly detected CC test cases to the total number of the CC test cases detected by the model, which is defined in Eq. 8:

$$Precision = \frac{|T_{dCC} \cap T_{CC}|}{T_{dCC}} \quad (8)$$

- **Recall:** Recall reflects the ratio of the number of the correctly detected CC test cases to the total number of the CC test cases, which is defined in Eq. 9:

$$Recall = \frac{|T_{dCC} \cap T_{CC}|}{T_{CC}} \quad (9)$$

- **F_1 measure:** F_1 is the combination of **Precision** and **Recall** to balance the two metrics [40]. It is defined in Eq. 10:

$$F_1 \text{ measure} = 2 \cdot \frac{Precision \cdot Recall}{Precision + Recall} \quad (10)$$

2) *Evaluation Metrics for FL:* In order to evaluate the performance of our model on improving fault localization, we use the following four measurements.

- **Top-K:** [36]–[38] It indicates the number of faulty versions with at least one faulty statement that is in the first K ($K=1,3,5,\dots$) position in the ranking list given by a FL approach. In the prior work, many researchers consider an FL approach is effective when it can locate the buggy elements in the top 5 positions [2], [10], [21]. Following the previous study [21], we assign K with values 1,3,5 for evaluation.
- **Mean First Rank (MFR):** [17] It firstly calculates the rank of the faulty statement that is locate first in each version and then compute the mean values of the ranks in the project.

- **Wilcoxon signed-rank tests (WSR):** [41] Following the prior work [3], [4], [7], we utilize Wilcoxon signed-rank tests to evaluate the statistical significance of the results.

D. Research Questions and Results

We evaluate the effectiveness of our approach through the following three research questions. We use four representative CC test case detection approaches as the baselines and compare them with NeuralCCD. The four CC test case detection approaches are depicted as follows:

- Masri *et al.* [18] propose two approaches named **Tech-I** and **Tech-II**, which are both based on cluster algorithm. Dasset *et al.* [25] present **RF**, which utilizes ensemble random forests classifier on CC detection. Yu *et al.* [28] present **CBCFL**, an approach that uses failure context on improving K-means cluster analysis.
- **CBCFL** makes its source code publicly available, while **Tech-I**, **Tech-II** and **RF** have no public source code and thus we reimplement the models based on the paper.

RQ1. How does NeuralCCD perform in detecting CC test cases as compared with the state-of-the-art CC detection approaches?

For RQ1, Table V shows the results of the three metrics (*i.e.*, recall, precision, F1) of the four baselines and our approach NeuralCCD. For the convenience of reading, we bold the best results. Table V shows that NeuralCCD performs better than the four baselines on precision when detecting CC test cases. Take Lang as an example. When using the three metrics, NeuralCCD has increased by 55% as compared with the baseline of **Tech-I**, which performs best among the baselines on CC detection precision. The experimental results of the three metrics show that NeuralCCD can effectively improve the precision of CC detection. Considering an approach with high recall value but low precision value on CC detection, when utilized on fault localization, it will lose much useful information, which may decrease the localization performance. Conversely, NeuralCCD may perform well for its high precision.

Summary for RQ1: In RQ1, we discuss the effectiveness of the NeuralCCD compared with that of the four CC detection approaches. Experimental results indicate that NeuralCCD performs better in CC detection. It shows that NeuralCCD is effective to improve the precision of detecting CC test cases with the CC feature extraction and deep learning.

RQ2. How effective is NeuralCCD as compared with the state-of-the-art CC detection approaches when utilized on fault localization?

For RQ2, we use five metrics (Top-1, Top-3, Top-5, MFR and WSR) to compare the performance of effective FL approaches after applying NeuralCCD and other four state-of-the-art CC detection techniques. Since the prior study [39] has empirically shown that Dstar, Ochiai, Barinel are the most effective ones in locating real faults and our benchmark programs are all real faults, we include the three FL approaches

TABLE V
CC detection results

Program	Approach	Recall	Precision	F1
Chart	<i>Tech-I</i>	96%	35%	52%
	<i>Tech-II</i>	98%	24%	38%
	RF	96%	13%	24%
	CBCFL	58%	18%	28%
	NeuralCCD	32%	67%	43%
Lang	<i>Tech-I</i>	70%	38%	49%
	<i>Tech-II</i>	73%	34%	47%
	RF	89%	15%	26%
	CBCFL	61%	29%	40%
	NeuralCCD	19%	93%	32%
Math	<i>Tech-I</i>	77%	43%	55%
	<i>Tech-II</i>	68%	19%	30%
	RF	95%	15%	26%
	CBCFL	56%	23%	33%
	NeuralCCD	16%	60%	25%
Mockito	<i>Tech-I</i>	82%	34%	48%
	<i>Tech-II</i>	81%	37%	51%
	RF	99%	32%	48%
	CBCFL	86%	42%	57%
	NeuralCCD	13%	47%	20%
Time	<i>Tech-I</i>	90%	4%	7%
	<i>Tech-II</i>	86%	4%	8%
	RF	99%	3%	6%
	CBCFL	84%	5%	10%
	NeuralCCD	30%	35%	32%

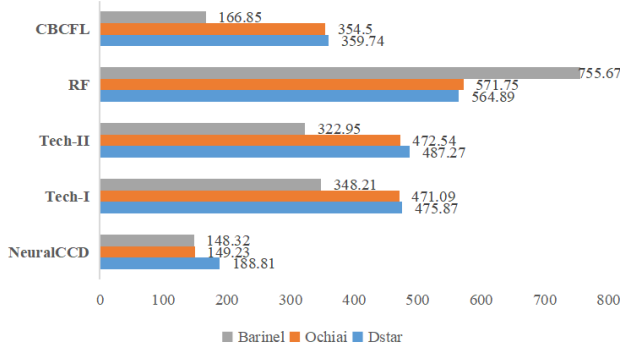


Fig. 6. MFR results of applying NeuralCCD and the four CC detection approaches on FL

to apply NeuralCCD and other four CC detection techniques. We use relabel strategy, which indicates changing the labels of CC test cases as failing labels, following the prior study [27], [43], [44].

Table VI shows the results of the three FL approaches applying NeuralCCD and the four state-of-the-art CC detection approaches. Take Math using Dstar as an example. The number of faults ranked with Top-1, Top-3 and Top-5 is 5, 8, 9 by applying *Tech-I* and is 8, 18, 19 by using NeuralCCD, meaning that NeuralCCD increases 60%, 125% and 111% as compared with *Tech-I*.

Fig. 6 shows the MFR values of three FL approaches after applying NeuralCCD and the four CC detection baselines. As shown in Fig. 6, NeuralCCD obtains the lowest MFR values on FL approaches in comparison to the four CC detection baselines, meaning that NeuralCCD performs the best on FL.

Table VII shows the result of WSR results of MFR metric of

Dstar after applying NeuralCCD and other four CC detection approaches. The significant level σ is 0.05. Taking Chart as an example (NeuralCCD vs. *Tech-I*), the p values of 2-tailed, 1-tailed(right) and 1-tailed(left) are 3.51E-03, 9.98E-01 and 1.76E-03 respectively. It means that the MFR of Dstar after applying NeuralCCD is significantly less than that of Dstar after applying *Tech-I*. Thus, we gain a Better conclusion, which indicates NeuralCCD performs better than *Tech-I* on improving the Dstar effectiveness in Chart. We can observe that NeuralCCD gains Better results over *Tech-I* and RF in all programs. For *Tech-II* and CBCFL gain Better in most programs, except for the Similar result occurred in Math and Mockito. The experimental results above show that NeuralCCD can significantly improve the effectiveness of FL.

Summary for RQ2: In RQ2, we discuss the effectiveness of NeuralCCD as compared with the four state-of-the-art CC detection approaches when applied for fault localization. The experimental results of Top-K, MFR and WSR show that NeuralCCD significantly outperforms the four state-the-art CC detection approaches when applied for fault localization.

RQ3. Does each component contributes to the effectiveness of NeuralCCD?

For RQ3, we analyze the impact of each component in NeuralCCD. To prove the validity of each component in NeuralCCD, we remove suspiciousness score feature, coverage ratio feature and similarity feature respectively and after that compare their performances in CC detection with NeuralCCD. We denote them as $NeuralCCD_{ssf-}$, $NeuralCCD_{crf-}$, $NeuralCCD_{sf-}$ respectively.

Table VIII shows the CC detection results of removing each component respectively. For example, without similarity feature, the performance of NeuralCCD on recall sharply decreases. The recall values of $NeuralCCD_{sf-}$ on all programs decrease under than 10%. All the results show that NeuralCCD performs better than $NeuralCCD_{ssf-}$, $NeuralCCD_{crf-}$, $NeuralCCD_{sf-}$, which means the three CC detection features are all useful when detecting CC test cases. Furthermore, we find that $NeuralCCD_{sf-}$ performs worst among them while $NeuralCCD_{crf-}$ performs comparatively the best, showing that similarity feature has the highest effectiveness on CC detection and coverage ratio feature has the lowest effectiveness.

Summary for RQ3: In RQ3, we discuss whether each component of NeuralCCD contributes to its CC detection effectiveness. The results show all the three CC detection features are useful in NeuralCCD, and similarity feature has the highest effectiveness in CC detection.

V. DISCUSSION

A. Threats to Validity

Implementation of baselines and our approach. The baselines that we implement may not entirely correct and have randomness themselves. On the one hand, we cannot guarantee

TABLE VI
Top-1,Top-3,Top-5 and MFR results of applying NeuralCCD and four CC detection approaches on FL.

program	Approach	Top-1			Top-3			Top-5			MFR		
		Dstar	Ochiai	Barinel	Dstar	Ochiai	Barinel	Dstar	Ochiai	Barinel	Dstar	Ochiai	Barinel
Chart	<i>Tech-I</i>	2	2	2	2	2	2	4	4	4	783.26	725.68	469.11
	<i>Tech-II</i>	2	2	1	2	2	2	3	3	4	878.84	831.00	504.53
	RF	0	0	0	0	0	0	0	0	1	1032.53	1034.74	1529.47
	CBCFL	1	1	1	1	1	1	2	3	3	467.60	455.40	258.27
	NeuralCCD	2	2	2	5	5	3	7	7	4	344.53	284.58	300.05
Lang	<i>Tech-I</i>	1	1	0	7	7	1	13	13	6	78.35	75.48	77.93
	<i>Tech-II</i>	2	2	0	10	10	3	13	13	8	82.65	82.17	82.46
	RF	0	0	0	1	1	1	3	3	1	182.87	213.80	330.24
	CBCFL	0	1	2	0	1	2	1	4	5	77.08	77.00	55.96
	NeuralCCD	2	2	2	6	6	4	15	17	12	22.57	22.43	37.39
Math	<i>Tech-I</i>	5	5	3	8	8	3	9	9	6	237.99	250.71	253.58
	<i>Tech-II</i>	3	3	2	6	6	4	8	8	6	220.39	224.43	220.61
	RF	2	2	0	3	3	0	4	4	0	267.36	267.88	391.50
	CBCFL	1	2	6	1	2	6	1	2	6	227.73	226.51	168.37
	NeuralCCD	8	8	7	18	17	9	19	19	12	114.95	127.30	157.50
Mockito	<i>Tech-I</i>	0	0	0	1	1	2	1	1	2	481.73	470.90	519.53
	<i>Tech-II</i>	0	0	0	2	2	2	2	2	2	510.60	455.77	421.53
	RF	1	2	2	1	2	2	1	2	3	503.30	508.03	1044.40
	CBCFL	1	1	0	2	2	2	2	2	3	312	302.84	143.24
	NeuralCCD	3	3	2	5	5	5	6	6	5	282.53	209.57	176.3
Time	<i>Tech-I</i>	0	0	0	0	0	0	0	0	0	1802.05	1792.10	838.90
	<i>Tech-II</i>	0	0	0	1	1	1	1	1	1	1779.70	1792.10	838.80
	RF	0	0	0	0	0	0	0	0	0	1920.50	1928.10	1710.20
	CBCFL	0	0	0	0	0	0	0	0	1	1090.18	1080.29	262.12
	NeuralCCD	0	0	1	1	1	4	1	1	5	509.75	281.50	176.00
Total	<i>Tech-I</i>	8	8	5	10	10	8	27	27	18	475.87	471.09	348.21
	<i>Tech-II</i>	7	7	3	20	20	13	27	27	21	487.27	472.54	322.95
	RF	3	4	2	5	6	3	11	12	8	564.89	571.75	755.67
	CBCFL	3	5	9	4	6	11	6	11	18	359.74	354.5	166.85
	NeuralCCD	15	15	14	35	34	25	48	50	38	188.81	149.23	148.32

TABLE VII
WSR results on FL.

program	Comparision	2-tailed	1-tailed(right)	1-tailed(left)	Conclusion
Chart	<i>NeuralCCD vs Tech-I</i>	3.51E-03	9.98E-01	1.76E-03	Better
	<i>NeuralCCD vs Tech-II</i>	6.83E-02	9.66E-01	3.42E-02	Better
	<i>NeuralCCD vs RF</i>	7.63E-06	1.00E+00	3.81E-06	Better
	<i>NeuralCCD vs CBCFL</i>	1.56E-02	9.5E-01	7.81E-02	Better
Lang	<i>NeuralCCD vs Tech-I</i>	1.59E-02	9.92E-01	7.93E-03	Better
	<i>NeuralCCD vs Tech-II</i>	2.16E-04	1.00E+00	1.08E-04	Better
	<i>NeuralCCD vs RF</i>	1.21E-04	1.00E+00	6.03E-05	Better
	<i>NeuralCCD vs CBCFL</i>	3.13E-02	1.00E+00	1.56E-02	Better
Math	<i>NeuralCCD vs Tech-I</i>	3.49E-03	9.98E-01	1.74E-03	Better
	<i>NeuralCCD vs Tech-II</i>	3.49E-01	8.25E-01	1.75E-01	Similar
	<i>NeuralCCD vs RF</i>	3.84E-04	1.00E+00	1.92E-04	Better
	<i>NeuralCCD vs CBCFL</i>	2.5E-01	1.00E+00	1.25E-01	Similar
Mockito	<i>NeuralCCD vs Tech-I</i>	3.13E-03	9.98E-01	1.56E-03	Better
	<i>NeuralCCD vs Tech-II</i>	9.59E-01	5.21E-01	4.79E-01	Similar
	<i>NeuralCCD vs RF</i>	6.16E-04	1.00E+00	3.08E-04	Better
	<i>NeuralCCD vs CBCFL</i>	2.29E-01	8.94E-01	1.14E-01	Similar
Time	<i>NeuralCCD vs Tech-I</i>	1.32E-04	1.00E+00	6.59E-05	Better
	<i>NeuralCCD vs Tech-II</i>	1.55E-04	1.00E+00	7.75E-05	Better
	<i>NeuralCCD vs RF</i>	1.91E-06	1.00E+00	9.54E-07	Better
	<i>NeuralCCD vs CBCFL</i>	7.63E-06	1.00E+00	3.81E-06	Better

TABLE VIII
CC detection results of NeuralCCD via removing each component

Program	Approach	Recall	Precision	F1
Chart	<i>NeuralCCD_{ssf-}</i>	17%	90%	29%
	<i>NeuralCCD_{crf-}</i>	0%	NaN	NaN
	<i>NeuralCCD_{sf-}</i>	5%	71%	9%
	<i>NeuralCCD</i>	32%	67%	43%
Lang	<i>NeuralCCD_{ssf-}</i>	17%	92%	29%
	<i>NeuralCCD_{crf-}</i>	27%	91%	42%
	<i>NeuralCCD_{sf-}</i>	7%	91%	13%
	<i>NeuralCCD</i>	19%	93%	32%
Math	<i>NeuralCCD_{ssf-}</i>	14%	59%	23%
	<i>NeuralCCD_{crf-}</i>	17%	62%	27%
	<i>NeuralCCD_{sf-}</i>	3%	65%	6%
	<i>NeuralCCD</i>	16%	60%	25%
Mockito	<i>NeuralCCD_{ssf-}</i>	3%	73%	6%
	<i>NeuralCCD_{crf-}</i>	12%	46%	19%
	<i>NeuralCCD_{sf-}</i>	1%	87%	2%
	<i>NeuralCCD</i>	13%	47%	30%
Time	<i>NeuralCCD_{ssf-}</i>	11%	42%	17%
	<i>NeuralCCD_{crf-}</i>	20%	42%	27%
	<i>NeuralCCD_{sf-}</i>	3%	34%	6%
	<i>NeuralCCD</i>	30%	35%	32%

The value NaN (Not a Number) indicates that the model does not detect any CC test case, which lead to the denominator in Precision formula being 0.

that our implement of the baselines are in full compliance with the implementations in their experiments. On the other hand, The baselines themselves have randomness. For **Tech-I** and **Tech-II**, the initial cluster centers for K-means are randomly generated which will influence the results of the cluster based technique. As for **RF**, the initial data split is randomly executed. Except that, we cannot get the parameter values of the random forest used by the author.

Generalization of our approach. We apply our approach on five representative programs. There may exist some unknown factors in other datasets. They may cause some potential generalization issues that our approach will not perform well on some datasets with special data features.

B. Reasons for NeuralCCD is effective and Generalizable

The reasons why NeuralCCD is superior to the baselines on the effectiveness and generalization are as follows: a) The coverage information of cc_e can reflect the inner differences of the passing and failing test cases more effectively. b) We consider the inner features of CC test cases. The CC test cases have the similar inner features of each other, and such features of them are different from that performed by passing and failing test cases. c). We use ensemble strategy on our approach to increase the generalization of our model.

VI. RELATED WORK

A. CC detection techniques based on heuristic algorithms.

The group of techniques utilize the heuristic algorithms to detect the CC test cases. Masri and Assi [18] propose a techniques which is referred Technique-I. They firstly identify a special set, which is called CCE by them, of program entities that are likely to be correlated with CC test cases.

The elements of the set, which are called cc_e , are executed by all failed test cases and a non-zero but not too large percentage of passed test cases, which is limited by a value θ . They then traverse all passed test cases to find the test cases that execute the elements in the CCE as their estimate of CC.

B. CC detection techniques based on machine learning algorithms.

This group of techniques utilize various machine learning algorithms on CC detection, where clustering-based techniques and supervised algorithms based ones are two main types.

In clustering-based techniques, the test cases are divided into different clusters bases on the differences of their execution on the program entity. Miao [42], [45] use the coverage matrix as the input of cluster analysis and regard the test cases that are grouped into the failed test cases cluster as the CC test cases. And a clustering approach that identifies CC test cases repressively, is proposed by Yang [43]. Masri and Assi also propose a cluster-bases technique in [22], which is referred *Technique-III*. The technique clusters the passed test cases into two clusters. They consider the elements in the cluster that has the higher average weight of suspicious cc_e as the CC test cases. Most of the techniques employ $K - Means$ algorithm.

Despite the inspiring performance of cluster-based CC detection techniques, choosing a reasonable cluster number has been a resistance on the improvement of such techniques. It has been demonstrated that a larger number of clusters will consequently cause a higher rate of false negatives but a lower rate of false positives [42], [45]. Such problem can be solved by the supervised algorithms based CC detection techniques. In this aspect, Xue et al. [23] adapt ensemble-based support vector machines [46] to detect CC tests and Feyzi and Parsa also take advantage of SVM with a customized kernel function. Dass et al. [25] try to apply random forests classifier on CC detection. They also use ensemble learning to improve the accuracy of their technique.

VII. CONCLUSION

In this paper, we propose NeuralCCD which utilizes the three features of CC test cases and a deep neural network to train a CC detection model. Specifically, we first extract the cc_e from all the statements to define three CC detection features; then use three features and CC labels as input and output to train a CC detection model based on multi-layer perceptron; finally we conduct large-scale experiments to evaluate NeuralCCD. The experimental results show that our approach is effective for CC detection.

In future work, we intend to extract and define more CC detection features, and explore more on deep learning to extend our approach.

ACKNOWLEDGMENTS

This work is partially supported by the National Natural Science Foundation of China (No. 62272072), the Fundamental Research Funds for the Central Universities (No. 2022CDJDX-005), and the Major Key Project of PCL (No. PCL2021A06).

REFERENCES

- [1] M. Papadakis and Y. Le Traon, "Metallaxis-FL: mutation-based fault localization," *Software Testing, Verification and Reliability*, vol. 25, no. 5-7, pp. 605-628, 2015.
- [2] J. Sohn and S. Yoo, "Flucss: Using code and change metrics to improve fault localization," in *Proceedings of the 26th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2017, pp. 273-283.
- [3] X. Li, W. Li, Y. Zhang, and L. Zhang, "DeepFL: Integrating multiple fault diagnosis dimensions for deep fault localization," in *Proceedings of the 28th ACM SIGSOFT international symposium on software testing and analysis*, 2019, pp. 169-180.
- [4] Zhuo, Zhang, Yan, et al. CNN-FL: An Effective Approach for Localizing Faults using Convolutional Neural Networks[C]// 2019 IEEE 26th International Conference on Software Analysis, Evolution and Reengineering (SANER). 0.
- [5] Y. Lou, A. Ghanbari, X. Li, L. Zhang, H. Zhang, D. Hao, and L. Zhang, "Can automated program repair refine fault localization? a unified debugging approach," in *Proceedings of the 29th ACM SIGSOFT International Symposium on Software Testing and Analysis*, 2020, pp. 75-87.
- [6] Lou Y, Zhu Q, Dong J et al. Boosting coverage-based fault localization via graph-based representation learning. 2021.
- [7] Y. Küçük, T. A. Henderson, and A. Podgurski, "Improving fault localization by integrating value and predicate based causal inference techniques," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 649-660.
- [8] Y. Lei, C. Liu, H. Xie, S. Huang, M. Yan, and Z. Xu, "BCL-FL: A data augmentation approach with between-class learning for fault localization," in *2022 IEEE International Conference on Software Analysis, Evolution and Reengineering (SANER)*. IEEE, 2022, pp. 289-300.
- [9] Thomas Repts, Thomas Ball, Manuvir Das, and James Larus. 1997. The use of program profiling for software maintenance with applications to the year 2000 problem. In *Software Engineering—Esec/Fse'97*. Springer, 432-449.
- [10] Y. Li, S. Wang, and T. Nguyen, "Fault localization with code coverage representation learning," in *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 2021, pp. 661-673.
- [11] R. Abreu, P. Zoetewij, and A. J. C. Van Gemund, "Spectrum-based multiple fault localization," in *Proceedings of the 24th IEEE/ACM International Conference on Automated Software Engineering (ASE 2009)*, 2009, pp. 88-99.
- [12] L. Naish, H. J. Lee, and K. Ramamohanarao, "A model for spectra-based software diagnosis," *ACM Transactions on software engineering and methodology (TOSEM)*, vol. 20, no. 3, pp. 1-32, 2011.
- [13] W. Wen, "Software fault localization based on program slicing spectrum," *2012 34th International Conference on Software Engineering (ICSE)*, 2012, pp. 1511-1514, doi: 10.1109/ICSE.2012.6227049.
- [14] W. E. Wong, V. Debroy, R. Gao, and Y. Li, "The dstar method for effective software fault localization," *IEEE Transactions on Reliability*, vol. 63, no. 1, pp. 290-308, 2013.
- [15] Lei, Y., Xie, H., Zhang, T., Yan, M., Xu, Z., and Sun, C. (2022). Feature-FL: Feature-Based Fault Localization. *IEEE Transactions on Reliability*, 71, 264-283.
- [16] Xia Li, Wei Li, Yuqun Zhang, and Lingming Zhang. 2019. DeepFL: integrating multiple fault diagnosis dimensions for deep fault localization. In *Proceedings of the 28th ACM SIGSOFT International Symposium on Software Testing and Analysis (ISSTA 2019)*. Association for Computing Machinery, New York, NY, USA, 169-180.
- [17] H. Xie, Y. Lei, M. Yan, Y. Yu, X. Xia, and X. Mao, "A universal data augmentation approach for fault localization," in *International Conference on Software Engineering (ICSE'22)*, 2022.
- [18] W. Masri and R. A. Assi, "Prevalence of coincidental correctness and mitigation of its impact on fault localization," *ACM transactions on software engineering and methodology (TOSEM)*, vol. 23, no. 1, pp. 1-28, 2014.
- [19] R. Abou Assi, C. Trad, M. Maalouf, and W. Masri, "Coincidental correctness in the defects4j benchmark," *Software Testing, Verification and Reliability*, vol. 29, no. 3, p. e1696, 2019.
- [20] R. Abou Assi, W. Masri, and C. Trad, "How detrimental is coincidental correctness to coverage-based fault detection and localization? an empirical study," *Software Testing, Verification and Reliability*, vol. 31, no. 5, p. e1762, 2021.
- [21] P. S. Kochhar, X. Xia, D. Lo, and S. Li, "Practitioners' expectations on automated fault localization," in *Proceedings of the 25th International Symposium on Software Testing and Analysis*, 2016, pp. 165-176.
- [22] W. Masri and R. Abou Assi, "Cleansing test suites from coincidental correctness to enhance fault-localization," in *2010 third international conference on software testing, verification and validation*. IEEE, 2010, pp. 165-174.
- [23] X. Xue, Y. Pang, and A. S. Namin, "Trimming test suites with coincidentally correct test cases for enhancing fault localizations," in *2014 IEEE 38th Annual Computer Software and Applications Conference*. IEEE, 2014, pp. 239-244.
- [24] Y. Liu, M. Li, Y. Wu, and Z. Li, "A weighted fuzzy classification approach to identify and manipulate coincidental correct test cases for fault localization," *Journal of Systems and Software*, vol. 151, pp. 20-37, 2019.
- [25] S. Dass, X. Xue, and A. S. Namin, "Ensemble random forests classifier for detecting coincidentally correct test cases," in *2020 IEEE 44th Annual Computers, Software, and Applications Conference (COMPSAC)*. IEEE, 2020, pp. 1326-1331.
- [26] F. Feyzi, "CGT-fl: using cooperative game theory to effective fault localization in presence of coincidental correctness," *Empirical Software Engineering*, vol. 25, no. 5, pp. 3873-3927, 2020.
- [27] A. Sabbaghi, M. R. Keyvanpour, and S. Parsa, "Fcci: A fuzzy expert system for identifying coincidental correct test cases," *Journal of Systems and Software*, vol. 168, p. 110635, 2020.
- [28] J. Yu, Y. Lei, H. Xie, L. Fu, and C. Liu, "Context-based cluster fault localization," in *2022 IEEE/ACM 30th International Conference on Program Comprehension (ICPC)*. IEEE, 2022, pp. 482-493.
- [29] Schmidhuber J. Deep learning in neural networks: an overview. *Neural Netw.* 2015 Jan;61:85-117. doi: 10.1016/j.neunet.2014.09.003. Epub 2014 Oct 13. PMID: 25462637.
- [30] Zhuo Zhang, Yan Lei, Xiaoguang Mao, Meng Yan, Ling Xu, and Xiaohong Zhang. 2021. A study of effectiveness of deep learning in locating real faults. *Information and Software Technology* 131 (2021), 106486.
- [31] Wei Zheng, Desheng Hu, and Jing Wang. 2016. Fault localization analysis based on deep neural network. *Mathematical Problems in Engineering* 2016 (2016).
- [32] Miao, Yi, Zhenyu Chen, Sihan Li, Zhihong Zhao and Yuming Zhou. "Identifying Coincidental Correctness for Fault Localization by Clustering Test Cases." *SEKE* (2012).
- [33] Souza, Higor Amario de, Marcos Lordello Chaim and Fabio Kon. "Spectrum-based Software Fault Localization: A Survey of Techniques, Advances, and Challenges." *ArXiv abs/1607.04347* (2016): n. pag.
- [34] Deza, M.M., Deza, E.: Encyclopedia of distances. In: Deza, M.M., Deza, E. (eds.) *Encyclopedia of Distances*, pp. 1-583. Springer, Heidelberg (2009).
- [35] Y. Lecun, Y. Bengio, and G. e. Hinton, "Deep learning," *Nature*, vol. 521, no. 7553, p. 436, 2015.
- [36] Feyzi, F. CGT-FL: using cooperative game theory to effective fault localization in presence of coincidental correctness. *Empir Software Eng* 25, 3873-3927 (2020).
- [37] Samira Rahimyar Heris and MohammadReza Keyvanpour. 2019. Effectiveness of Weighted Neural Network on Accuracy of Software Fault Localization. In *2019 5th International Conference on Web Research (ICWR)*. IEEE, 100-104.
- [38] Yiğit Küçük, Tim AD Henderson, and Andy Podgurski. 2021. Improving fault localization by integrating value and predicate based causal inference techniques. In *2021 IEEE/ACM 43rd International Conference on Software Engineering (ICSE)*. IEEE, 649-660.
- [39] S. Pearson, J. Campos, R. Just, G. Fraser, R. Abreu, M. D. Ernst, D. Pang, and B. Keller, "Evaluating & improving fault localization techniques," *University of Washington Department of Computer Science and Engineering*, Seattle, WA, USA, Tech. Rep. UW-CSE-16-08-03, p. 27, 2016.
- [40] N. Chinchor and B. M. Sundheim, "Muc-5 evaluation metrics," in *Fifth Message Understanding Conference (MUC-5): Proceedings of a Conference Held in Baltimore, Maryland, August 25-27, 1993*, 1993.
- [41] F. Wilcoxon, "Individual comparisons by ranking methods," in *Breakthroughs in statistics*. Springer, 1992, pp. 196-202.
- [42] Miao, Yi, Zhenyu Chen, Sihan Li, Zhihong Zhao and Yuming Zhou. "Identifying Coincidental Correctness for Fault Localization by Clustering Test Cases." *SEKE* (2012).
- [43] Yang, Xiaoshuang, Mengleng Liu, Ming Cao, Lei Zhao and Lina Wang. "Regression detection of Coincidental Correctness via Weighted Clustering." *2015 IEEE 39th Annual Computer Software and Applications Conference* (2015): 115-120.

- [44] Feyzi, F., Parsa, S. A program slicing-based method for effective detection of coincidentally correct test cases. *Computing* 100, 927–969 (2018).
- [45] Miao, Y., Chen, Z., Li, S., Zhao, Z., Zhou, Y., 2013. A clustering-based strategy to identify coincidental correctness in fault localization. In. *J. Softw. Eng. Knowl. Eng.* 23 (05), 721–741.
- [46] Kim, HC., Pang, S., Je, HM., Kim, D., Bang, SY. (2002). Support Vector Machine Ensemble with Bagging. In: Lee, SW., Verri, A. (eds) *Pattern Recognition with Support Vector Machines. SVM 2002. Lecture Notes in Computer Science*, vol 2388. Springer, Berlin, Heidelberg.
- [47] Xie, Xiaoyuan, Kuo, Fei Ching, Chen, Tsong Yueh and Yoo, Shin and Harman, Mark. "Provably Optimal and Human-Competitive Results in SBSE for Spectrum Based Fault Localisation." the 2013 International Symposium on Search Based Software Engineering, Springer Berlin Heidelberg, 2013, pp. 224-238.
- [48] Xiaoyuan Xie, Tsong Yueh Chen, Fei Ching Kuo and Baowen Xu. "A theoretical analysis of the risk evaluation formulas for spectrum-based fault localization." *Acm Transactions on Software Engineering and Methodology (TOSEM)*, 2013.